Generalize or Die: Operating Systems Support for Memristor-based Accelerators

Pedro Bruel*, Sai Rahul Chalamalasetti†, Chris Dalton‡, Izzat El Hajj§, Alfredo Goldman*, Catherine Graves†
Wen-mei Hwu§ FIEEE, Phil Laplante¶ FIEEE, Dejan Milojicic† FIEEE, Geoffrey Ndu†, and John Paul Strachan†
†Hewlett Packard Labs, Palo Alto, CA; emails: [first-name.last-name]@hpe.com
‡HP Labs, Bristol, UK; emails: cid@hp.com
¶Penn State, Malvern, PA; email: pal11@psu.edu
§UIUC, Urbana Champaign, IL; emails: elhajj2@illinois.edu, w-hwu@illinois.edu
*University of Sao Paulo, Sao Paulo, Brazil; emails: phrb@ime.usp.br, gold@ime.usp.br

***Abstract*—The deceleration of transistor feature size scaling has motivated growing adoption of specialized accelerators implemented as GPUs, FPGAs, ASICs, and more recently new types of computing such as neuromorphic, bio-inspired, ultra low energy, reversible, stochastic, optical, quantum, combinations, and others unforeseen. There is a tension between specialization and generalization, with the current state trending to master slave models where accelerators (slaves) are instructed by a general purpose system (master) running an Operating System (OS). Traditionally, an OS is a layer between hardware and applications and its primary function is to manage hardware resources and provide a common abstraction to applications. Does this function, however, apply to new types of computing paradigms?**

This paper revisits OS functionality for memristor-based accelerators. We explore one accelerator implementation, the Dot Product Engine (DPE), for a select pattern of applications in machine learning, imaging, and scientific computing and a small set of use cases. We explore typical OS functionality, such as reconfiguration, partitioning, security, virtualization, and programming. We also explore new types of functionality, such as precision and trustworthiness of reconfiguration. We claim that making an accelerator, such as the DPE, more general will result in broader adoption and better utilization.

I. Introduction

Advances in computing are often represented by fixed, application specific hardware. For example, one of the first general purpose computers, The Electronic Numerical Integrator and Computer (ENIAC), required re-wiring of a plug board to program it for each application [16]. Likewise, Systolic, Wavefront [24], and Dataflow [21] architectures each needed to be reconfigured for a specific computation, such as matrix multiplication or convolution. Even Field Programmable Gate Arrays, though reconfigurable, need sophisticated operating system (OS) support to act as anything more than a fixed system during operation [35]. Throughout the evolution of these computing solutions, OS support has been very scarce, relegating these devices to act as early accelerators and not as general purpose computers. Even sophisticated co-processing devices, such as Graphics Processing Units (GPU), have highly specialized instruction sets that render their use as general purpose computing resources difficult.

Conversely, OS designers have not easily adapted to the changes in processing paradigms and increasing importance of power, largely building systems that rely on a task-based workload model. Although there have been many notable attempts at building OSes for accelerators, none of these have been completely successful [25].

In this paper we explore the OS functionality applied to generalizing a memristor-based accelerator, using a Dot Product Engine (DPE) as an example that represents an analog form of computer with digital components embedded into the architecture. We explore generalizing this accelerator by making it more reconfigurable and partitionable, as well as more secure. We also identify some of the unique characteristics of this kind of system, such as different precision settings of DPE and trustworthiness of the deployed configuration into DPE.

The rest of the paper is organized in the following manner. Section II provides background on OS support for generalizing accelerators and on memristor-based DPEs. Section III lists a few applications that could benefit from DPE, motivating the need to generalize it. Section IV describes the DPE architecture and implications on OS. Section V describes how we envision DPE programming and code generation will be done. Section VI explores the OS support needed for generalizing DPE. Sections VII and VIII delve into virtualization and autotuning respectively. Finally, Section IX provides a summary and outlines future work.

II. Background

OS Support for Generalizing Accelerators. Recent attempts at building OS support for in-situ reconfigurable FPGA systems and GPUs provide some clues to practical OS support that would allow for a high level of computing hardware diversity. For example, for a soft-reconfigurable FPGA system, a set of fundamental services that should be provided by an OS has been prescribed. These OS services include: resource allocation, resource partitioning, application placement, and routing [37]. More recently, a set of new OS abstractions were proposed [32] to support GPUs and other accelerator devices to be used for general purpose computing in diverse operating environments. These new abstractions are based on a dataflow computing model. That is, the application programming interface (API) library consists of graph functions: create/open

graph, create/open a port, write/read to/from a channel/port, and run a graph. Laplante and Milojicic [28] studied the problem of OS support for diverse applications on heterogeneous architectures and prescribed a set of characteristics for an appropriate solution. In particular, they noted that in a new, fully abstract OS archetype, hardware and workload abstraction mechanisms are easy to extend.

Memristor-based Dot Product Engines. Many special-purpose accelerators have been proposed for neural network acceleration [10], [11], [33], for which dot-products are a fundamental operation. Memristor-based DPEs have been shown to be particularly promising [33] for computation patterns (such as inference) where one of the operands is constant and frequently reused. The DPE accelerates this operation by programming the constant operand into a memristor crossbar and performing the computation in-situ with a streaming input. Details of memristor-based DPE architectures and implications on OS functionality are discussed in Section IV, but we first motivate that the computation pattern of concern is prevalent.

