Computing In-Memory, Revisited

Dejan Milojicic
Systems Lab
Hewlett Packard Labs
Palo Alto, CA, USA
dejan.milojicic@hpe.com

Kirk Bresniker
Office of CTO
Hewlett Packard Labs
Palo Alto, CA, USA
kirk.bresniker@hpe.com

Gary Campbell
Security Lab
Hewlett Packard Labs
Palo Alto, CA, USA
gary.campbell@hpe.com

Paolo Faraboschi
Systems Lab
Hewlett Packard Labs
Palo Alto, CA, USA
paolo.faraboschi@hpe.com

John Paul Strachan
Systems Lab
Hewlett Packard Labs
Palo Alto, CA, USA
john-paul.strachan@hpe.com

Stan Williams
Systems Lab
Hewlett Packard Labs
Palo Alto, CA, USA
stan.williams@hpe.com

***Abstract*—The Von Neumann's architecture has been the dominant computing paradigm ever since its inception in the mid-forties. It revolves around the concept of a "stored program" in memory, and a central processing unit that executes the program. As an alternative, Processing-In-Memory (PIM) ideas have been around for at least two decades, however with very limited adoption. Today, three trends are creating a compelling motivation to take a second look. Novel devices such as memristor blur the boundary between memory and compute, effectively providing both in the same element. Power efficiency has become very important, both in the datacenter and at the edge. Machine learning applications driven by a data-flow model have become ubiquitous. In this paper, we sketch our Computing-In-Memory (CIM) vision, and its substantial performance and power improvement potential. Compared to PIM models, CIM more clearly separates computing from memory. We then discuss the programming model, which we consider the biggest challenge. We close by describing how CIM impacts non-functional characteristics, such as reliability, scale, and configurability.**

Keywords*—*Architecture, computing, memory, interconnects, accelerators, programming, configuring, performance, scaling.

I. Introduction

The Von Neumann model has dominated computing systems ever since its introduction [1] in 1945. It has proven to be exceptionally useful for almost seven decades [2][3][4]. Its strength is based on simplicity: data and instructions are stored and accessed from memory by loading them into the central processing unit (CPU), which executes control and arithmetic/logic operations (see Fig 1). Over time, memory access latency started to become a problem as CPUs became faster than memory. As a result cache hierarchies appeared to bring major benefits (improved memory access latency), but also problems (cache coherence complexity and security flaws). Alternative ideas, such as Processing in Memory (PIM), have been proposed in the last two decades, with relatively little success outside of limited domains like databases and storage systems [5][6][7][8]. Most of that work was focused on stationary data with some processing collocated.

One consequence of the challenges faced by a Von Neumann architecture is the steady reduction of computing systems' ability to effectively operate on large data. This is visible in the ratio of the memory bandwidth (bytes/s) over computing speed (flops/s). Fig. 2 shows the steady drop over time from a byte/flop ratio of 1.0 (where all data in memory is readily available at processor speeds) to several orders of magnitude lower.

The combination of increased data volumes and data mining applications with limited compute intensity and locality are making this imbalance even more challenging today. There is a strong interest to find ways to reverse the historical trend and significantly increase the bytes/flops ratio. The introduction of novel memory devices that can combine storage and computing in the same cell provides an opening for such a reversal. With these devices, it makes much more sense to bring the computation to memory. This is also the basis of what at HPE we call "memory driven computing" [9]. In our CIM vision, we focus primarily on dataflow-like architectures, where data is continuously input into device which is able to both store some data and computation.

The rest of the paper is organized as follows. In Section II we provide background and motivation, and we try answering the question why CIM will be successful, when so many previous similar efforts have not resulted in broad adoption. Section three III presents the CIM model, including logic and core architecture, analogy to object oriented systems,

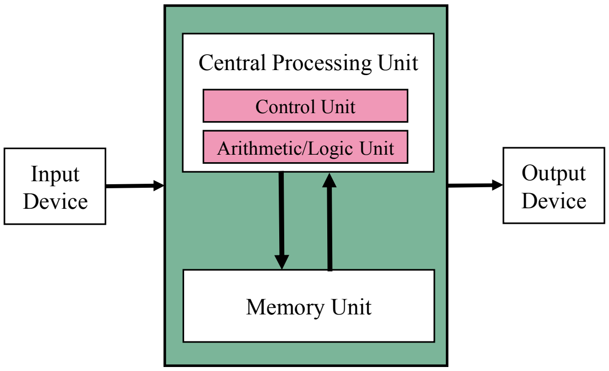

Fig 1. Von Neumann Architecture

2575-8411/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDCS.2018.00130

programming languages and run-times/operating systems. In Section IV, we discuss security, virtualization and resource management. In Section V we explore non-functional characteristics, such as fault tolerance, scaling, configurability, and supportability. In Section VI, we present some results of our own work. Finally, VII summarizes paper and discusses next steps.

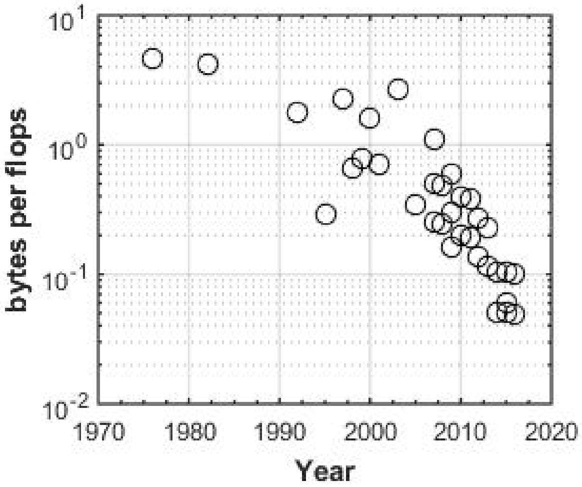

Fig 2. Memory bandwidth per processor floating point operations (FLOP)

II. Background and Motivation

In this section we provide a background and motivation in terms of a brief history of hardware, use cases, and applications. We then close section by discussing societal implications and why we believe that CIM can be successful.

A. *Hardware*

The status quo relationship between compute and memory, referring back to the EDVAC paper [1], is shifting due to the emerging memory technologies. The persistence of memory is shifting the temporal and energy scalability of techniques that trade space and compute, such as memoization. The realization that the current economic forces, the end of Dennard's Law [10][11], and the imminent challenges of Moore's [12][13] and Rocks's Laws [14], have led us to consolidation and monoculture, which in turn has left us exposed. Several generations of performance improvements may have to be yielded back due to fundamental insecurity, as exemplified by some of the recent vulnerability discoveries around Meltdown and Spectre [15][16].

