2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)

Near-Memory and In-Storage FPGA Acceleration for Emerging Cognitive Computing Workloads

(Invited Paper)

Ashutosh Dhar*, Sitao Huang*, Jinjun Xiong*†, Damir Jamsek§,
Bruno Mesnet§, Jian Huang*, Nam Sung Kim*, Wen-mei Hwu*, Deming Chen*
*The C3SR Center, University of Illinois, Urbana-Champaign, †IBM Research, §IBM Systems
{adhar2, shuang91, jianh, nskim, w-hwu, dchen}@illinois.edu {jinjun, jamsek}@us.ibm.com bruno.mesnet@fr.ibm.com

***Abstract*—The slow down in Moore's Law has resulted in poor scaling of performance and energy. This slow down in scaling has been accompanied by the explosive growth of cognitive computing applications, creating a demand for high performance and energy efficient solutions. Amidst this climate, FPGA-based accelerators are emerging as a potential platform for deploying accelerators for cognitive computing workloads. However, the slow-down in scaling also limits the scaling of memory and I/O bandwidths. Additionally, a growing fraction of energy is spent on data transfer between off-chip memory and the compute units. Thus, now more than ever, there is a need to leverage near-memory and in-storage computing to maximize the bandwidth available to accelerators, and further improve energy efficiency. In this paper, we make the case for leveraging FPGAs in near-memory and in-storage settings, and present opportunities and challenges in such scenarios. We introduce a conceptual FPGA-based near-data processing architecture, and discuss innovations in architecture, systems, and compilers for accelerating cognitive computing workloads.**

***Keywords*-Near Memory Acceleration, In-Storage Acceleration, NMA, FPGA, Cognitive Computing, Deep Learning**

I. Introduction

The last few decades have witnessed massive leaps in computing performance and energy efficiency. These advances have been greatly aided by rapid developments of semiconductor technology, with Moore's Law delivering not only shrinking transistor sizes, but also performance and energy scaling with each new generation. These developments allowed general-purpose processors (i.e., CPUs) to deliver programmability and performance for the applications at hand. However, the end of Dennard's eponymous scaling law and the slow down, and eventual end, of Moore's Law have forced us to re-examine our approach to the design of computer systems. Designers of general-purpose processors are now not only faced with stringent power and thermal budgets, but also diminished returns from successive generations of CMOS technology due to increased complexity and costs of transistor scaling, and difficulty in extracting ILP (Instruction Level Parallelism). The impact of these challenges is further aggravated by the emergence of *Cognitive Computing*. Cognitive computing seeks to mimic human thinking, relying on machine learning, speech and vision processing, and most recently *Deep Learning*. These tasks continue to grow in computational, memory, and storage complexity, further exacerbating the slow down in general purpose performance.

The need for increased computational performance and sensitivity to energy efficiency have prompted a paradigm shift in system design, with architects looking to off-chip *accelerators* to provide application or domain-specific solutions. While ASICs (Application-Specific Integrated Circuits) provide the greatest specialization and energy efficiency, GPUs (Graphics Processing Units) offer a more programmable solution with high computational performance. The tradeoff is that ASICs incur high NRE (Non-Recurring Engineering) costs, and GPUs offer limited specialization and reduced energy-efficiency. However, the algorithms and techniques at the heart of Cognitive Computing tasks continue to evolve rapidly. Thus, we must find a balance between programmability, specialization, performance, and power. Reconfigurable architectures such as FPGAs (Field Programmable Gate Arrays) are gaining traction fast towards finding this *one-size-fits-all* solution. The ability to reconfigure FPGAs provides a unique advantage, allowing specialized circuits to be deployed for each task or application, while still delivering high performance and energy efficiency.

While the performance of accelerator platforms have managed to keep up with the computational requirements of Cognitive Computing, inadequate memory and I/O performance can throttle their performance. Consider, for example, an Nvidia Volta GPU with 14.03 TFLOP/s of single-precision throughput, and 900 GB/s of memory bandwidth. In order to achieve peak computational throughput, the GPU needs to reuse each operand, fetched from memory, 62.3 times [1]. Similarly, a Xilinx VU9P Virtex UltraScale+ FPGA can achieve a peak single-precision throughput of 7.3 TFLOP/s, and can leverage two HBM2 stacks of memory for up to 512 GB/s of memory bandwidth[1]. We would need a reuse factor of at least 57 in order to achieve peak throughput. Thus, the ability to fully leverage these accelerators is limited by the application data-reuse characteristics and its memory bandwidth. However, in the post-Moore era,

[1]VU9P has 12288 DSPs, and we assume 3 DSPs per floating point MAC unit, operating at F_{MAX} of 891 MHz. Each MAC performs two operations-multiple and add. HBM2 has a peak bandwidth of 256 GB/s per stack.

