Computation-In-Memory Based Parallel Adder

Hoang Anh Du Nguyen, Lei Xie, Mottaqiallah Taouil, Razvan Nane, Said Hamdioui, Koen Bertels
Laboratory of Computer Engineering, Faculty of EE, Mathematics and CS
Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands
Email: H.A.DuNguyen@tudelft.nl

***Abstract*—Today's computing systems suffer from memory/communication bottleneck, resulting in energy and performance inefficiency. This makes them incapable to solve data-intensive applications within economically acceptable limits. Computation-In-Memory (CIM) architecture, based on the integration of storage and computation in the same physical location using non-volatile memristor technology offers a potential solution for the memory bottleneck. This paper presents a CIM based parallel adder, and shows its potentials and superiority for intensive computing and massive parallelism by comparing it with state-of-the art computing systems including multicore, GPU and FPGA architecture. The results show that CIM based parallel adder can achieve at least two orders of magnitude improvement in computational efficiency, energy efficiency and area efficiency.**

I. Introduction

In the last several decades, CMOS down-scaling has been the primary driver behind computer performance improvement [1]. However, CMOS technology is reaching its physical -if not economical- limits [2]. Down-scaling devices has led to many challenges such as leakage power [2], reliability [3], fabrication process and turnaround time [4], test complexity [4], cost for mask and design [5], and yield [6]. Furthermore, the performance gain by increasing clock speed has saturated since early 2000 [7]; today, speed-up is no longer the result of a faster clock, but rather a result of parallelization on multi-core and many-core systems. However, the number of parallel cores that can be programed and the computation efficiency that can be extracted are tending to saturate as well [8]. Obviously, all today's computing systems are mainly built on John von Neumann stored-program computer concept [9]. A major drawback of this computer design is the gap between the processing units and the main memory, the so-called memory bottleneck [7,10]. For data-intensive applications, the memory bottleneck is becoming even more severe and is putting major limitations both on performance and energy consumption. All of these motivate the need for a new architecture being able to (a) eliminate the communication bottleneck and support massive parallelism to increase the overall performance, (b) reduce the energy inefficiency to improve the computation efficiency.

Getting the memory closer the processing unit and reducing the memory bottleneck has attracted a lot of attention. In 1969, Logic-In-Memory (LIM) was originally introduced as a memory accelerator [11]; i.e., add some processing units close to main memory. In 1992, LIM concept re-appeared and named computational RAM, and typically uses the same accelerator concept where these are supposed to perform operations needed by the memory such as address translations [12]. In the late 1990s and early 2000s, Processor-In-Memory (PIM) was proposed [13] and manufactured [14]. PIM is based on splitting the main memory in different parts, each with surrounded computing units to bring the computation near to the memory; the architecture has a master CPU that takes care of the overall control. PIM concept was later used and refined for different applications; examples are EXECUBE [15], IRAM [16], FlexRAM [17], DIVA [18], Gilgamesh [19]. In 2004, Memory-In-Logic (MIL), which provides massive addressable memory on the processor, was proposed for supercomputer systems [20]. All mentioned above efforts have tried to close the gap between processor and memory speed [21]. However, as the computation and the storage are kept separately, they fundamentally use von Neumann stored-program computer concept and therefore suffer from memory bottleneck, which negatively impacts the performance [7].

This paper uses Computation-In-Memory (CIM) concept, that we have recently developed [22], to design a parallel adder and illustrate the huge potential of such an architecture for a simple case study: intensive arithmetic operations (additions). The architecture uses a revolutionary approach based on (a) the integration of storage and computation in the same physical location, and (b) non-volatile memristor technology [23]. It is worth noting that adding multiple numbers is a basic yet very representative operation in big data applications [24]. In existing architectures (e.g., multicore, GPU, and FPGA), simple operations such as adding multiple numbers already face the memory bottleneck. As the processors have to fetch huge amounts of data from memory, the intrinsic parallelism cannot be exploited fully in such architectures.

The main contributions of this paper are:

- A CIM based parallel adder for intensive computing.
- The evaluation of the proposed adder and comparison of its performance with traditional architectures including multicore, GPU and FPGA.

The proposed design achieves at least two orders of magnitudes improvements for big problems!

The rest of this paper is structured as follows. Section II briefly describes the concept of CIM architecture, and presents the CIM parallel adder. Section III provides estimations of CIM parallel adder's performance and compares it with other traditional architectures. Section IV shows our evaluation results and analysis. Finally, section V concludes this paper.