The problems we wish to tackle, which are not dominated by algorithmic velocity but by data throughput, are also shifting rapidly. As data intensity near the processing elements increases, photonics interconnects grow in importance, since they enable communications from centimeters to kilometers at the same energy per bit, varying only in the time of flight. Finally, there is the realization that the Turing symbolic model, as instantiated by von Neumann, forces us perhaps unnaturally to the digital domain. In many cases, we constrain ourselves to very inefficiently solve approximate problems with high digital accuracy. This comes at expense of linear and non-linear analog systems that, while complex and even borderline chaotic, may be far more efficient than conventional digital approaches, and ultimately a better fit for the underlying problem.

Add all of these together and we end up wanting a novel solution for the optimization of in-memory models of complex, real time physical and economic systems, where the scale of data necessitates approaches where data movement must be the fundamental cost. In this new world, compute is free (the last few Moore's Law steps will see to that), but data is priceless, abundant, and we can finally deal with that abundancy.

There are a number of approaches to these new types of computing, and the new IEEE "*Rebooting Computing*" initiative was started as a focal point for researchers in this field [17]. Specific attempts at CIM using technologies, such as ReRAM [18], STT MRAM [19], Memristor [20], DRAM [21][22][23], and SRAM [24], have recently been developed.

B. *Use Cases*

CIM is well suited to address a variety of fields, such as sensors, robotics, control, and scientific computing. In this paper we focus on edge computing and memory intensive applications. The following characteristics are common across them.

- **Data is close to computation**, there is no need to move it, which results in power and performance optimization. In the past, other approaches used offloading [25] and migration [26] towards the data, but not as effectively and breaking programming models.
- **Data is persistent.** The idea is that the application state can be constantly captured over time and upon reboot or restart (due to failure) it will be available to continue computation. This naturally leads to storing data in non-volatile devices (such as NVM), but also opens the door to other forms of distributed persistence based on data replication schemes.
- **Applications employ dataflow**. Data manipulation, understanding and mining matches well dataflow programming models. These also better suit the notion of data collocated with computing elements and opens up opportunities for embarrassingly parallel computing.

Edge computing. We typically consider edge computing close to the data generation sources, such as sensors, or other devices. We contrast "edge" with "cloud", where data is processed in a (logically) centralized location. Edge computing assumes that moving all the data to the cloud is too onerous, and enough computing power at the edge is necessary to consolidate the data prior to passing it on to a cloud-centralized phase. This is also the place where some analytics and learning can take place to filter out (triage) redundant data and extract meaningful information. Edge applications typically consists of streaming processes taking device data from sensors, such as cameras. For example, applying deep learning inference at the edge can convert raw data (e.g., an image or video) into a tagged meta-data representation (e.g., classified objects or recognized text), thus massively reducing the size to something that can be efficiently transferred to the cloud. Computing in memory is very relevant to edge computing: it lowers cost, improves performance, and lowers power consumption. These are very important characteristics in any computing device, but particularly in edge devices and even more so when devices are energy constraints or battery operated [27].

Memory-centric computing. When value and size of data grows higher than computation, data (and traditional storage) are treated as first level citizen, surrounded by computation as needed. This data is harder to move (because of size and security concerns), so it makes sense to bring computation closer to it. This field is an ideal match for computing in memory where computation is literally allocated in physical vicinity to the data. For example graph-heavy applications (typical in the intelligence community) need to track information over a long time, the graphs are hard to reproduce after reboots/failures due to their sheer size, or the lengthy history that would need to be repeated. Social networking applications are a variation of graph problem, with potentially larger scale but lower service level agreement (SLA) requirements. In both examples the benefits from CIM are clear and similar.

Finally, a common thread that ties these fields together is **Deep Learning**. As content complexity increases, making representation learning indispensable [28], a growing use of Artificial Intelligence (AI) and Machine Learning (ML) can leverage CIM because of the dataflow nature of tensor operations, and the underlying matrix operations that are involved. We discuss that in the remainder of the paper.

C. Applications

There are a number of applications that can benefit from the CIM model. Neural networks, used in pattern recognition, are a natural fit for the dataflow nature of CIM. The ability to create layers of networks and (re-)configure them to trained models fits with how CIM can be organized. Matrix multiplication-based scientific algorithms are at the foundation of neural networks, and also map well to the CIM model. Memory-side and storage-side accelerator functions are commonly optimized using low power accelerator devices that could be also implemented using CIM model.

Appendix at the end of the paper contains some high level comparison of the applications and their suitability to CIM model. It is important to emphasize that CIM is not meant to be solution to all applications and for all problems. Some applications will continue to be best served by Von Neumann model, while others will benefit more or less from CIM. Von Neumann can still be the de Facto approach. But the use cases and applications presented in this Section can warrant special purpose CIM hardware that only apply to them.

D. Societal Implications

The computing evolution has moved from general to special purpose ever more so. Purely based on the number of instances and computing power, most of traditional computing migrated towards mobile devices (phones) in less than a decade, relatively a very short time. Today, computing is increasingly moving towards the so called edge, where a sea of sensor devices are deployed to control every facet of human life. Some of the sensors are cameras that are associated with image/video/voice recognition, others similarly track various physical artefacts with possible ability to also actuate/control. Processing all of this data can critically impact human and whole nations' existence.

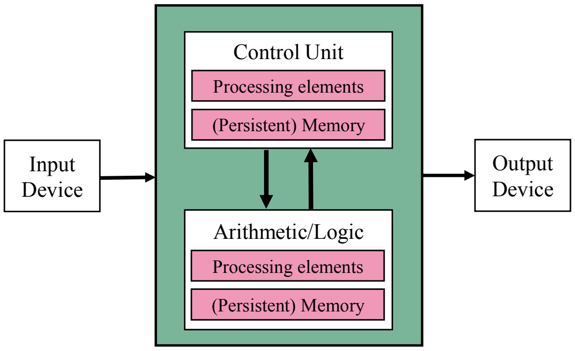

Fig 3. Evolving CIM Model

In addition, increasingly relying on artificial intelligence (initially using machine learning and deep learning) takes us to unchartered territory. The need arises for increased performance to process all the data at low power both in data centers and even more so at the edge to reduce data transfer to data centers and cloud. New ethical approaches to design are being introduced to standardize ways how to treat artificial intelligence and account for human being in the first place [29]. In addition, the approaches to cybersecurity are evaluated for their use of artificial intelligence and machine learning [30].