978-1-7281-3391-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ISVLSI.2019.00021

traditional processor-memory interfaces are expected to stop scaling [2], limiting scaling of memory bandwidth. Similarly, limitations of I/O bandwidth over interfaces such as PCIe can further exacerbate the problem of operand delivery. Not only is operand delivery to the FPGA limited by PCIe bandwidth, but in the case of data-intensive applications such as Cognitive Computing, loading datasets from storage devices is also constrained by traditional I/O channels. The discrepancies in I/O and memory bandwidths are illustrated in Fig. 1. We consider a system with an NVMe based storage solution, hosted on a $4\times$ PCIe 3.0 [2] channel from which the dataset must first be loaded into the host system's memory (1), transferred to the FPGA's memory over another $16\times$ PCIe 3.0 channel (2). Once these have been completed, the FPGA can load operands on demand from its memory (3).

Simultaneously, the cost of moving data from off-chip devices continues to account for a significant portion of the power budget. Recent studies estimate the cost of fetching data from off-chip DRAM 28 to 45 pJ/bit (40nm). In contrast, reading a 16bit operand from a 32K-word SRAM array costs 11pJ only [3]. Thus, the growing demands for energy efficiency, memory bandwidth, and fast access to large amounts of data have prompted researchers to explore placing general-purpose compute and accelerators closer to memory and storage, in the form of near memory acceleration [4, 5] and in-storage computing [6–8]. The proliferation of these solutions has been, however, slow, limited by technology challenges in integrating computing and memory, program transparency, flexibility, and the sheer complexity of these systems. For the rest of this work, we will refer to near-memory and in-storage computing collectively as Near-Data Processing (NDP).

In this paper, we submit that advances in key enabling technologies across compilers, memories, and systems have created opportunities to leverage the flexibility, performance, and efficiency of FPGA for NDP. In the rest of this work, we present the enabling technologies, and explore the opportunities for FPGA+NDP across compilation, systems, and architectures for Cognitive Computing applications. We make a case for leveraging FPGA+NDP in training cognitive computing models, and discuss a conceptual architecture that unifies near-memory and in-storage computing solutions in a scalable fashion. We then discuss the challenges and opportunities in a new High-Level Synthesis (HLS) engine for the proposed system, along with system-wide collaborative computing.

II. Towards FPGA based Near Data Processing

In this section we present the advances in algorithms, architectures, and devices that have made the confluence of FPGAs and NDP possible. In particular, we will focus on the challenges and opportunities that have been opened.

[2]Current off-the-shelf NVMe drives are unable to fully saturate the bandwidth of a $4\times$ PCIe 3.0 channel.

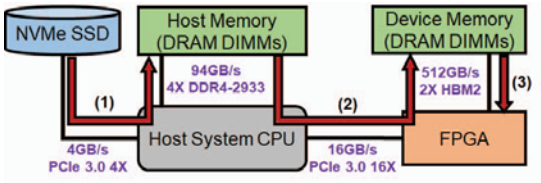

Figure 1. Representative system with I/O and memory bandwidths

A. Enabling Technologies

Cognitive Computing:
In this paper we focus on the domain of Cognitive Computing. Limiting our discussion to a single domain allows us to focus on a more specialized solution. However, the wide breadth of applications across the domain demonstrate great diversity in computational and memory patterns, memory capacity requirements, latency constraints, and even algorithmically. While cognitive computing leverages several families of algorithms and kernels, *Deep Learning* has proven to be the most successful recently, and is also the most challenging in its requirements. Running real-time inference of a cognitive model requires loading large models, and performing several hundreds of Giga operations, all while operating under a tight power budget. On the other hand, training models involves orders of magnitude greater number of operations, and requires loading and processing several hundred Gigabytes of training data. Even with the field of Deep Learning, each model architecture can present remarkably different requirements. For example, models heavy with convolutional layers behave as compute-bound tasks, whereas recurrent layers can often be memory-bound. While these variations are a challenge to designers, they also offer ample opportunities for innovation, as we shall discuss in the next section.

Memory and Storage:
Host DRAM memory is typically packaged in DIMMs (Dual In-line Memory Modules), with individual DRAM chips being organized on the DIMMs into a hierarchy of *Ranks*. Multiple DIMMs can be placed on a single memory channel, with the host processor providing a memory controller per channel. Early attempts at near memory acceleration (NMA) and processing-in-memory (PIM) attempted to integrate compute units on the same die as DRAM, which was fraught with many challenges. However, more recent attempts have looked to integrate accelerators in the memory buffers of LRDIMMs [9], or close to the DRAM chips on the DIMM [10]. This enables the use of standard DRAM technology, while maintaining the existing LRDIMM architecture.