III. Applications and Use Cases

Dot-product operations with frequently reused constant operands are common in a wide range of applications in machine learning, imaging, and scientific computing. We discuss some of these applications and their use cases in this section to further motivate the need for generalizing DPE.

Machine Learning. The benefit of using DPEs to accelerate machine learning workloads such as convolutional neural network (CNN) inference has already been demonstrated [33]. In this setup, each memristor crossbar represents a set of neurons in the same layer with the neuron weights being stored in the memristors as the constant operand of the dot product. Results have shown that using DPEs to accelerate CNN inference can lead to 14.8×, 5.5×, and 7.5× better throughput, energy, and computational density respectively compared to DaDianNao [10], a state-of-the-art digital architecture.

Imaging. Another class that would benefit from DPEs comprises imaging applications such as nonlinear diffraction-tomography based on multi-level fast multipole formulations [12]. These algorithms reduce the complexity of method of moments computations by approximating interactions with distant clusters via one representative point for each cluster. The resulting pattern is multiple levels of aggregation followed by multiple levels of disaggregation, each essentially consisting of matrix-vector multiplications by constant interaction matrices. Since the same matrices are reused for different clusters within a single aggregation or disaggregation level, this computation is well-suited for DPE whereby each memristor crossbar processes a different level.

Scientific Computing. Computational fluid dynamics (CFD) is widely used in simulations of weather and plasma combustion. The fundamental stencil operation consists of multiplying values from neighboring grid elements by constant weights and accumulating them to compute a value of a grid element for the next time-step. The same weights are used across grid-points and across time-steps. Such operations are good matches for the DPE, whereby each memristor crossbar processes a set of grid-points and the halo elements between crossbars are exchanged at each time-step.

Some scientific applications may not be well-suited for the DPE because the weight of the dot-product changes throughout the computation. These include N-body problems modeling interactions between atoms, planets, etc. Here, weights depend on distances between simulated bodies which vary spatially and temporally. Such patterns require support for fast rewrite operations, and may be better suited to future accelerators.

Some challenges presented by scientific computations include the desire for higher precision and demand for sparse computations which may necessitate support for light programmable control flow on the DPE. These requirements have implications on OS support, which is discussed in Section VI.

Use Cases. Using the DPE as a machine learning accelerator has numerous use cases with different OS implications. For example, in smart vehicle applications, DPEs can be deployed into microprocessors and microcontrollers embedded in the vehicles, and at points in the smart highway system. This system is real-time and safety-critical, requiring extremely low error rates which can benefit from OS support. As another example, in airport safety assurance, video recognition can be dynamically adapted when a potential threat is identified and requires higher precision or higher resolution.

Using DPE as an imaging accelerator has even more use cases. It can be deployed in hospital imaging equipment running in embedded microprocessors and microcontrollers. Depending on the procedure required, the OS is able to assign computational load, share data (securely and privately) between designated systems, and route (on the fly) between applications of differing precision. Alternatively, it can be deployed as a high performance co-processor running in remote supercomputers as a service and used by hospitals for multiple applications. The OS must manage the sharing by different clients guaranteeing fair access while preserving security and privacy requirements. Similar deployment strategies apply to scientific applications (Section III).

IV. Dot Product Engine

The Dot Product Engine (DPE) [18] is a hardware accelerator for matrix-vector multiplication using crossbar arrays to perform highly parallel multiplications and additions through analog operations. The concept is not new [36], [26], with actual implementations utilizing various technologies such as Flash, phase change memories (PCM), and oxide-based memristors. Circuit demonstrations have been shown in stand alone platforms [29], as well as part of larger programmable analog computing systems [15].

A full architecture was recently proposed [33] that integrates analog DPE components in a scalable, pipelined flow with digital routing, buffers, and other components to flexibly support inference computations for a broad range of Convolutional Neural Networks (CNN). Figure 1 shows the basic layout of this architecture, called ISAAC (In-Situ Analog Arithmetic in

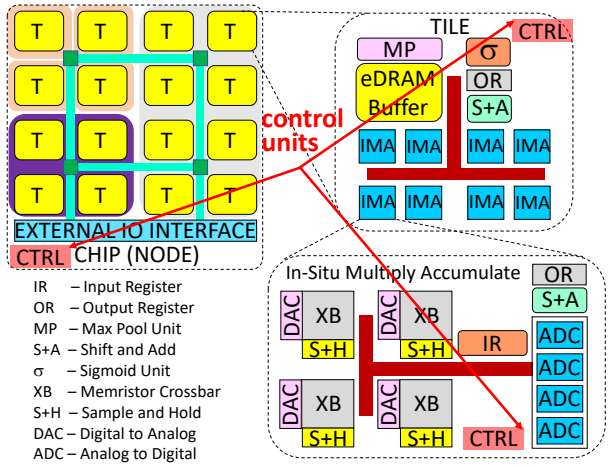

Fig. 1: DPE-based accelerator for convolutional neural networks [33]. The DPE can be partitioned by clustering tiles, and control units can be deployed at different levels.