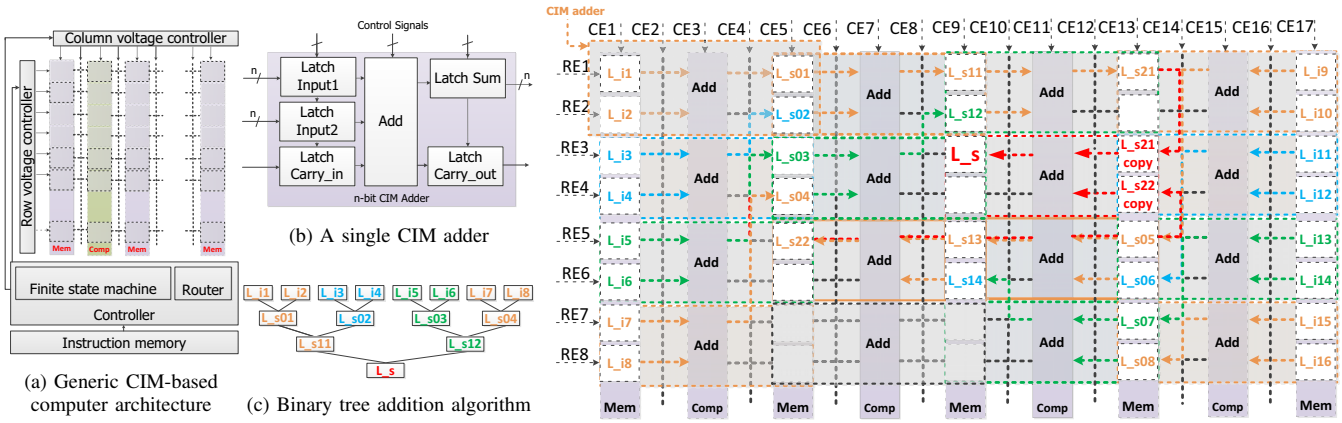
II. CIM Parallel Adder

This section briefly first describes the CIM architecture. Thereafter, it presents the concept of the parallel adder. Finally, it demonstrates how to map these adders efficiently on the crossbar architecture.

A. Generic CIM Computer Architecture

The main advantage of CIM architecture over von-Neumann architectures is the tight integration of both computing

978-1-4673-7849-9/15/$31.00



(a) Generic CIM-based computer architecture

(b) A single CIM adder

(c) Binary tree addition algorithm

(d) CIM adders mapped onto CIM architecture

Fig. 1: CIM parallel adder

and storing operations using the same physical crossbar. Hence, massive parallelism is achieved with minimized communication.

Fig. 1a shows the main three CIM architecture components: crossbar, communication network and controller. The crossbar consists of memristors that are used to implement logic functions and/or storage. The communication network is either implemented within the crossbar or by using separate metal layers. A controller that is implemented by CMOS devices handles auxiliary operations such as distributing data and controlling signals to the crossbar.

The crossbar is specialized to perform computation and storage operations in cells organized in rows and columns. Each cell can be a computational unit (such as an adder or multiplier) or storage location (such as a memory cell). The cells in a row or column can be configured with the same or different functionality. The communication in CIM architecture has maximum flexibility. The architecture allows bi-directional communication in both horizontal and vertical direction. The controller contains a router and a finite state machine (FSM). The router provides the FSM with a communication scheme for data distribution and movements. The FSM fetches instructions from an instruction memory (e.g. hard disk), converts fetched instructions to controlling signals for the row/column voltage controller.

In this paper, our focus is to investigate the effectiveness of the crossbar with respect to computation and storage. Details on controller and communication are under further investigation.

B. A CIM-based Adder Tree

Fig. 1b shows a single CIM adder. The basic computational unit is an *n*-bit adder [25,26], which is surrounded by a number of memory cells (latches). An *n*-bit adder contains three *n*-bit latches (two for the inputs and one for the sum), a 1-bit carry-in and a 1-bit carry-out latch.

The CIM parallel adder arranges multiple CIM adders in a binary tree network. The carry-in and carry-out registers of an adder are connected properly to generate correct addition results. The binary tree network is ineffective using traditional platforms due to the difference between processor and memory fabrication. A processor coupled with a large amount of memory is unrealistic with traditional CMOS technology. Using the new features of the CIM architecture and its underlying memristor technology, the adder tree can be effectively mapped to reduce addition latency and increase resource utilization.