Recent interest in NMA has been driven by the development of stacked DRAM organizations such as Hybrid Memory Cubes (HMC) and High Bandwidth Memory (HBM). HBM and HMC rely on 3D integration to stack DRAM dies above a logic die, and designers may use 2.5D integration

to couple them with host processors. The presence of this logic die has been the focus of several recent works, with researchers looking to integrate compute in the logic level, particularly in HMC [4, 5]. While similar in spirit, HBM and HMC have different approaches and targets. HMC is designed to be connected to a host processor and be chained together to provide higher capacities. HBM is a wide-I/O architecture, designed to provide high bandwidth for devices like GPUs. The logic layer in HMC contains memory controllers for the DRAM banks in the stacks, and uses a high speed serialization/deserialization (SerDes) interface to the host processor, which can result in higher latencies and idle power [9]. In contrast, HBM logic dies include only the DRAM PHY, TSVs, and test logic, and the memory controllers are present on the host processor. Additionally, HBM stacks are coupled with the hosts via 2.5D integration. This provides us with opportunities to integrate accelerators, such as FPGAs, in the logic die without changes to the host, which we discuss later in this paper.

Solid state drives (SSD) are fast gaining market share with their increasing capacities and high speeds. A typical SSD is comprised of an SSD controller, a DRAM buffer, and a NAND flash array organized across multiple channels. The controller is comprised on an embedded processor, cache controller for the DRAM buffer, and channel controllers [8]. The high speed of the flash chips, the presence of the embedded processors, and the increasing demand for processing large datasets have led to interest in near-storage processing. These in-storage processing units are typically embedded CPUs [8], but recent attempts have leveraged FPGAs as well [6, 7]. These approaches, however, do not fully leverage the FPGA's ability to reconfigure itself and provide truly custom solution, and do not focus on Cognitive Computing. The ability to bring memory and storage closer, while leveraging the reconfigurability of FPGAs is a unique and rich angle, which we will focus on later in this work.

I/O and Communication Channels:

While memory and storage advances have enabled the placement of compute closer to data, shuttling data between the host processor and these accelerators remains a challenge, as discussed earlier. Memory channels, such as DDR, have tight timing requirements, which makes any modification of in the datapath difficult. In addition, they do not allow for non-memory units to access these high-bandwidth and low-latency paths. Similarly, PCIe interfaces offer high latencies and do not allow for arbitrary lane organization to maximize bandwidth. Finally, both interfaces have no support for data coherence. Getting around coherency issues would involve overheads of explicit copies, locks, and critical sections. Coherent I/O interfaces such as IBM's CAPI are an elegant solution to these issues. However CAPI is still reliant on PCIe for data-movement. The introduction of the OpenCAPI [11] standard, however, creates new avenues to explore. OpenCAPI enables a high bandwidth, low latency

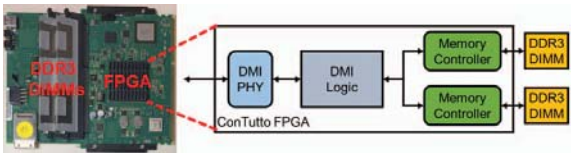

Figure 2. The Contutto experimental platform

interface to memories, storage devices, and accelerators, while also handling coherency. Thus, we now have a solution to integrate NDP accelerators in a systematic, scalable and non-disruptive manner, natively.

NDP Platforms:

The high development cost of NMA systems has often been a barrier to entry. The ability to leverage off-the-shelf components would enable faster adoption of NDP. The proposed experimental Contutto system [12] provides this opportunity. Designed for IBM's proprietary DMI memory interface, Contutto places an FPGA in between the host memory interface and the DRAM DIMMs. Thus, Contutto provides an opportunity to deploy NMA accelerators in a real system, with no modification to the host system or the DIMMs. Towards this goal, we successfully leveraged Contutto to demonstrate a novel architecture that provided application-transparent NMA [10]. Since the FPGA has to interact with the memory through the limits of the DIMM organization, we are unable to fully leverage the benefits of NMA. However, we envision future revisions of Contutto that provide closer coupling with the DRAM chips directly. Note that this is a less costly way of integrating FPGAs in the memory system than embedding them in the logic layers of HBM/HMC, or even 2.5D package integration. In addition, the use of new I/O interfaces such as OpenCAPI and OpenPOWER memory bus [13] can help create tighter integration between the updated Contutto platform and the host system.

High-Level Synthesis:

A key challenge to the widespread adoption of FPGAs is their programmability. The high degree of specialization of FPGAs requires the development of custom circuits for the accelerator. Traditionally, this would have involved writing RTL (Register-Transfer Level) code in Verilog/VHDL, and would take several months of design and verification time. Alternatively, an overlay architecture could be developed. These overlay architectures are generic accelerators for a narrow domain of applications and tasks, but require a one-time development cost for a range of applications and tasks. However, this deprives us of all the advantages of FPGAs. Recently, High-Level Synthesis (HLS) has emerged as a solution to these programmability woes. HLS transforms application code written in high-level languages like C/C++ into RTL automatically. In doing so HLS significantly speeds up the development and verification of customized solutions [14]. Thus, designers can quickly adapt to changes in algorithms and hardware by rewriting their high-level