Crossbars). The performance of this architecture was shown to exceed GPUs and digital ASICs by many orders of magnitude.

At a high level (Figure 1), an ISAAC chip is composed of a number of tiles (labeled T), each of which is further composed of eDRAM buffers to store input values, a number of in-situ multiply-accumulate (IMA) units, and output registers to aggregate results. Everything is connected with a shared bus. Each tile also includes shift-and-add, sigmoid, and max-pool units. Each IMA contains several crossbar arrays as well as shared ADCs.

Hybrid analog/digital accelerators, such as ISAAC, offer a potential route for continued growth in focused but important applications while the performance of general purpose CMOS hardware is flattening. Using the DPE accelerator as a core example, we highlight some of these challenges here, and Section VI describes the OS support layers that can flexibly address them more broadly.

A key performance aspect of the ISAAC architecture is the use of in-memory processing, whereby memristor arrays not only store neural network weights, but are also the location for computation. This reduction in data-fetching is a requirement to realize performance improvements, but it also imposes a strict data flow that also needs to be pipelined for performance. In ISAAC, a finite state machine (expressed as control units in Figure 1.) is envisioned to control this data flow needs to be configured by the compiler and OS. Additionally, portions of the chip can be physically assigned to different CNNs under computation, possibly at differing priority levels.

In addition to the tile partitioning for a specific task, another performance-determining parameter is the bit-precision needed in the matrix-vector multiplication operations. Neural networks are highly tolerant of low precision [13], [31], [39], and this reduces the burden on the ADCs, which dominate power consumption and require more time to capture analog signals with higher resolution. Hence, the precision needed to implement each neural network layer becomes an important

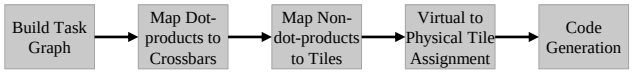

Fig. 2: Programming Framework Compilation Flow.

parameter to trade-off performance (high throughput, low power) versus classification accuracy. Choosing the optimal operating point requires careful management.

Mapping neural network layers of various sizes to the appropriate physical memristor crossbar arrays is another complex management task. As described above, computations that span multiple physical arrays require additional peripheral computations to reassemble the aggregate results. However, simply placing the neural network layer into the largest possible array can lead to larger errors (analog signals deteriorate with size and increased non-linear effects) and slower operating speeds.

The next sections outline how some of the unique trade-offs that are present in non-traditional, non-digital accelerators and must be managed by OS support layers.

V. Programming Framework

A programming framework for DPEs should enable programmers to write computation kernels with dot-products as the basic building block. We envision a domain-specific language (DSL) and compiler that provide a dot-product primitive, as well as a library of frequently used dot-product-based operations (such as matrix-vector and matrix-matrix multiplications). The DSL must also include types to specify data arrays that are constant throughout the computation in order to enforce that one dot-product operand is always constant as well as distinguish data that needs to be loaded to the memristor cells from data that needs to be streamed through the dot-product engine.

The programming framework we envision is outlined in Figure 2. The code written by the programmer is symbolically executed to build a task graph centered around dot-product operations. This approach has become increasingly popular in recent years [1], especially for machine learning workloads. The graph is then compiled to the DPE architecture as follows.

All the nodes in the graph performing dot-product operations with the same constant vector or matrix are identified and mapped to the same crossbar. If a crossbar receives too many dot-product operations, one potential optimization is to duplicate that crossbar and spread the operations to the duplicates to avoid that crossbar becoming a bottleneck in the system. Duplication has to keep in mind that there are a limited number of physical crossbars in total on the DPE and must therefore stay within the resource budget.

Once dot-product operations have been mapped to crossbars, the non-dot-product operations need to be mapped to the same tiles containing those crossbars. The objective of this mapping is to maximize data affinity, to avoid communication, and to minimize redundant computation. The basic algorithm for mapping non-dot-product operations is as follows. If all the sources of an operation are mapped to a tile, then that

operation is mapped to the same tile. Likewise, if all the destinations of an operation are mapped to a tile, that operation is mapped to the same tile. In the situation where an operation has all its sources in one tile and all its destinations in another tile, it can be mapped to either tile. Here, various heuristics must be applied to choose where to map that operation or a tuning framework (see Section VIII) can be used to find the best option in the design space. In the situation where an operation has its sources in multiple tiles and its destinations in multiple tiles, mapping the operation to a single tile will incur too much communication overhead. Instead, the operation can be replicated across destination tiles where it is redundantly computed but the communication is reduced. In general, control flow can introduce uncertainty in source and destination relations. Prediction techniques that convert control flow into data flow can be used to address such uncertainty.

Once all operations have been mapped to tiles, a sub-graph for each tile has essentially been constructed. So far in the process, *virtual* tiles are used which are not yet been bound to physical tiles on the DPE. The edges across sub-graphs can be used to derive the amount of communication between virtual tiles. This information can then be used to assign virtual tiles to physical tiles in such a way that minimizes the communication distance. Again, multiple mappings may be possible so a tuning framework could be useful to evaluate and pick the best mapping.