Fig. 1d shows a mapped binary adder tree with 16 inputs (see Fig. 1c for an 8 input example) on the CIM architecture. Each CIM adder corresponds to the adder presented in Fig 1b. Note that the output latches *sum* at each adder are reused as input latches in the next adder stage. The first column of the crossbar gets the first half of the inputs L_i. *Add* units in the second column add every two corresponding input latches L_i and store results in corresponding output latches L_s in the third column. The fourth column adds up results from output latches of the third column. Another direction of computation happens from the final column backwards to utilize as many resources as possible. In other words, a cell in CIM architecture is configured to an *add* unit or a latch. The interconnects (dotted lines) between multiple rows and columns represent communication channels among cells.

The crossbar array for N additions contains at least $\frac{N}{2} \times (log_2(N))$ CIM adders. Due to the multi-directional characteristic of the CIM-based adder, additions can operate in two direction flows (from left to right and vice versa) of the array (as shown in Fig. 1d). These bi-directional operations efficiently exploit resources in the architecture. Therefore, the architecture is designed with one additional column and half number of rows in comparison with the above-mentioned size. Hence, for N inputs the delay and array size equals $log_2(N)+1$ and $\frac{N}{4} \times (2log_2(N)+1)$ cells, respectively (each adder processes two inputs). With this design, every operation is performed on a distinct operational and storage unit; hence, there is no operation overlapping at a particular location. In addition, maximum number of adders in the architecture are used to avoid idle adders. With smart communication schemes, the architecture can be pipelined to increase overall performance.

III. Architectural Considerations

To illustrate how the CIM architecture improves the state-of-the-art, the performance of CIM is evaluated and compared

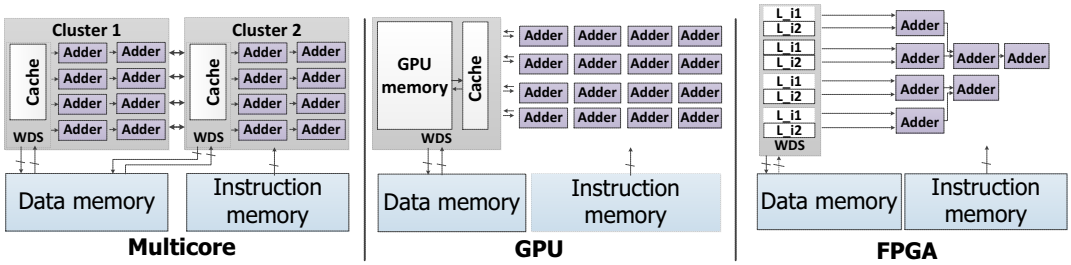


Fig. 2: Multicore vs GPU vs FPGA vs CIM parallel adders

against modern computer architectures using the addition test case. The binary tree addition algorithm (as shown in Fig. 1c) was mapped on four architectures. The following section describes the architecture models and performance metrics.

A. Architecture Models

For this estimation, we build simplified models that embed the basic characteristics of multicore, GPU and FPGA platforms. These models are referred to as multicore, GPU and FPGA architecture. To prevent an unfair comparison between the different architectures, optimistic assumptions are being made for the modern architectures (multicore, GPU and FPGA architecture) while pessimistic ones for CIM. For the multicore, GPU and FPGA architecture, an architecture with only parallel adder (not a complete processor) is used. For the CIM architecture, we use the dedicated adder tree of Fig. 1d. We assume that all modern architectures have processing units, working data sets, a proper controller and memories. Fig. 2 depicts assumed architecture for the multicore, GPU and FPGA. The CIM architecture, in contrast to the other architectures, only requires an instruction memory as the data is stored inside the crossbar. For the other architectures, a data memory is required. As this paper focuses on the computation and storage only, the cost for initial data load, controller and memories are not considered for all evaluated platforms.

The main assumptions for all the architectures are described in Table I. All of them are evaluated assuming 2^2 to 2^{21} inputs using 32-bit adders. For each architecture, the most optimistic available technology data is used. That is, multicore and GPU are based on data from 22nm technology, while the adder tree in the FPGA architecture was simulated with Virtex-7 (FFG1157 FPGA) using the 28nm technology library. For CIM architecture, assumptions are based on the ITRS 5nm memristors [28]. Though the physical memristor size has an order of magnitude benefit in comparison with other architectures, the estimation aims to show architectural impact more than technology dependence. In addition, worst-case assumptions are made for CIM architecture while the best-case for the other architectures. It is important to note that the comparison is one-by-one between CIM and other architectures. Indirect comparisons among multicore, GPU and FPGA are not relevant in this paper due to the optimistic assumptions made for each architecture.