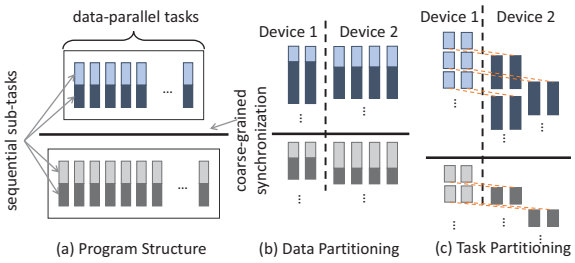

Figure 3. Program with Many Data-parallel Tasks (a) and Two Collaborative Execution Strategies: (b) Data Partitioning and (c) Task Partitioning

code, and synthesizing a new IP. When combined with the reconfigurability of the FPGA, HLS enables designers to fine-tune accelerators to the exact requirements of their applications and tasks. As a case study, consider a Long-term Recurrent Convolutional Network (LRCN) deep neural network in [15]. With the help of HLS, we were able to deliver a solution $4.65\times$ faster and $17.5\times$ more energy efficient than a general purpose processor.

However, modern HLS solutions require a significant amount of guidance from the designer in order to generate efficient solutions [14], and the compilers make implicit assumptions about the target architecture. In order to leverage FPGAs in NDP scenarios, we must generate accelerator IPs optimized to tolerate the idiosyncrasies of these systems. For example, traditional accelerator development attempts to keep as much data as possible on the FPGA chip itself. However, in NDP scenarios we must leverage the significantly higher memory bandwidth that is made available. Similarly, atomic operations have traditionally been of limited interest for HLS users. However, leveraging atomic operations is a very powerful feature of NDP. Thus, in order to enable FPGA+NDP systems, we see several opportunities to innovate in the HLS compiler stack.

Collaborative Computing:
With the introduction of all these heterogeneous computing elements, we must consider how to orchestrate and leverage all the compute capabilities of different components in such complicated system. Collaborative computing [16] was proposed as a concept and guideline on how to improve the efficiency of programming and optimizing heterogeneous systems. Collaboration between devices is achieved by partitioning compute tasks among computing devices. Fig. 3 illustrates how a typical parallel program structure can be mapped to two different collaboration computing schemes: data partitioning scheme and task partitioning scheme.

In an NDP scenario, compute units and storage devices are much closer to each other, compared to the traditional systems. With this close integration, processing units are more aware of low-level data storage and accessing patterns, which enables more close collaboration opportunities between devices. Thus, at a task-level and system-level, several potential collaboration opportunities can be exploited.

III. An NDP Approach to Cognitive Computing

In this section we take a deeper look at cognitive computing workloads, and discuss a new approach to cognitive computing with FPGA-based near data processing. We outline the features, challenges, and opportunities of a new NDP-based heterogeneous architecture for cognitive computing, a new HLS engine for NDP, dynamic reconfiguration for NDP, and collaborative computing.

As discussed in the previous section, emerging cognitive computing workloads exhibit diverse behaviors. For the rest of this section, we will draw a line between inferencing and training workloads, and consider the variation within each of them. To begin with, let us consider inference workloads. Inferencing is sensitive to latency and energy. At run-time, a batch of images, speech segments, text etc. must be fed in to an inference model. These live inputs could be sourced in real-time from I/O-attached devices or from disk. Raw data from sensors, networks, or disks is traditionally loaded into host memory, before being processed locally or offloaded to an accelerator. Hence, there are several opportunities to leverage NDP. First, by placing accelerators at the sensors themselves. However, this approach has several drawbacks: (1) It can increase the cost of these I/O devices, (2) Sensors have tight power budgets that limit how much processing we can perform, (3) Sensors continue to evolve and tend to be replaced or updated frequently. Thus, we note that there are two locations for accelerators in these scenarios: the memory, and the storage device.

Next, we consider training workloads. Training is a computational and memory intensive workload. In the case of Deep Neural Networks (DNNs), training via backpropagation requires performing three passes over the entire DNN model, beginning with a forward pass (same as inference), and then two additional backward passes to compute activation and weight gradients. Thus, on an average, training involves $3\times$ as many operations per input sample, as compared to inference, and significantly greater number of activations as well. Furthermore, training DNNs involves performing this forward-and-backward pass on hundreds of thousands of samples. Due to the forward-and-backward nature of the task, training often requires much larger working sets in memory, and has significantly stronger data-dependencies that limits parallelization. The process of training a cognitive computing model is further complicated with use of distributed training routines that spreads out training workload across multiple accelerators. The combination of global communication and synchronization due to distributive training, the higher computational workload, the larger demands for memory capacity and bandwidth, and the complex data-flow dependencies make training cognitive models a much more challenging task. However, the memory and data oriented nature of the task makes it ideal for NDP. Furthermore, the high overheads of distributed