E. Can CIM Be Successful?

Computing in Memory was attempted many times in the past in various incarnations (PIM [5][6][7][8][31][32][33][34] and near memory processing [23][25][35][36][37]). And it has gained a lot of interest lately [38][39][40][41][42][43][44][45]. Why do we (and other researchers) believe that we stand chances of gaining adoption?

We believe that earlier attempts were ahead of their time and the current attempts are timely because of the "perfect storm" effect caused by the convergence of five trends:

New technologies, such as neuromorphic, bio-inspired, adiabatic, reversible, approximate, quantum, and combinations of these. This is the right time to revisit the Von Neumann model and attempt to overcome problems with caches, security, complexity, etc.

Application demand. Image, video and audio recognition, and large scale data analytics are based on increasing processing power but at less power consumption. These applications dominate processing compared to general purpose computing and are becoming center of attention of CPU vendors and IT companies.

Critical to mankind. Deep learning applications are increasingly being deployed in every facet of daily life (in phones, consumer devices, sensors, autonomous vehicles, manufacturing, industrial control, infrastructure, etc.) and mankind existence is increasingly dependent on automation, IT and cybersecurity, which in turn is enabled by more powerful computing.

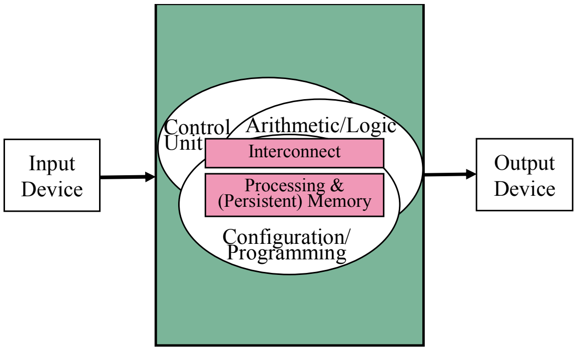

Fig 5. Evolving CIM Model, Cont.

Economy of scale. In the past, PIM lacked the economy of scale of IoT, while Von Neumann computers had the lion share of computing. With the increasing likelihood that hardware accelerators for AI/ML/DL will be broadly deployed, not just for gaming, crypto-currencies, and HPC applications, but also on sensors and mobile devices, the economy of scale is turning to their favor.

Limited alternatives for continued performance growth. In many ways, the von Neumann architecture maintained pre-eminence because the multi-decade exponential growth in CMOS masked many of its limitations and reduced the incentives to switch platforms. As noted earlier, such scaling in CMOS has not only slowed down, but is imminently approaching physical and economic limits. It is only fitting, therefore, that the end of Moore's Law and Dennard Scaling means the end of a von Neumann hegemony amid a desperate need to find performance growth using alternative approaches.

III. The Computing in-Memory Model

The CIM approach technologically and architecturally collocates processing and memory together, for compute (logical, arithmetic) and control functions (see Fig 3).

In addition to processing and memory functionality, interconnects also become an integral part of the CIM model, and programming/configuration becomes the core functionality above control and arithmetic/logic units (see Fig 4). Interconnects are critical as they enable reconfiguration of the paths for the dataflow model, and allow reconnecting individual units into application-specific workflows. Interconnect standards whose architecture includes accelerators (such as [46][47][48]) are the prime candidate for success in this domain.

Fig 5 shows a possible organization of a CIM device. A CIM micro-unit consists of control, data, and processing components (logic/arithmetic). Multiple CIM micro-units build a CIM unit when they are connected in a predefined configuration. They can be organized in tiles, and multiple tiles can be further scaled up (not shown in the figure). This figure allows for different degrees of heterogeneity, every CIM unit can be different. This in many ways resembles distributed systems and lessons learned from distributed systems could be readily applied, especially in terms of reliability and scaling.

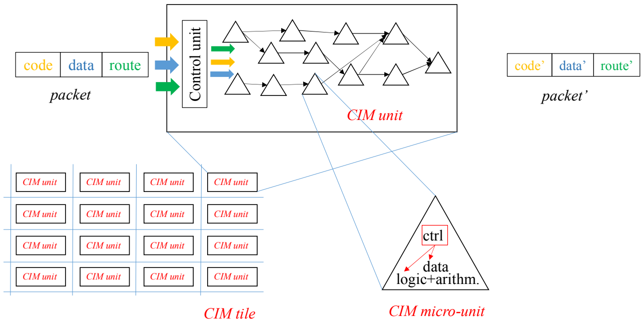

Fig 4. One possible implementation of the CIM model

A. Logic and Core Architecture

Different teams have approached the core operations differently. Chen et al. rely on AND, OR, and XOR operations upon which to build all other logic [18]. Borghetti et al. are using NOT and IMP (material imply) as two core logic operations [20]. Hardware architectures are based on these operations. Various approaches to hardware architecture, classified by Khoram et al. [35], rely on: matrix multiplication (dot products) combined with shared memory, such as in ISAAC [49] and memristive Boltzmann machine [50]; neuromorphic systems mimicking human brain, such as in FlexRAM [51] and work by Liu [52]; associative processors known as content addressable memory combined with nonvolatile memory, such as TCAM [53][54] and Associative Processors [55][56][57]; and coarse grained reconfigurable architectures [58], such as nonvolatile FPGA [59] and reconfigurable in-memory computing architecture [60].

B. Programming CIM

CIM programming adopts static, dynamic, and self-reprogrammable dataflow concepts. Each programming concept brings an additional degree of flexibility achieved by reconfiguring different aspects of the CIM architecture.

Static dataflow is the natural extension of existing dataflow computational models, such as the ISAAC architecture [49]. Through the instruction set, applications can program the CIM crossbars to implement a target neural network that would execute over and over again. With CIM, the inherent colocation of memory and computation enables additional flexibility in how computation is configured. This enables more opportunities for training, as well as feed-forward and closed loops. This is an evolution from FPGA-like configuration of code, to loading a binary into processor, such as CUDA code into GPU.

