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 different reliability, scale, configurability, and security.

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].

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].

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 III presents the CIM model, including logic and core architecture, analogy to object oriented systems, 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. Finally, Section VI discusses the next steps.

Fig 1. Von Neumann Architecture
II. BACKGROUND AND MOTIVATION

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, that’s where we started. 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 unnecessarily 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.

**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.

![Memory bandwidth per processor floating point operations (FLOP)](image)
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.

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. It is today 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.

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?

![Fig 3. Evolving CIM Model](image_url)

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 three 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 increasing dependent on automation, IT and cybersecurity, which in turn is enabled by more powerful computing.

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.

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).

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]; neuro-morphic 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][66] 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 coloration 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.

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

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.
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.

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 will therefore 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.

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

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).

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.

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Approaches to Computing</th>
<th>In-Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming model (common)</td>
<td>multi-threaded</td>
<td>dataflow</td>
</tr>
<tr>
<td>Scaling (per system)</td>
<td>100s of cores (e.g. HPE Hawks)</td>
<td>200 racks (e.g. Exascale)</td>
</tr>
<tr>
<td>Failure tolerance</td>
<td>whole partition fails</td>
<td>failover to another machine</td>
</tr>
<tr>
<td>Security</td>
<td>whole partition</td>
<td>machine boundary</td>
</tr>
<tr>
<td>Robustness</td>
<td>OS-dependent</td>
<td>cluster dependent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>application-specific</td>
</tr>
</tbody>
</table>

VI. 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, such as not building in security requirements. As architects, systems and applications developers start to develop 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.
If the paper gets accepted, we will expand it to a full paper and provide quantification of functional and non-functional characteristics, such as performance, power, scale, and reliability. We will also provide examples of algorithms that are suitable for CIM and describe technologies behind CIM, such as memristors. We will also provide a description of the system we are working on that has some elements of the CIM.

ACKNOWLEDGMENTS

We would like to thank Izzat El Hajj of UIUC and Aayush Ankit of Purdue, as well as our colleagues from Hewlett Packard Labs for reviewing the earlier version of the paper and for their valuable feedback. It helped improved the content and presentation.

REFERENCES

[34] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. PIM-enabled instructions: a low-overhead, locality-aware processing-in-