Table I
A COMPARISON OF POPULAR DNN ARCHITECTURES

		GoogleNet	Resnet-50	Deep Speech 2
Depth		27	52	11
Conv	# Layers	21	49	3
	BatchNorm	No	Yes	Yes
	# Weights	5.67M	20.40M	3.15M
	# Activations	4.27M	8.67M	325M
FC	# Layers	1	1	1
	# Weights	1M	2M	2.95M
	# Activations	1K	1K	1.72K
RNN	# Layers	N/A	N/A	7
	# Weights	N/A	N/A	28.9M
	# Activations	N/A	N/A	12.04K
Total Params		6.64M	25.6M	35M
Total Ops		1.47G	3.8G	65G

training can be minimized if we distribute work across multiple NDP modules. The memory and upcoming high-speed I/O interfaces like OpenCAPI can enable significantly faster distributed computing, as we demonstrated earlier with Memory Channel Networks (MCN) [10]. Thus, we submit that NDP can be a powerful tool for both inference and training workloads in cognitive computing.

While training and inference have their own unique requirements, difference in cognitive computing models can contribute a great degree of variation as well. Table I lists a few popular DNN models[3]. Note the wide variation in depths, operations, parameters etc. across these models. However, even models with a similar number of operations or parameters can vary significantly based on the types of operations. For example, while convolutional layers are compute-bound operations, Recurrent Neural Network (RNN) layers involve memory-bound matrix-vector operations. Thus a Convolutional Neural Network (CNN) and a RNN with similar number of operations or parameters can have very different requirements. The effect of normalizations and element-wise operations can add another layer of complexity. Consider Resnet-50 in Table. I, each convolution is followed by a batch normalization operation. During forward propagation, activations must be read and normalized using pre-computed means and variance. However, during backpropagation, the mean and variance of the activations must be computed on the fly. This requires reading all the activations twice, in order to compute mean and variance, and then reading the activations once more to normalize them. Thus, across forward and backward propagation, the activations are read four times, and written back twice. So in a model like Resnet-50, compute-bound convolutions are interleaved with memory-bound batchnorms during training.

Fortunately, the list of fundamental operations in cognitive computing is limited and well known, such as: convolutions, GEMM (General Matrix-Matrix Multiplication), GEMV (General Matrix-Vector Multiplication), batch normalizations, element-wise operations, activation functions

[3]Estimates for Deep Speech 2 based on hidden state and input sequence size of 1280.

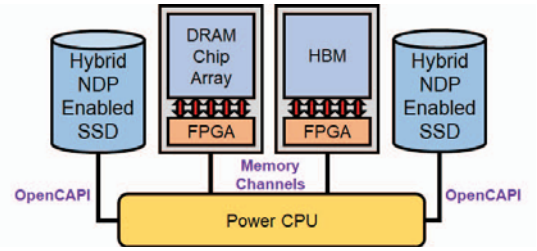

Figure 4. Full system with heterogeneous compute, memory, storage, and NDP

etc. Combined with the flexibility of FPGAs and the rapid IP development via HLS, deploying highly customized solutions for both training and inference can be achieved with FPGA-based NDP.

A. Heterogeneous NDP Computing Architecture

Towards our goal of leveraging FPGA-based NDP for cognitive computing, in this section we present a conceptual heterogeneous NDP computing system. With a wide variation of memory bandwidth, memory capacity, and compute requirements for cognitive computing, we envision a heterogeneous system comprised of various types of storage and memory elements, each coupled with FPGA-based computing. We propose enabling NDP across the whole system from host memory to storage devices.

We begin by examining two options for near memory computing. First, as alluded to earlier, we consider the possibility of coupling high bandwidth memory (HBM) and FPGAs. While the current HBM architecture has some space on the logic die to integrate a custom FPGA solution, it may not be enough to fully leverage the benefits of NMA. In this case, we could consider a new architecture, based on HBM, with a larger logic die for FPGA based NMA. However, modern FPGAs are significantly larger than the existing HBM logic dies. Thus, this could be a very invasive approach and would require significant engineering effort and re-design. Alternatively, we propose a more conservative approach: 2.5D integration. HBM natively supports 2.5D integration, thus enabling a single package of FPGA and DRAM tightly coupled together. Interfacing the HBM+FPGA package to the host processor can be enabled via OpenCAPI [11], OpenPOWER memory bus [13], or IBM's DMI interface [12]. The relatively longer latency of HBM, however, may not be ideal for host processor's main memory. Thus, our second option revisits traditional DRAM DDR-DIMMs. We propose leveraging a new Contutto-like [12] like design with closer coupling between the FPGA and the DRAM chips, eliminating the DIMM abstraction and improving bandwidth utilization. Furthermore, we can leverage both architectures in a single system, providing high bandwidth and low latency memories for complete flexibility, as shown in Fig. 4.