Dynamic dataflow assumes that the data coming in can be dynamically routed to those parts of the CIM at different granularity as a function of the state in the CIM and the input data. The routing could be expressed explicitly as a part of the incoming packet or it could be implicit as a function of the state in CIM, or both.

Finally, **self-programmable dataflow** enables carrying code as a part of the packets to dynamically program functions as packets arrive. This allows the highest level of flexibility in programming. Past research exists on this topic, but no production-level commercial equivalent exists as yet.

Section D, at a high level, describes the programming language and system software support to enable these configuration models.

C. *Analogy to Object Oriented and Modular Systems*

In many ways, the CIM model resembles some aspects of object oriented approaches by hiding the internal data and computation methods, and only exposing external abstract interfaces that specify the intent. Similarly, CIM holds both data and operations inside the hardware cells and allows data to flow in and out of it. Object oriented systems could be mapped on top of CIM and leverage the additional security protection (see Section IV.A) and reliability (see Section V.A) that CIM offers. However, this would require a lot of research, as of now dataflow languages and systems are obvious match.

D. *Programming Languages*

Just like neural networks have evolved and are being supported by a plethora of platforms and development environments, we also expect that new expressive programming models will evolve for CIM. They will require programming languages to map onto the control and processing instruction sets for CIM.

CIM programming languages will need to understand the micro-unit level: how data is received from outside of the micro-unit, how programs are loaded, how micro-units are configured, how memory is allocated, data decrypted, etc. Compilers will further need to understand the architecture across micro-units and across tiles: data locality and how data is streamed across micro-units and across tiles; how graphs are built and mapped to physical units; etc. Fortunately, there is substantial work on dataflow compilers and programming languages and in support of CIM [61][62][63][64][65][66][67][68] that can be leveraged.

E. *Run-times and Operating Systems*

Just like accelerators, initially CIM components will be used as slave devices, attached to traditional systems running standard operating system and using traditional runtimes. CIM binaries compiled from CIM programming languages may be downloaded into CIM devices, just like CUDA is being used for GPUs or equivalent tools built for FPGAs. Both of these different approaches are viable to CIM although they would result with different degrees of reconfigurability.

Over time, we expect that CIM units will evolve from the master-slave model to a cooperative relationship, where both traditional and CIM models can co-exist side-by-side. Once CIM has proven its effectiveness, we can expect integration in the same hardware module which therefore will require operating systems support. Finally, CIM computers can start running natively requiring full run time and operating system support (Fig 6).

One of the advantages of CIM is the built-in support of heterogeneous devices. The runtime and operating systems will need to hide this heterogeneity and expose a common interfaces to users [69]. For the cooperative, integrated and native models, we expect that entirely new operating systems will be developed to support CIM units natively [70][71].

F. *Interactions between Von Neumann and CIM models*

Von Neumann and CIM systems can coexist through coarse grain, and fine grain architectural interactions. One can be integrated within the other, beyond the perspective of run times and operating systems described above.

Von Neumann within CIM model allows for Von Neumann components executing within CIM, for example, in support of control functions, or performing more general operations.

CIM within Von Neumann model can result by using CIM as Von Neumann system memory, enabling built-in memory acceleration on an otherwise traditional Von Neumann architecture.

IV. SECURITY, VIRTUALIZATION, RESOURCE MANAGEMENT

Security has often been considered as an afterthought, with performance and reliability always dominating the requirements. As a consequence numerous bugs, leaks, and exploits are consistently being discovered. A complete new architecture paradigm opens a terrific opportunity to reconsider security as a first class requirement. Along with security come the tightly related requirements for virtualization and resource management.

A. *Security*

Security can benefit from several aspects of CIM. Packet based communication is better understood than the shared memory model with multiple threads accessing shared memory. Paths can be better secured by partitioning and data can be inspected prior and after entering and exiting CIM model, and therefore making higher security guarantees. A dataflow architecture can introduce barriers as a containment mechanism to stop propagation of errors and bugs. Affected stationary data does not propagate and can be contained where it is stored without access by other components. Packets in flight can be encrypted and networking key protection model can be readily applied. Data can be verified against the processing element and vice versa [72]. Finally, even though CIM relies on dataflow, some data sharing may be required to enable common data pools across the layers of CIM. Fine grained protection, for example based on capabilities such as CHERI [73], would be the ideal complement to further enhance the security model.

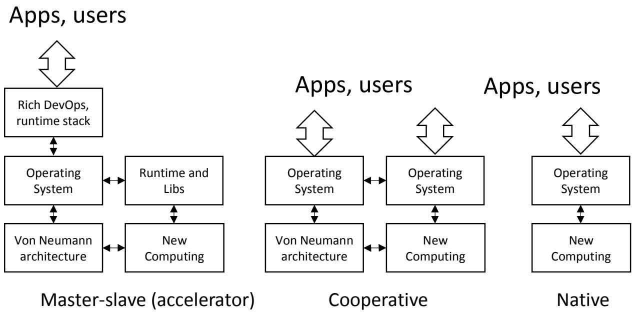

Fig 6. Evolution of Computing in Memory

B. Virtualization and Partitioning

Similarly to security, virtualization and partitioning can substantially benefit from CIM. An intuitive analogy to the CIM model is Network Function Virtualization (NFV) in the networking space. NFV has been well understood and it supports an equivalent functionality of high end proprietary boxes running in software on commodity servers. Many network virtualization approaches can be directly applied to CIM model. In particular:

Dynamic hardware isolation: similarly to security containment, parts of the CIM components can be completely isolated from other parts for security reasons.

Quality of service: minimal performance influence from one stream to another is achieved by provisioning enough interconnect. This is equally important for quality of service and to prevent leaking information across streams.

Failover: should streams be redirected for performance or reliability reasons, switching to other components would have minimal impact on performance.

C. Resource Management

Traditional load balancing techniques, such as distributing, pinning, and measuring loads also apply to CIM

Load information management is required before any action is undertaken. It assumes measuring latencies and bandwidth of each stream, as well as usage of individual and aggregate resources.

Load balancing can be accomplished by redirecting streams to underutilized SIM components. In certain cases to achieve guaranteed performance some of the streams may need to be pinned to given CIM modules. In other cases, they can be free to dynamically assign or reassign.

Enabling closed loops means that performance of certain parts of the CIM modules may influence others, which can be used to manage performance according to given SLA agreements.