With operations grouped into sub-graphs and sub-graphs bound to physical tiles, the code can now be generated to configure and run the DPE. For the device, the sub-graph of each physical tile is traversed to generate the control logic for each tile. One challenge in doing so is recovering loop information to minimize code bloat due to loop unrolling. For the host, code is generated to configure physical tiles with the corresponding crossbar weights, to load the control logic for each physical tile onto that tile, and to push parameters onto the DPE to launch a kernel.

VI. Operating System Support

The DPE is "programmed" for a specific application by configuring the crossbar switch, loading weights into the memristor cells, setting arithmetic signs, and managing the pipeline. For CNNs, APIs can be provided to allow a software program to affect such a configuration to one or more DPEs (see Figure 4, left side).

An immediate problem in the above scenario is that the application is tightly coupled with the specific hardware, in this case, the DPEs. We wish for the OS to support any number of applications such as those described in Section III. A richer set of APIs needs to be provided that communicates with the OS providing not only diverse functional capability, but also traditional OS guarantees for security, multi-tasking, and tuning. Through the OS APIs, accelerator-specific drivers provide for the full set of DPE hardware configurations (Figure 4, right).

Reconfiguration and Run-time Partitioning The DPE as an accelerator can be used in various configurations either as a slave to a master (general purpose processor) or as a standalone accelerator unit. In both arrangements, the DPE architecture requires some form of control and data flow protocols for proper functionality of the system. Therefore, at the data level, the operations of moving data in and out of the DPE array would use traditional block-level transfers with memory mapped I/O for master/slave configuration or offload engines (e.g., TCP, UDP OE) to directly interact with data from external devices through network protocols. In both reported I/O cases, the reconfiguration phase through the CPU or DPE array controller will setup memory buffer locations for data to go in and out of the DPE array.

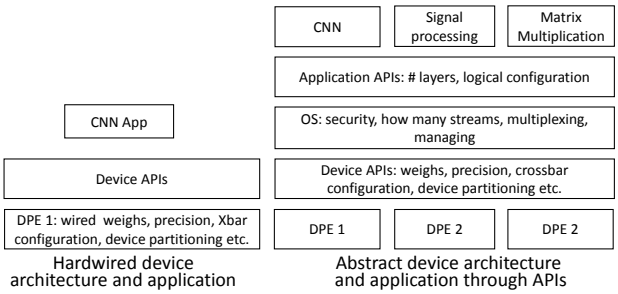

Fig. 3: Abstract OS Evolution.

For the execution/control phase, the OS leverages input from the DPE compiler that is aware of different memristor array granularity on the chip (e.g., array size and precision capability) to load the input program to a data flow graph of macro tasks (CNNs) to sub tasks that are executed at the specific tiles. A macro task and sub task represent the set of operations performed at Tile and IMA topology level such as setup of network for data movement across multiple memristor array units, notifying task completion to subsequent data receiving blocks that new data is available for the sink memristor array to compute, directing the data from IMA's to required special purpose cores (e.g., MP, OR, etc., see Figure 1), and setting up the precision of an array. Precision is affected by the ADC and DAC configurations at the sub task level through the IMA controllers. The DPE array configuration of data flow graphs, macro, and sub tasks is achieved through programming the controllers at each topological stage respectively shown in Figure 1.

Some of the trade-offs in reconfiguring the accelerator entail accuracy, power, and performance (throughput and latency). The relationships can be complex and open to empirical and theoretical evaluation. The OS helps in making run-time changes to optimize between the various factors. For example if power is limited, performance or precision can be traded off. Similarly, performance can be traded off for precision.

The DPE can be used for a single stream of data or in a more comprehensive way such as **partitioning** for multiple flows within the same or different hardware chips (See Figure 4). Partitions differ in various ways, including:

- **Precision.** At an airport, in real-time terrorist identification, video recognition can increase precision. Similarly, a critical medical procedure may require higher precision

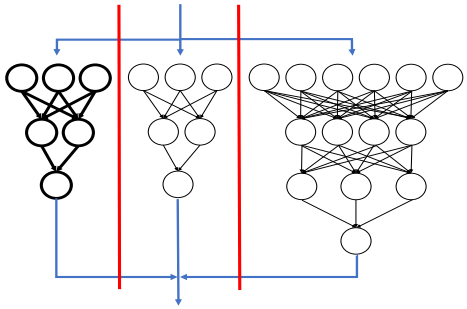

Fig. 4: Partitioning and Run-time Scheduling. Left graph has higher weights and precision than middle. Right graph is deeper. DPE OS routes requests by means of controllers.

compared to a routine one, but intelligent vehicles may trade off precision in favor of response time.

- **Security.** Videos with credentialed officers may be routed to a higher security partition. This would require a pre-stage recognition of tags that officers may carry which is omitted from picture.
- **Function.** Multiple functions can be programmed into the accelerator and dynamically selected or connected at run-time. For example, if a dangerous situation is identified in an autonomous vehicle, different paths could be immediately taken within real-time constraints.