Next, we consider the storage subsystem. The embedded processor in the SSD controller is responsible for running the

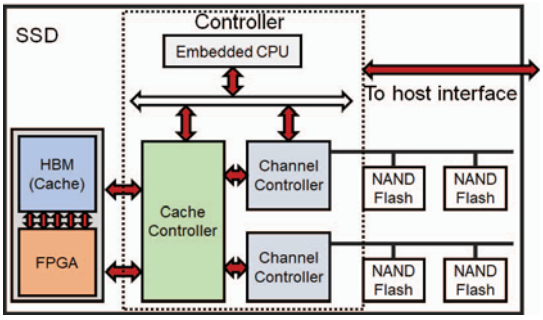

Figure 5. Proposed hybrid NDP SSD system

SSD firmware and managing operations such as interfacing with the host, garbage collection, and address translation. These processors may experience enough idle time so that they may be used for computation [8]. However, they may not sufficient for cognitive computing workloads. It may be possible to add more powerful embedded CPUs, but these may not be very power efficient. Thus, an FPGA is a much more compelling solution. Here we note that a DRAM cache is already part of the datapath of the SSD [8]. Thus, instead of adding an FPGA or embedded processor in the main controller, we propose replacing the DRAM cache with an FPGA-based NMA-enabled DRAM unit, such as the HBM+FPGA module we described earlier. Thus we introduce in-storage computing via near-memory computing! This is a potentially less disruptive approach that allows us to leverage the vast experience of NMA research in the new area of in-storage computing. By placing the compute in the DRAM-cache, the existing channel and NAND controller do not need to be modified, and a high bandwidth cache will help leverage the internal bandwidth across channels. In order to perform in-storage computing, the existing controller need simply load blocks from the NAND units into the cache. By performing computation in the SSD's cache, we limit potential coherence issues as well. Finally, it reduces an additional level of data movement in the SSD from the DRAM cache to an accelerator, over the controller bus, and then to the host or back to the DRAM cache. Fig. 5 illustrates our architecture.

B. NDP For DNN Training and Inference

FPGAs have traditionally been relegated to inference workloads. The lack of computational performance, the limited bandwidth and capacity of memory, and the programming model have been the key barriers to entry for FPGAs in the training world. In addition, limited library support in popular machine learning and deep learning frameworks exacerbates the problem. With the above described architecture, we can now realize scalable distributed training. With multiple FPGAs spread across the system, connected either via coherent-I/O interfaces or memory channels, we have a network of FPGAs with large high-bandwidth memories attached to them. In addition, close proximity to the storage devices enables even faster loading of training sets to the devices. Existing libraries can be leveraged with the help of application-transparent approaches like Memory Channel Networks [10]. Finally, the availability of FPGAs in the distributed training system, enables us to perform in-network acceleration as demonstrated previously in [17, 18].

Another opportunity presents itself in the form of dynamic reconfiguration of the FPGAs. Leveraging reconfiguration to tune the architecture of a platform has been shown to be feasible and capable of improving energy efficiency [19]. Similarly, dynamic reconfiguration of the FPGA can enable the tuning of accelerator architectures on the fly, adapting the training infrastructure to better fit the target model. For example, we can tune the architecture for CNNs instead of RNNs, by creating wide GEMV engines, rather than GEMM engines. In particular, we can reconfigure the on-chip memory organization of the accelerators, which can significantly impact the end-to-end training performance.

Finally, with the help of dynamic reconfiguration, we can explore dynamically adjusting the precision of the compute engines. As the training process continues, the degree of precision increases, with the gradient descent making smaller and more fine-tuned adjustments as it approaches convergence. However, earlier stages of the training can be potentially sped up by sacrificing precision. By dynamically reconfiguring to lower precision, we can instantiate a greater number of compute units, and we reduce the required memory bandwidth as well. In a NMA scenario, this can significantly boost performance.

Similarly, low-precision is even more advantageous in inference scenarios. Many DNN inference tasks have been shown to work well with low-bitwidth integers or fixed-point numbers. The Project BrainWave at Microsoft adopts low-bitwidth block floating-point format to represent numbers in deep learning accelerator [20]. These formats, referred to as ms-fp8 and ms-fp9, use only 8 bits and 9 bits, efficiently packed into reconfigurable resources on FPGAs, while preserving most of the accuracy in deep learning inference tasks. Near-memory or in-storage processors can convert the number format efficiently and transparently, which can significantly reduce the cost of customized number format conversion overhead. Besides, the storage layout can also be customized according to the number format so that the data access efficiency can be maximized. Reduced precision also reduces the required bandwidth of the application, allowing the NDP accelerators to better use the available internal bandwidth, and deliver higher performance.