V. Non-Functional Characteristics

Non-functional characteristics of new hardware and computing-memory models are equally important for its success as the functional characteristics. In this section we discuss failure tolerance, scaling, configurability, and serviceability.

A. Failure tolerance

Reliability and fault tolerance techniques, such as fault detection, containment, prevention, and recovery need to be revisited to take into account the CIM characteristics.

Fault detection can be accomplished at any component level, starting from the micro-unit in our example, through higher level components. Detection can use extra bits on data or instruction states. Faults could be detected from micro-unit to the largest unit.

Fault containment is required once a fault is detected to prevent it from spreading it further (and cause silent data corruption, for example). Boundaries of each component are the convenient place which can be shut down for exception handling in case a fault is detected.

Fault prevention can be accomplished through redundancy of information and components. Any component can be replicated, just like information can be protected using ECC. In CIM, redundancy can be achieved at every layer. Compared to traditional systems, there is more symmetry among layers so similar techniques could be used.

Fault recovery by failing over to redundant components. For more reliable computation, the data can be held in preceding components until computation is completed or in case of failure redirected to another component.

As we can see, there is a lot of similarity with traditional resilience approaches. We believe that the dataflow nature of CIM, and the reliance on implicit message passing rather than shared memory, results in more reliable systems (Table 1).

Table 1 Comparison of Different Approaches to Computing

Comparison	**Approaches to Computing**		
	Von Neumann		**In-Memory**
	Parallel (shared memory)	**Distributed**	
programming model (common)	multi-threaded	message passing	dataflow
scaling (per system)	100s of cores (eg HPE Hawks)	200 racks (e.g. Exascale)	no perceived limit, higher then exascale
failure tolerance	whole partition fails	failover to another machine	stream redirection to redundant unit
security	whole partition	machine boundary	packet and stream based
robustness	OS-dependent	cluster dependent	application-specific

B. Scaling

Scaling CIM is relatively straightforward; it is in many ways similar to scaling web server farms if the individual elements are stateless and only execute data streams. If the CIM modules are stateful, scaling is more complicated: it requires scaling of each class of the modules and then spreading the state across added modules. It also require interactions with the end-to-end application.

C. Configurability

There are many design points that enable reconfiguration of a CIM architecture. Different precision and number of bits can be configured at the lowest level. Reconnecting components enables reconfiguration at higher levels. This would be similar to Coarse Grained Reconfigurable Architectures (CGRA) [58] and systems, such as ADRES [74], PipeRench [62], and MorphoSys [75].

D. Serviceability

Deployed equipment is increasingly hard to support, which is even more important for systems at the edge. This motivates the need for graceful aging and self-healing at multiple levels of CIM components. Understanding how individual devices age can enable switching them out of active configurations preventing failures from even happening. If nothing else helps, closed loops enable more reliable functioning of deployed CIM modules: from device to central management, from device/management layer to support agents; and from device/management support agents to design engineers.

E. Discussion

Table 1 compares the different approaches to computing. Because of its streaming nature, the dataflow and the networking models are similar. There is no perceived limit on scale other than in terms of power (and cost), which for CIM is better than traditional due to the adoption of new technologies used, such as memristors [20]. Failing components can be replaced by redundant units and packets resent either from the source or from cached component. Security is similar to networks, where packets in flight are encrypted. Compared to other models, robustness is application-specific because a lot of application code is built into the silicon.

VI. Dot Product Engine: performance, power, scale

At Hewlett Packard labs, we have pursued a form of CIM under the umbrella of the Dot Product Engine project. In the light of categories discussed in Section III.B, we have implemented static data flow CIM model which enables us to program and reconfigure the CIM for classes of neural networks. This is the follow-on work to the ISAAC architecture [49], which we extended to be more programmable. More detailed description and performance are beyond the scope of this paper and are under submission elsewhere, here we provide orders of magnitude performance benefits.

For neural network class of applications (), we achieved latency between 10 and 10^4 times better than CPUs and between 10 and 10^2 better than GPUs; we achieved bandwidth better 10^3-10^6 times compared to modern CPUs and comparable to modern GPUs. In terms of power, benefits are even larger, power is 10^3-10^6 better than CPUs and 10-10^3 better than GPUs. Therefore the performance benefits of using CIM are real. Of course they cannot be generalized to all applications.

In terms of scale, we consider acceptable scaling to existing neural networks by having multiple boards interconnected through standard and proprietary interconnects. Most of the challenges we expect in terms of hiding the asymmetric latency for writing memristors based devices.

VII. Summary and Next Steps

In this paper, we have motivated the need for adopting the new computing architecture, Computing-in-Memory. We discussed use cases where it could be beneficial, as well as applications. We then presented the CIM model in more detail, discussed security and other non-functional characteristics and compared them against those in von Neumann architecture.

Computing in Memory is already being demonstrated in research and slowly adopted in product prototypes. Whether it will take off is something that only future applications will demonstrate. However, it is important to avoid repeating mistakes from the past. These are there most important aspects that need further in-depth exploration: exploiting inherent application parallelism; security; and software development productivity.

Recognizing dominant applications of the future that are suitable for CIM will also depend on the application inherent parallelism and developing corresponding architectural support to exploit this parallelism.

As architects and systems and applications developers start to design solutions, security needs to be treated as first level requirement, if we do not want to be haunted by the bugs and vulnerabilities of the past.

Finally, the largest cost in systems solutions lies in software development. Therefore whatever computing models are developed will also require conducive ways of software development.

Acknowledgments

We thank reviewers for very useful feedback. We also thank Izzat El Hajj of UIUC and Aayush Ankit of Purdue for reviewing the earlier version of the paper and for their valuable feedback. Our colleagues from Hewlett Packard Labs, especially Harumi Kuno and Avi Mendelson of Technion, provided feedback on the latest draft of the paper. Their feedback substantially helped improve the content and presentation of this report.

Appendix A Application Suitability for CIM

Not all applications are well-suited for CIM and vice versa. In Table 2, we compare a few classes of applications in terms of computation, memory, and communication demands. We then try to assess whether the applications as well specific characteristics are suited for CIM. While it is beyond the scope of this paper to be precise in terms of classification (we would require detailed measurements, configurations of applications, configurations of computers, etc.), some patterns emerge.

Looking at the different columns, CIM benefits from applications characterized by low computation, high data, high operational intensity, low communication, and high parallelism. Applications with substantial computation needs are better suited with the traditional Von Neumann architectures. Similar is true for applications with substantial communication needs.