Run-time partitioning enables higher degrees of freedom of run-time composing flows with different characteristics. In many ways, this resembles Software Defined Networks and/or Network Function Virtualization and indeed many principles could be applied to memristor-based accelerators.

Security Architecture. As systems become more complex, connected, and interdependent, accelerators are getting intense scrutiny because they may contain unknown exploits and bugs. However, to effectively implement security across a system, control points must exist where a supervisory entity can interpose and validate any security-critical control operations and data flows.

Traditional general purpose systems have the luxury of having the OS kernel implement security-critical control points. Accelerator-centric systems, such as the DPE, typically do not have such a luxury. Moreover, in DPE systems, interposition comes at a latency cost; and unlike conventional CPUs with hardware support for multiple privilege levels, the architecture and implementation of accelerators do not naturally support the provision of security control points directly as part of the computational functions/units.

A key function of a general purpose (multi-tasking) OS running on a conventional CPU is to provide fine-grain temporal and spatial resource-use separation between competing users. In the accelerator model, the requirements around temporal and spatial separation are much more coarse, and most of the time the OS can be uninvolved. This latter point gives us a degree of flexibility in terms of how the required security control points can be achieved for an accelerator-based system compared to a general purpose CPU system. As previously noted, it seems beneficial to model the system, from a security perspective, as a network of interconnected elements and use control and validation of the setup of routing between elements (and flows within elements) as the underlying foundation for the provision of the required security control points rather than dynamic security controls within the elements themselves. This model is somewhat analogous to the problem of securing SDN-based network systems [22].

To support the routing-controlled based security model just described, we require two broad capabilities. Alongside the foundational security requirement of being able to control (and validate) the setup and removal of routing and flows between and within elements, we also need to be able to reliably identify individual computation elements so that we can be confident that we are routing data through and to the intended elements. For individual accelerator functions we need the following basic set of security primitives:

- **Assurance** over the function implemented (e.g., robust cryptographic identity built-in).
- **Trustowrthiness** of the function to be deployed.
- **Protection (integrity)** of the function deployed.

For the advanced use cases described earlier, additional security primitives are required:

- Support for **safe data partitioning** within the engine. Access only within partition should be enabled.
- Support for **safe concurrent use**. No cross-effects between partitions should be possible.

Precise requirements will vary based on the particular use cases being addressed. Our general security goal is establishing confidence in the results of the analytical outputs of the DPE system from a data provenance and lineage perspective.

VII. Virtualization

One of the barriers to adoption of accelerators in enterprise environments, especially cloud computing, is their lack of (or limited) support for virtualization [9]. Virtualization loosely refers to the ability to securely and transparently share underlying hardware resources among different users e.g. processes, virtual machines, CPUs, etc. with each user independently marshalling and running its own code and commands [34]. The aim of virtualization is increased hardware utilization and consequently lower power consumption. Almost every major component in a typical enterprise environment including I/O devices is designed to support virtualization. Accelerators that support virtualization will gain more traction in the enterprise environment.

In the context of the DPE, the aim of virtualization is to support multiple virtual DPE stream layers on an underlying physical DPE. This is achieved by effectively multiplexing or time-slicing different streams on the same hardware, as shown in Figure 5, while allowing a resource manager to assign priority to streams.

From a high level, virtualization is achieved by replicating the external interfaces of the DPE to provide multiple virtual interfaces. Virtual interfaces are essentially separate

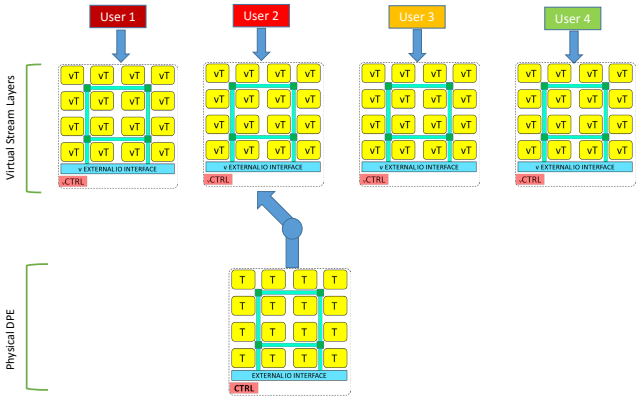

Fig. 5: DPE virtualization

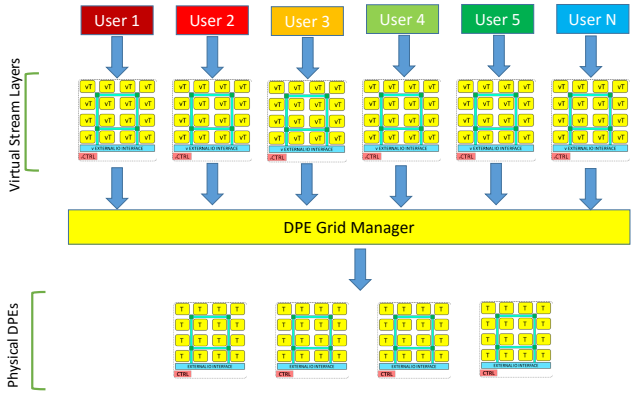

Fig. 6: RackScale DPE Approach

compartments for user and configuration data. Each virtual interface is associated with a stream layer (see Figure 5). For example, the virtual interfaces for the CTRL unit (see Figure 1) are the CTRL's configuration registers replicated for each of the stream layers supported by a DPE. This is similar to Simultaneous Multithreading (SMT) (or Hyperthreading) in processors where as little hardware as possible is replicated to support multiple hardware threads (e.g., only 5% extra chip area to support SMT in Intel Xeons [27]).