C. Leveraging FPGA Reconfiguration for Scalable NDP

So far we have looked at leveraging reconfiguration capabilities of the FPGA from an application perspective. However, at a system level, reconfiguration can provide powerful features as well. Most importantly, the reconfigurability of the FPGA can potentially enable an FPGA-

based NDP module be highly portable since adapting to new interfaces and protocols is greatly simplified. Thus, we can potentially port NDP modules across platforms, systems, ISAs etc by reconfiguring the FPGA with the appropriate protocols, interfaces etc. For example, memory interfaces and protocols tend to last for several years and can be leveraged by several generations of processors, each of which may have new ISAs, new ISA features, or coherence protocols. Programmability and compatibility with the host architecture has been a key challenge for NMA. However, we can circumvent these challenges, while maintaining the same NMA architecture and module, by simply instantiating a new controller or interface on the FPGAs.

D. High Level Synthesis for NDP

With a scalable NDP system in place, along with a target group of applications and tasks, we turn our attention to the programmability of FPGAs. As discussed earlier, HLS provides us with a great opportunity to simplify the programmability of FPGAs by automatically transforming high level code to RTL. However, modern HLS solutions still require a significant amount of guidance from the designer in order to generate efficient solutions [14], and the compilers make implicit assumptions about the target architecture. Thus, we need an HLS engine that is aware of the idiosyncrasies of our NDP system.

First, a cognitive computing specific HLS engine enables us to simplify the designers job since the set of operations are well understood. HLS engines can leverage template designs of fundamental operations, allowing the designer to perform fast design space exploration (DSE) to find an optimal design for the target model. Using predefined templates also helps speed up the verification process. Second, cognitive computing tends to have deterministic compute and memory patterns, which opens up new avenues for the HLS engine's scheduler. The deterministic nature of the kernels allows the compiler to reorder loops, and thus memory accesses, potentially creating more accelerator architectures that are better tuned for the available resources. In particular, new ways of buffering operands and intermediate results can be explored aggressively, since the loop design space is well defined. Third, current HLS engines do not consider whether the target FPGA is being used in an NDP scenario, which can potentially result in wasted performance. While the key advantage of NMA is the increased bandwidth available, this additional bandwidth can be wasted if it is not leveraged correctly. We note that the deterministic nature of cognitive computing can potentially allow the HLS engine to schedule off-chip accesses to the memory in a manner that is far more bandwidth efficient. By providing the HLS engine with details on the banking, organization, page sizes, etc we can ensure that the sequence of loads and stores is highly tuned to the target memory system. For example, the HLS engine's compiler can perform loop unrolling, reordering, and scheduling in a fashion that increases the hit-rate in the DRAM row-buffers and work on improved page-locality. Finally, towards our goal of distributed training on our system, our new HLS engine needs to be able to support atomic operations and global collectives like scatter, gather, reduce, etc. The end result of all these specializations is optimal leveraging of a tuned infrastructure, algorithm, accelerator architecture, and communication.

E. NDP Systems and Collaborative Computing

Placing FPGAs across the memory and storage systems requires careful consideration of the area, performance, and power constraints. For example, DIMMs may have a peak power limit, while 2.5D integration may have area constraints. The logic to implement memory and I/O controllers may take up a large amount of the FPGA's resources, limiting how much computational performance can be obtained from the FPGA. Also, the size of the FPGA affects not only its performance, but also the power efficiency, compile and reconfiguration times, with larger FPGAs requiring longer times to synthesize, place and route, and reconfigure. In some scenarios, then, an array of small FPGAs can potentially be beneficial, since they can be quickly reconfigured in parallel, deliver high aggregate I/O bandwidth, and even more fine-grained customization. Hence, it is possible, that our system may have different types of memories, storage devices, and FPGAs of different sizes.

The final piece of our puzzle is distributing work across all the NDP enabled memories and storage units in the system. We have defined their architecture, interfaces, and programmability, and now focus on collaborative computing in this system. In the NDP scenarios, compute units and storage devices are much closer to each other compared to the traditional systems. With this close integration, processing units are more aware of low-level data storage and accessing patterns, which enables more close collaboration opportunities between devices. In addition, the high memory capacity and bandwidth provided by NDP amplifies the benefits from collaborative computing.

Several potential collaboration opportunities can be exploited in the NDP context as follows. First, the host processors and storage devices have processing capabilities, of varied performance. This enables hybrid and fine-grained collaboration at both compute device side and storage device side. Host processors typically have strong branch and complex task processing capabilities, while in-storage accelerators feature distributed, fine-grained streaming processing capabilities. Computing tasks can then be partitioned and assigned according to the nature of the tasks, and collaboration can happen in a hybrid data partitioning and task partitioning manner. Second, with the high memory bandwidth provided by NDP, data movement between compute devices becomes much cheaper. This allows dynamic workload balancing between devices, which is the key to achieving high effi-

ciency in a heterogeneous system. Third, unified memory and system-wide atomic operations will be more accessible and therefore more applications can benefit from using fine-grained task-partitioning collaboration scheme. Overall, we see a rich environment to explore collaborative computing in various shapes and forms.