Looking at the different rows, machine learning, neural networks, graph problems and data bases (analytics) can benefit from CIM. KVs, data bases with transactions and FEM can benefit moderately, while others will not benefit.

It is also important to point out that CIM can be very successfully combined with the traditional Von Neumann models. In many use cases where multiple applications co-exist, some suited for CIM, other for Von Neumann, a combined distributed or parallel architecture could be deployed.

The future work would constitute conducting a more thorough workload characterization and analysis akin to dwarfs papers [76][77].

Table 2. Suitability of different classes of applications to CIM model and vice versa

Class of application	Characteristics						
	Compute intensive	Data intensive		Operational intensity Flop/byte (temporal locality)	Communication (iterative)	Parallelism (dependencies)	CIM
		Bandwidth	size				
Machine learning	high	high	high	high	low	high	high
Neural Networks	high	high	high	high	low	high	high
Graph problems (FB, intel.)	low	medium	high	high	high	high	high
Bayesian inference	high	low	low	high	high	medium	low
Markov chain	high	low	low	low	high	high	low
KVSs (persistency layer)	low	high	high	low	medium	high	medium
Data Bases (analytics)	low	high	high	low	medium	high	high
Data Bases (transactions)	medium	high	low to med.	high	high	medium	medium
Search (indexing problem)	high	high	high	high	high	high	low
Optimization problem (resource allocation)	high	low	low	high	high	low	low
Scientific Computing	high	low to high	low to high	medium	high	high	low
Finite Element Modelling	high	low	medium	medium	high	high	medium
Collaborative (mail, chat,..)	low	high	medium	low	high	low	low
Signal (image) processing	high	high	high	low	high	medium	low
CIM	low	high	high	high	low	high	

REFERENCES

[1] J. von Neumann, Report on the EDVAC, June 30, 1945.

[2] D.E. Culler, J.P. Singh, and A. Gupta, “Parallel Computer Architecture, Morgan Kaufmann, 1999.

[3] W.J. Dally and B. Towles, “Principles and Practices of Interconnection Networks, Morgan Kaufmann, 2004.

[4] D.A. Patterson and J.L. Hennessy, “Computer Organization and Design, The Hardware/Software Interface,” Morgan Kaufmann, 2009.

[5] Mary Hall, Peter Kogge, Jeff Koller, Pedro Diniz, Jacqueline Chame, Jeff Draper, Jeff LaCoss, John Granacki, Jay Brockman, Apoorv Srivastava, William Athas, Vincent Freeh, Jaewook Shin, Joonseok Park, “Mapping irregular applications to DIVA, a PIM-based data-intensive architecture”, Proceedings of the 1999 ACM/IEEE conference on Supercomputing, 57.

[6] PM Kogge, JB Brockman, T Sterling, G Gao, “Processing in memory: Chips to petaflops,” Workshop on Mixing Logic and DRAM: Chips that Compute and Remember at ISCA 97.

[7] PM Kogge, JB Brockman, VW Freeh, “PIM architectures to support petaflops level computation in the HTMT machine,” Innovative Architecture for Future Generation High-Performance Processors and Systems, 1999. International Workshop, pp 35-44.

[8] D. Patterson et al., “A Case for Intelligent RAM,” IEEE Micro, 1997.

[9] P. Faraboschi, K. Keeton, T. Marsland, D. Milojicic, “Beyond Processor-centric Operating Systems,” Proceedings of the 15th USENIX Workshop on Hot Topics in Operating Systems (HotOS XV), 2015, Kartause Ittingen, Switzerland.

[10] Dennard, Robert H.; Gaensslen, Fritz; Yu, Hwa-Nien; Rideout, Leo; Bassous, Ernest; LeBlanc, Andre (October 1974). "Design of ion-implanted MOSFET's with very small physical dimensions". IEEE Journal of Solid State Circuits. SC–9 (5).

[11] McMenamin, Adrian (April 15, 2013). "The end of Dennard scaling". Retrieved January 23, 2014.

[12] Moore, Gordon E. (1965-04-19). "Cramming more components onto integrated circuits". Electronics. Retrieved 2016-07-01.

[13] Moore, Gordon (2006). "Chapter 7: Moore's law at 40". In Brock, David. Understanding Moore's Law: Four Decades of Innovation. Chemical Heritage Foundation. pp. 67–84. ISBN 0-941901-41-6. Retrieved March 15, 2015.

[14] B. Schaller. The Origin, Nature, and Implications of “Moore’s Law” (September 26, 1996).

[15] Brad Chacos and Michael Simon, “Meltdown and Spectre FAQ: How the critical CPU flaws affect PCs and Macs”, PC World, https://www.pcworld.com/article/3245606/security/intel-x86-cpu-kernel-bug-faq-how-it-affects-pc-mac.html

[16] Conte, Thomas A., DeBenedictis, Erik, Mendelson, Avi, Milojicic, D., "Rebooting Computers to Avoid Meltdown and Spectre" in Computer, vol. 51, no. 4, April 2018, to appear.

[17] IEEE Rebooting Computing, https://rebootingcomputing.ieee.org/.

[18] Wei-Hao Chen, Wen-Jang Lin, Li-Ya Lai, Shuangchen Li, Chien-Hua Hsu, Huan-Ting Lin, Heng-Yuan Lee, Jian-Wei Su, Yuan Xie, Shyh-Shyuan Sheu, Meng-Fan Chang, “A 16Mb dual-mode ReRAM macro with sub-14ns computing-in-memory and memory functions enabled by self-write termination scheme,” Proceedings of the 2017 IEEE International Conference on Electron Devices Meeting (IEDM), December 2017.

[19] S. Jain, A. Ranjan, K. Roy and A. Raghunathan, "Computing in Memory With Spin-Transfer Torque Magnetic RAM," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. PP, no. 99, pp. 1-14. doi: 10.1109/TVLSI.2017.2776954

[20] Julien Borghetti, Gregory S. Snider, Philip J. Kuekes, J. Joshua Yang, Duncan R. Stewart, R. Stanley Williams, “‘Memristive’ switches enable ‘stateful’ logic operations via material implication,” NATURE, Vol 464/8 April 2010, pp873-876.