A hardware scheduler dynamically selects on each time-slice the layer to run via the appropriate virtual interfaces. The hardware scheduler supports a number of scheduling algorithms such as round robin and weighted round robin. The DPE provides a separate administration interface to enable a resource manager (e.g., an OS) to dynamically select the scheduling algorithm. A resource manager can use the administration interface to manage the type of actions a user is allowed to perform. For example, a stream layer can be prevented from reconfiguring the DPE. The hardware scheduler also provides mechanisms which enable a resource manager to dynamically remove layers from consideration for scheduling. This can be employed, for example, to temporarily dedicate the physical DPE to a particular stream layer. Time slicing can also be disabled completely. The hardware transparently loads and drains stream layers on the physical tiles ensuring strong isolation between stream layers.

The basic hardware virtualization mechanism described above is leveraged to build a virtual platform capable of managing multiple DPEs and operating at the rack-scale. Instead of managing physical DPE separately, we pool them together and create a resource pool under the active management of a DPE Grid Manager (see Figure 6). The Grid Manager is external software that manages multiple DPEs and provides OS-like services to DPE users. DPEs are dynamically requested, allocated and deallocated by a tenant via the Grid Manager. For public cloud computing environments, Grid Manager facilitates usage tracking and billing.

Our approach to virtualization has a number of advantages:

- Virtual layer approach retains the streaming design of the DPE. Further, it does not change the in-situ nature of processing as the weights are still on-chip.
- It allows a resource manager to prioritize the processing of particular layers without the need to drain queues.
- Virtualization is transparent to users sharing the DPE. Each user gets the same interfaces/features as the physical DPE, no program modifications are required.

Cryptographical Isolation. For sensitive workloads, the DPE can be extended to support the use of encrypted user and configuration data using user supplied encryption key on a layer by layer basis. This may be quite useful in a cloud computing environment where a user does not control physical access to the DPE and may not want to trust the administrator. Instead of each stream layer user supplying data to the DPE in plaintext, data are now encrypted with user supplied key using a symmetric encryption algorithm such as AES [30]. The DPE automatically and transparently decrypts the input data just before processing on a tile. It also re-encrypts the output data before it leaves the tile. So at any point in time data at rest inside a system is encrypted making it difficult for an administrator to snoop at sensitive user information. We assume that an adversary cannot non-destructively physically snoop on data inside the DPE.

A user can securely provide an encryption key to the DPE using asymmetric (public key) cryptography. A unique private key is securely embedded inside each DPE by the manufacturer and the corresponding public key is generated. The manufacturer also securely provisions a digital certificate chain, backed by a certification authority, that can be used to verify the authenticity of the keys and consequently that of the DPE. The user and the DPE now use an appropriate key-exchange protocol such as SIGMA [23] to securely derive a symmetric key for encrypting data and communication between them.

VIII. Hardware and Software Autotuning

The configuration of the hardware and software described in this paper defines a large and complex design space and presents an optimization challenge. We propose to apply autotuning techniques and search heuristics to aid DPE hardware design and software configuration and optimization.

General purpose autotuning frameworks, such as OpenTuner [3], combine different search heuristics to optimize a user-specified metric. This approach yields good results in different hardware and software configuration domains. PetaBricks [2] and Tangram [7] are languages, compilers, and autotuners that introduce abstractions that enable programmers to define multiple algorithms for the same problem. The ParamILS framework [20] applies stochastic local search methods for algorithm configuration and parameter tuning. Bosboom *et al.* and Eliahu used OpenTuner to implement a domain specific language for data-flow programming [4] and a framework for recursive parallel algorithm optimization [14]. Xu et al. [38] use distributed OpenTuner instances to optimize the FPGA compilation flow, and Huang et al. [19] study the effect of compiler optimizations in High-Level Synthesis for FPGAs, showing the complexity of the search space and the difficulty of its exhaustive exploration.

Changes in hardware and software parameters used to configure the DPE generate trade-offs and complex parameter relationships. These parameters can be explored empirically and theoretically, as discussed in Sections V and VI.

Table I shows hardware and software parameters that define a design and reconfiguration space for the DPE. The impact of these parameters on several metrics can be measured using analytical models and simulations. Table II shows metrics and measurement strategies that can be optimized for the DPE.