IV. Conclusion

Advances in memory and storage devices, I/O interfaces, compilers, and FPGAs have enabled the confluence of near data processing and FPGAs. In the era of cognitive computing, FPGA-based NDP solutions offer a balance between specialization, performance, energy-efficiency, and programmability. Thus, in this work we presented our vision of enabling FPGA-based NDP across the system. We presented a conceptual heterogeneous system architecture that places FPGAs across all levels of the memory and storage system. We discussed FPGA-based architectures for leveraging NMA in 2.5D integrated HBM modules, as well as traditional DRAM based modules, and presented a scalable way to add in-storage processing in SSD devices by introducing an FPGA-based NMA processor in the SSD's DRAM cache. With the help of distributed processing across all the NDP modules, we envision this system as a powerful and energy-efficient solution for training and inference of cognitive computing models. The architecture also opens up novel research directions in application tuning, accelerator development, high level synthesis compilers, and system-wide collaborative computing. Finally, we discussed the unique advantages that the FPGA's reconfiguration abilities provide. Towards our goal of a system-wide heterogeneous NDP architecture, we have begun to explore all of above opportunities and have found scenarios for the future of FPGA-based NDP for cognitive computing.

Acknowledgments

This work is supported in part by (1) IBM-ILLINOIS Center for Cognitive Computing Systems Research (C3SR) - a research collaboration as part of the IBM AI Horizons Network, (2) CRISP, one of six centers in JUMP - a Semiconductor Research Corporation (SRC) program sponsored by DARPA, and (3) Defense Advanced Research Projects Agency (DARPA) contract FA8750-18-2-0108, under the DARPA MTO Software Defined Hardware program.

References

[1] W. W. Hwu, I. El Hajj, S. Garcia de Gonzalo, and et al., "Rebooting the data access hierarchy of computing systems," in *2017 IEEE International Conference on Rebooting Computing (ICRC)*.

[2] N. S. Kim, D. Chen, J. Xiong, and W. W. Hwu, "Heterogeneous computing meets near-memory acceleration and high-level synthesis in the post-moore era," *IEEE Micro*, 2017.

[3] A. Pedram, S. Richardson, and M. H. et.al., "Dark memory and accelerator-rich system optimization in the dark silicon era," *IEEE Design Test*, 2017.

[4] M. Gao and C. Kozyrakis, "HRL: Efficient and flexible reconfigurable logic for near-data processing," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[5] D. Kim, J. Kung, S. Chai, and et al., "Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*.

[6] S. Seshadri, M. Gahagan, S. Bhaskaran, and et al., "Willow: A user-programmable SSD," in *the 11th USENIX Conference on Operating Systems Design and Implementation*.

[7] S.-W. Jun, M. Liu, S. Lee, and et al., "BlueDBM: An appliance for big data analytics," in *42nd Annual International Symposium on Computer Architecture*, 2015.

[8] H. Choe, S. Lee, and S. P. et al., "Near-data processing for machine learning," *CoRR*, vol. abs/1610.02273, 2016.

[9] H. Asghari-Moghaddam, Y. H. Son, J. H. Ahn, and N. S. Kim, "Chameleon: Versatile and practical near-dram acceleration architecture for large memory systems," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[10] M. Alian, S. W. Min, H. Asgharimoghaddam, and et al., "Application-transparent near-memory processing architecture with memory channel network," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[11] IBM, "OpenCAPI." [Online]. Available: https://opencapi.org/

[12] B. Sukhwani, T. Roewer, C. L. Haymes, and et al., "ConTutto: A novel FPGA-based prototyping platform enabling innovation in the memory subsystem of a server class processor," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017.

[13] IBM, "OpenPOWER memory bus." [Online]. Available: https://openpowerfoundation.org/

[14] K. Campbell, W. Zuo, and D. Chen, "New advances of high-level synthesis for efficient and reliable hardware design," *Integration, the VLSI Journal*, 2017.

[15] X. Zhang, X. Liu, A. Ramachandran, and et al., "High-performance video content recognition with long-term recurrent convolutional network for fpga," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*.

[16] S. Huang, L.-W. Chang, I. E. Hajj, and et al., "Analysis and modeling of collaborative execution strategies for heterogeneous CPU-FPGA architectures," in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*.

[17] Y. Li, I.-J. Liu, D. Chen, and et al., "Accelerating distributed reinforcement learning with in-switch computing," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*.

[18] Y. Li, J. Park, M. Alian, and et al., "A network-centric hardware/algorithm co-design to accelerate distributed training of deep neural networks," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018.

[19] A. Dhar and D. Chen, "Efficient GPGPU computing with cross-core resource sharing and core reconfiguration," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, April 2017, pp. 48–55.

[20] J. Fowers, K. Ovtcharov, M. Papamichael, and et al., "A configurable cloud-scale DNN processor for real-time AI," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*.