[21] Erik P. DeBenedictis ; Jeanine Cook ; Sriseshan Srikanth ; Thomas M. Conte, “Superstrider associative array architecture: Approved for unlimited unclassified release: SAND2017-7089 C.,” Proceedings of the High Performance Extreme Computing Conference (HPEC), 2017 IEEE.

[22] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry. 2017. Ambit: in-memory accelerator for bulk bitwise operations using commodity DRAM technology. In

Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50 '17). ACM, New York, NY, USA, 273-287.

[23] 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," 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, 2016, pp. 1-13. doi: 10.1109/MICRO.2016.7783753

[24] F. K. Hsueh et al., "TSV-free FinFET-based Monolithic 3D+-IC with computing-in-memory SRAM cell for intelligent IoT devices," 2017 IEEE International Electron Devices Meeting (IEDM), San Francisco, CA, USA, 2017, pp. 12.6.1-12.6.4. doi: 10.1109/IEDM.2017.8268380

[25] K. Hsieh et al., "Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems," ISCA, 2016.

[26] Milojicic, D., Douglis, F., Paindaveine, Y., Wheeler, R., Zhou, S, "Process Migration Survey", ACM Computing Surveys, vol 32, no 3, September 2000, pp 241-299.

[27] C. Dou et al., "Challenges of emerging memory and memristor based circuits: Nonvolatile logics, IoT security, deep learning and neuromorphic computing," 2017 IEEE 12th International Conference on ASIC (ASICON), Guiyang, 2017, pp. 140-143. doi: 10.1109/ASICON.2017.8252431

[28] Goodfellow, Ian, et al. Deep learning. Vol. 1. Cambridge: MIT press, 2016

[29] IEEE Ethically Aligned Design (EAD), Version 2, A Vision for Prioritizing Human Well-being with Autonomous and Intelligent Systems.

[30] IEEE Technology Trend Paper: Artificial Intelligence and Machine Learning Applied to Cybersecurity, 2018.

[31] D. G. Elliott, M. Stumm, W. M. Snelgrove, et al. Computational ram: implementing processors in memory. IEEE Design Test of Computers, 16(1):32–41, Jan 1999.

[32] Jay B. Brockman, Shyamkumar Thoziyoor, Shannon K. Kuntz, and Peter M. Kogge. 2004. A low cost, multithreaded processing-in-memory system. In Proceedings of the 3rd workshop on Memory performance issues: in conjunction with the 31st international symposium on computer architecture (WMPI '04). ACM, New York, NY, USA, 16-22.

[33] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A scalable processing-in-memory accelerator for parallel graph processing. In Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA '15). ACM, New York, NY, USA, 105-117.

[34] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. PIM-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture. In Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA '15). ACM, New York, NY, USA, 336-348.

[35] Soroosh Khoram, Yue Zha, Jialiang Zhang, and Jing Li. 2017. Challenges and Opportunities: From Near-memory Computing to In-memory Computing. In Proceedings of the 2017 ACM on International Symposium on Physical Design (ISPD '17). ACM, New York, NY, USA, 43-46.

[36] Dimitrios Skarlatos, Nam Sung Kim, and Josep Torrellas. 2017. Pageforge: a near-memory content-aware page-merging architecture. In Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50 '17). ACM, New York, NY, USA, 302-314.

[37] J. Picorel, D. Jevdjic and B. Falsafi, "Near-Memory Address Translation," 2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT), Portland, OR, 2017, pp. 303-317. doi: 10.1109/PACT.2017.56

[38] A. Boroumand, S. Ghose, M. Patel, H. Hassan, B. Lucia, K. Hsieh, K. T. Malladi, H. Zheng, and O. Mutlu, "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," IEEE Computer Architecture Letters (CAL), June 2016.

[39] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello and Y. LeCun, "NeuFlow: A runtime reconfigurable dataflow processor for vision," CVPR 2011 WORKSHOPS, Colorado Springs, CO, 2011, pp. 109-116.

[40] Amir Morad, Leonid Yavits, Shahar Kvatinsky, and Ran Ginosar. 2016. Resistive GP-SIMD Processing-In-Memory. ACM Trans. Archit. Code Optim. 12, 4, Article 57 (January 2016), 22 pages.

[41] Pedro Trancoso. 2015. Moving to memoryland: in-memory computation for existing applications. In Proceedings of the 12th ACM International Conference on Computing Frontiers (CF '15). ACM, New York, NY, USA, , Article 32 , 6 pages.

[42] Dong Ping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph Greathouse, Mitesh Meswani, Mark Nutter, and Mike Ignatowski. 2013. A new perspective on processing-in-memory architecture design. In Proceedings of the ACM SIGPLAN Workshop on Memory Systems Performance and Correctness (MSPC '13). ACM, New York, NY, USA, Article 7 , 3 pages.

[43] Dongping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph L. Greathouse, Lifan Xu, and Michael Ignatowski. 2014. TOP-PIM: throughput-oriented programmable processing in memory. In Proceedings of the 23rd international symposium on High-performance parallel and distributed computing (HPDC '14). ACM, New York, NY, USA, 85-98.

[44] Leibin Ni, Hantao Huang, Zichuan Liu, Rajiv V. Joshi, and Hao Yu. 2017. Distributed In-Memory Computing on Binary RRAM Crossbar. J. Emerg. Technol. Comput. Syst. 13, 3, Article 36 (March 2017), 18 pages.

[45] J. Zhan et al., "A unified memory network architecture for in-memory computing in commodity servers," 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, 2016, pp. 1-14.

[46] CCIX, Cache Coherent Interconnect for Accelerators, www.ccixconsortium.com.

[47] GenZ Consortium, genzconsortium.org/.

[48] OpenCAPI, Open Coherent Accelrerator Processor Interface, opencapi.org.

[49] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. 2016. ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16). IEEE Press, Piscataway, NJ, USA, 14-26.

[50] G. Khodabandehloo, M. Mirhassani, and M. Ahmadi.Analog implementation of a novel resistive-type sigmoidal neuron. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 20(4):750-754,April 2012.

[51] J. Torrellas, "FlexRAM: Toward an advanced Intelligent Memory system: A retrospective paper," 2012 IEEE 30th International Conference on Computer Design (ICCD), Montreal, QC, 2012, pp. 3-4. doi: 10.1109/ICCD.2012.6378607

[52] Chenchen Liu, Bonan Yan, Chaofei Yang, Linghao Song, Zheng Li, Beiye Liu, Yiran Chen, Hai Li, Qing Wu, and Hao Jiang. 2015. A spiking neuromorphic design with resistive crossbar. In Proceedings of the 52nd Annual Design Automation Conference (DAC '15). ACM, New York, NY, USA,, Article 14, 6 pages.