Tunable Parameters	
Hardware	*Software*
Routing tables, eDRAM buffer size; bit size in IMA, layer; bus width; crossbar & IMA number in tiles	Memristor arrays, layers & routing tables mapping
ADC hardware configuration and heterogeneous tile designs	ADC and tile usage

TABLE I: Tunable software and hardware DPE parameters

Tuning Metrics and Measurements	
Metrics	*Measurements*
Bandwidth, latency, execution time, router area, throughput	Modelled area and power overhead for components
Computation, power, crossbar efficiency	Cycle accurate simulation for: tile communication; eDRAM access
Usability, flexibility	Empirical, qualitative

TABLE II: Tuning metrics & measurement strategies for DPE

For the software configuration, Figure 7 shows the results [5] we obtained using autotuning to optimize the selection of CUDA compiler flags for Rodinia [8] and matrix multiplication applications. The applications shown in Figure 7 are matrix multiplication with uncoalesced access to global (MMU) and shared memory (MSU), coalesced access to global (MMG) and shared memory (MMS), gaussian elimination (GAU), heart wall medical imaging benchmark (HWL) and the Needleman-Wunsh dynamic programming algorithm (NDL).

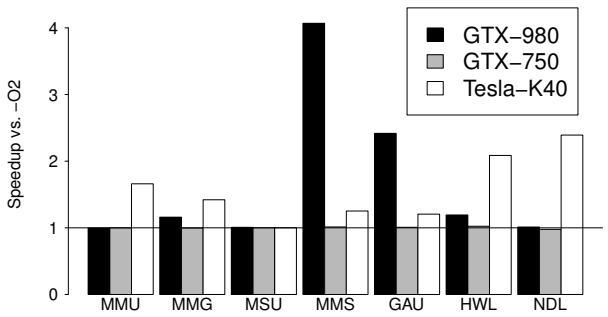

Fig. 7: Speedup vs. `-O2` after autotuning NVCC compiler parameters for heterogeneous applications [5].

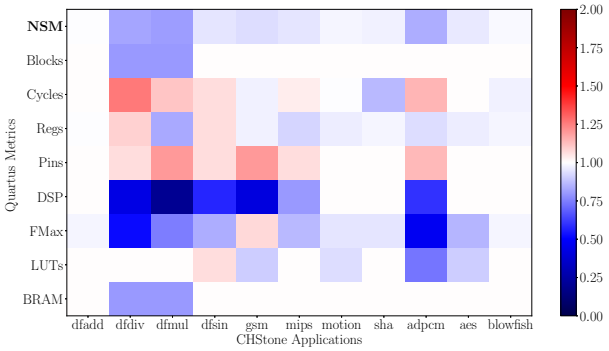

Fig. 8: Relative decrease in Quartus-reported FPGA metrics for CHStone [17] High-Level Synthesis (HLS) benchmark.

For the hardware configuration, Figure 8 shows the results we obtained using autotuning to optimize the configuration of a High-Level Synthesis (HLS) tool [6] subject to a series of FPGA hardware parameters. The results in Figure 8 were obtained in the Stratix V DE5-Net FPGA, after tuning HLS parameters. The intensity of blue squares indicate how much was improved for each metric and application pair. The autotuning objective was the Normalized Sum of Metrics (NSM), shown in the first row of the same figure.

Domain-agnostic autotuning frameworks can aid the exploration and optimization of the DPE hardware and software configurations and design space. We intend to use this autotuning approach to optimize the DPE parameters listed in Table I, subject to the metrics listed in Table II.

IX. Summary and Future Work

We have presented a case for a more general approach to accelerators by introducing OS support. We have described how reconfiguring, partitioning, securing, and programming memristors-based accelerators can support applications and use cases described in Section III. Some aspects, such as precision and trustworthiness of configurations deployed are atypical for general purpose systems and require new solutions. This resulted in broadening their use without compromising all the benefits of special purpose design, such as performance and power. In the future, we plan to re-implement DPEs with more knobs for run-time reconfiguration. We will also expand

the system software layers to support DPEs and make them more reconfigurable. In addition, we are identifying other potential applications that can broaden their deployment.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Jason Ansel, Cy Chan, Yee Lok Wong, Marek Olszewski, Qin Zhao, Alan Edelman, and Saman Amarasinghe. Petabricks: A language and compiler for algorithmic choice. *SIGPLAN Not.*, 44(6):38–49, June 2009.

[3] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. Opentuner: An extensible framework for program autotuning. In *Proceedings of the 23rd international conference on Parallel architectures and compilation*, pages 303–316. ACM, 2014.

[4] Jeffrey Bosboom, Sumanaruban Rajadurai, Weng-Fai Wong, and Saman Amarasinghe. StreamJIT: a commensal compiler for high-performance stream programming. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, pages 177–195, Portland, OR, USA, 2014. ACM.

[5] Pedro Bruel, Marcos Amars, and Alfredo Goldman. Autotuning cuda compiler parameters for heterogeneous applications using the opentuner framework. *Concurrency and Computation: Practice and Experience*, pages e3973–n/a, 2017.

[6] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Tomasz Czajkowski, Stephen D Brown, and Jason H Anderson. Legup: An open-source high-level synthesis tool for fpga-based processor/accelerator systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(2):24, 2013.

[7] Li-Wen Chang, Izzat El Hajj, Christopher Rodrigues, Juan Gómez-Luna, and Wen-mei Hwu. Efficient kernel synthesis for performance portable programming. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016.

[8] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *IEEE International Symposium on Workload Characterization, 2009.*, pages 44–54. IEEE, 2009.

[9] Fei Chen, Yi Shan, Yu Zhang, Yu Wang, Hubertus Franke, Xiaotao Chang, and Kun Wang. Enabling fpgas in the cloud. In *Proceedings of the 11th ACM Conference on Computing Frontiers*, page 3. ACM, 2014.

[10] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE Computer Society, 2014.

[11] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 27–39. IEEE Press, 2016.

[12] Ronald Coifman, Vladimir Rokhlin, and Stephen Wandzura. The fast multipole method for the wave equation: A pedestrian prescription. *IEEE Antennas and Propagation Magazine*, 35(3):7–12, 1993.

[13] Matthieu Courbariaux, Yoshua Bengio, and J David. Low precision arithmetic for deep learning. *CoRR, abs/1412.7024*, 2014.

[14] David Eliahu, Omer Spillinger, Armando Fox, and James Demmel. Frpa: A framework for recursive parallel algorithms. Master's thesis, EECS Department, University of California, Berkeley, May 2015.

[15] S. George, S. Kim, S. Shah, J. Hasler, M. Collins, F. Adil, R. Wunderlich, S. Nease, and S. Ramakrishnan. A programmable and configurable mixed-mode fpaa soc. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(6):2253–2261, June 2016.

[16] Herman H Goldstine and Adele Goldstine. The electronic numerical integrator and computer (eniac). *Mathematical Tables and Other Aids to Computation*, 2(15):97–110, 1946.

[17] Yuko Hara, Hiroyuki Tomiyama, Shinya Honda, Hiroaki Takada, and Katsuya Ishii. Chstone: A benchmark program suite for practical c-based high-level synthesis. In *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, pages 1192–1195. IEEE, 2008.

[18] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams. Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2016.

[19] Qijing Huang, Ruolong Lian, Andrew Canis, Jongsok Choi, Ryan Xi, Nazanin Calagar, Stephen Brown, and Jason Anderson. The effect of compiler optimizations on high-level synthesis-generated hardware. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 8(3):14, 2015.

[20] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306, 2009.

[21] R. A. Iannucci. Toward a dataflow/von neumann hybrid architecture. In *[1988] The 15th Annual International Symposium on Computer Architecture. Conference Proceedings*, pages 131–140, May 1988.

[22] Ludovic Jacquin, Adrian L Shaw, and Chris Dalton. Towards trusted software-defined networks using a hardware-based integrity measurement architecture. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–6. IEEE, 2015.

[23] Hugo Krawczyk. Sigma: The sign-and-macapproach to authenticated diffie-hellman and its use in the ike protocols. In *Annual International Cryptology Conference*, pages 400–425. Springer, 2003.

[24] Sun-Yuan Kung. On supercomputing with systolic/wavefront array processors. *Proceedings of the IEEE*, 72(7):867–884, 1984.

[25] P. Laplante and D. Milojicic. Rethinking operating systems for rebooted computing. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8, Oct 2016.

[26] Konstantin K. Likharev. Crossnets: Neuromorphic hybrid cmos/nanoelectronic networks. *Science of Advanced Materials*, 3(3):322–331, 2011.

[27] Deborah Marr, Frank Binns, David Hill, Glenn Hinton, David Koufaty, Allan Miller, and Michael Upton. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, 6(1), 2002.

[28] D. Milojicic and T. Roscoe. Outlook on operating systems. *Computer*, 49(1):43–51, Jan 2016.

[29] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature*, 521(7550):61–64, 2015.

[30] NIST FIPS Pub. 197: Advanced encryption standard (aes). *Federal Information Processing Standards Publication*, 197(441):0311, 2001.

[31] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[32] Christopher J. Rossbach, Jon Currey, Mark Silberstein, Baishakhi Ray, and Emmett Witchel. Ptask: Operating system abstractions to manage gpus as compute devices. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 233–248, New York, NY, USA, 2011. ACM.

[33] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 14–26, June 2016.

[34] Amit Singh. An introduction to virtualization. *kernelthread. com, January*, 2004.

[35] Hayden Kwok-Hay So and Robert W Brodersen. Improving usability of fpga-based reconfigurable computers through operating system support. In *Field Programmable Logic and Applications, 2006. FPL'06. International Conference on*, pages 1–6. IEEE, 2006.

[36] K. Steinbuch. Die lernmatrix. *Kybernetik*, 1(1):36–45, 1961.

[37] Grant Wigley and David Kearney. The development of an operating system for reconfigurable computing. In *Field-Programmable Custom Computing Machines, 2001. FCCM'01. The 9th Annual IEEE Symposium on*, pages 249–250. IEEE, 2001.

[38] Chang Xu, Gai Liu, Ritchie Zhao, Stephen Yang, Guojie Luo, and Zhiru Zhang. A parallel bandit-based approach for autotuning fpga compilation. In *Proc. of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 157–166. ACM, 2017.

[39] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint:1606.06160*, 2016.