[53] Caxton C. Foster. 1976. Content Addressable Parallel Processors. John Wiley & Sons, Inc., New York, NY, USA

[54] Qing Guo, Xiaochen Guo, Yuxin Bai, and Engin İpek. 2011. A resistive TCAM accelerator for data-intensive computing. In Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44). ACM, New York, NY, USA, 339-350.

[55] Isaac D. Scherson and Sener Ilgen. 1989. A Reconfigurable Fully Parallel Associative Processor. J. Parallel Distrib.Comput. 6, 1 (Feb. 1989), 69–89.

[56] L. Yavits, S. Kvatinsky, A. Morad, and R. Ginosar. 2015. Resistive Associative Processor. 14, 2 (July 2015), 148–151.

[57] L. Yavits, A. Morad, and R. Ginosar. 2015. Computer Architecture with Associative Processor Replacing Last-Level Cache and SIMD Accelerator. IEEE Trans. Comput. 64, 2 (Feb 2015), 368–381.

[58] M. Wijtvliet, L. Waeijen and H. Corporaal, "Coarse grained reconfigurable architectures in the past 25 years: Overview and

classification," 2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), Agios Konstantinos, 2016, pp. 235-244. doi: 10.1109/SAMOS.2016.7818353

[59] J. Cong and B. Xiao, "FPGA-RPI: A Novel FPGA Architecture With RRAM-Based Programmable Interconnects," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 22, no. 4, pp. 864-877, April 2014. doi: 10.1109/TVLSI.2013.2259512

[60] Yue Zha and Jing Li. 2016. "Reconfigurable in-memory computing with resistive memory crossbar." In Proceedings of the 35th International Conference on Computer-Aided Design (ICCAD '16). ACM, New York, NY, USA, Article 120, 8 pages.

[61] M. Budiu, G. Venkataramani, T. Chelcea et al. Spatial computation. ASPLOS XI, pages 14–26, New York, NY, USA, 2004. ACM.

[62] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe and R. R. Taylor, "PipeRench: a reconfigurable architecture and compiler," in Computer, vol. 33, no. 4, pp. 70-77, Apr 2000. doi: 10.1109/2.839324

[63] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe and R. R. Taylor, "PipeRench: a reconfigurable architecture and compiler," in Computer, vol. 33, no. 4, pp. 70-77, Apr 2000. doi: 10.1109/2.839324

[64] The OpenSPL Consortium. Openspl: Revealing the power of spatial computing. Technical report, Dec. 2013.

[65] Jintao Yu, Tom Hogervorst, and Razvan Nane. 2017. A Domain-Specific Language and Compiler for Computation-in-Memory Skeletons. In Proceedings of the on Great Lakes Symposium on VLSI 2017 (GLSVLSI '17). ACM, New York, NY, USA, 71-76.

[66] Sparsh Mittal. 2016. A Survey of Techniques for Approximate Computing. ACM Comput. Surv. 48, 4, Article 62 (March 2016), 33 pages.

[67] P. Dlugosch, D. Brown, P. Glendenning, et al. An efficient and scalable semiconductor architecture for parallel automata processing. TPDS, 25(12):3088–3098, Dec 2014.

[68] O. Pell, O. Mencer, K. H. Tsoi et al. Maximum Performance Computing with Dataflow Engines, pages 747–774. Springer New York, New York, NY, 2013.

[69] W-m. Hwu (edited by), "Heterogeneous System Archtiecture," Morgan Kaufmann, 2016.

[70] Bruel, P., Chalamalasetti, S., Dalton, C., El Hajj, I., Goldman, A., Graves, C., Hwu, W.-M., Laplante, P., Milojicic, D., Ndu, G., Strachan, J.P., "Generalize or Die: Operating Systems Supportfor Memristor-based Accelerators," paper invited at the 2nd International Conference on Rebooting Computing, Washington DC, November 2017.

[71] D. Milojicic and T. Roscoe, "Outlook on Operating Systems," in Computer, vol. 49, no. 1, pp. 43-51, Jan. 2016. doi: 10.1109/MC.2016.19

[72] Achermann, R., Dalton, C., Faraboschi, P., Hoffmann, M., Milojicic, D., Ndu, G., Richardson, A., Roscoe, T., Watson, R..N.M., "Separating Translation from Protection in Address Spaces with Dynamic Remapping," Proceedings of the ACM 16th Workshop on Hot Topics in Operating Systems, 118-124.

[73] Robert N. M. Watson, Jonathan Woodruff, Peter G. Neumann, Simon W. Moore, Jonathan Anderson, David Chisnall, Nirav Dave, Brooks Davis, Khilan Gudka, Ben Laurie, Steven J. Murdoch, Robert Norton, Michael Roe, Stacey Son, and Munraj Vadera. CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization, Proceedings of the 36th IEEE Symposium on Security and Privacy ("Oakland"), San Jose, California, USA, May 2015.

[74] Mei B., Vernalde S., Verkest D., De Man H., Lauwereins R. (2003) ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix. In: Y. K. Cheung P., Constantinides G.A. (eds) Field Programmable Logic and Application. FPL 2003. Lecture Notes in Computer Science, vol 2778. Springer, Berlin, Heidelberg.

[75] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. C. Filho, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications," IEEE Transactions on Computers, vol. 49, no. 5, May 2000.

[76] Konstantinos Krommydas, Wu-Chun Feng, Christos D. Antonopoulos, and Nikolaos Bellas. 2016. OpenDwarfs: Characterization of Dwarf-Based Benchmarks on Fixed and Reconfigurable Architectures. J. Signal Process. Syst. 85, 3 (December 2016), 373-392.

[77] Asanović, Krste and Bodik, Ras and Catanzaro, Bryan Christopher and Gebis, Joseph James and Husbands, Parry and Keutzer, Kurt and Patterson, David A. and Plishker, William Lester and Shalf, John and Williams, Samuel Webb and Yelick, Katherine A., "The Landscape of Parallel Computing Research: A View from Berkeley", TR UCB/EECS-2006-183, EECS Department, University of California, Berkeley, December 2006.