Each processing unit is an adder that is organized depending on the architecture's characteristics. For the multicore architecture, 32 adders are grouped in a cluster (mimicking a 32-core system). Multiple clusters together form the architecture. Each adder gets its inputs from a cache. For the GPU architecture, each GPU core contains only a single adder. Hence, the GPU architecture has a large number of adders. The adders in multicore and GPU architecture are assumed to execute as many parallel tasks as possible. For FPGA architecture, the adders are connected in a binary tree network, the same one used in Fig 1c for CIM. The difference between multicore or GPU architecture versus FPGA or CIM architecture is whether the adders are organized in a binary tree network or not. Adders in FPGA and CIM architecture are organized using a binary tree addition network, while multicore and GPU architecture use as many adders as possible at every stage of computation. Note that the logarithmic addition algorithm is also used in these cases. However, the organization of adders impacts the area and power metrics of each architecture.

The working data set (WDS) differs for each architecture. Data is loaded initially from a data or program memory (e.g. hard disk, etc.) to this WDS. For multicore architecture, the WDS is an 8KB cache for each cluster. For GPU architecture, the WDS includes a 6GB GPU global memory and a 64KB global cache for the whole architecture. As data is loaded from memories, a particular hit rate, hit delay, miss rate, and miss penalty are assumed for multicore and GPU architecture. These memory characteristics are based on optimistic estimations on existing multicore and GPU architectures. In particular, the area and power data for cache is derived from data of 512KB-cache. The area and power of GPU global memory are provided by NVIDIA GPU datasheets [37]. As hit delay for a cache in multicore and GPU is assumed as fast as 1 cycle (with 1GHz clock rate), the register file is not considered in these architectures. For FPGA architecture, the WDS consists of a big register file. For CIM architecture, the WDS consists of interleaved latches with adders as described in Section II-B. Four architectures with different configurations of processing units and working data sets show the diversity of computer architectures in this estimation.

For multicore, GPU and CIM architecture, estimations were performed on assumptions listed in Table I. The FPGA implementation is generated by Vivado HLS tool [27] to ensure a good FPGA design and simulated by Xilinx ISE [27]. Due to large simulation time and limited FPGA chip area, measurements from a small-scale implementation simulation are scaled up for large-scale FGPA implementation. The calculations and measurements include only evaluated architecture components such as memory and adders.

TABLE I: Assumptions for each architecture

Arch.	Multicore	GPU		FPGA	CIM
Technology	22nm			28nm	5nm
Adder design	Rippled eight 4-bit Carry-look-ahead adder			auto-mapped on FPGA	Aachen Toggle-Cell-Adder
Area-parameter	208 CMOS gates [24]			$1.47mm^2$/adder [27]	34 memristors [25]
	$0.248um^2$/gate [28]				$100nm^2$
	$51.6um^2$/adder				$3400nm^2$
Delay-parameter	18 gates [24]			7.17ns/adder [27]	133 steps [25]
	9ps/gate [29,30]				200ps/step [31]
	162ps/adder				26600ps/adder
Energy-parameter	Static power + Dynamic power			0.0173W/adder [27]	Dynamic power
Leakage power consumption	$I_{leakage} * V_{dd}$ 6.15pA [28] * 0.86V				0
Dynamic power consumption	67mW/gate [29,32]				246fJ [28]
Memory design	8KB cache/cluster	64KB cache	6GB GPU global memory	Register file	Scalable memristor-based memory
Memory operating frequency	1GHz			No cache miss	No cache miss
Hit rate	0.95	0.90	0.995		
Load delay on hit	1 cycle	1 cycle	96 cycles [33,34]		
Missed penalty	165 cycles [35]	96 cycles [33,34]	165 cycles [35]		
Area	$0.0092mm^2$/cache [36]	$0.0737mm^2$/cache [36]	$529mm^2$/memory [37]		
Static power	0.0156W/cache [38]	0.125W/cache [38]	68W/memory [37]		
Dynamic power	25% static power	25% static power	25% static power		

Adders are designed specifically for their target architecture. Multicore and GPU architecture use a ripple carry adder with eight 4-bit Carry-Look-Ahead adders. The delay of an adder is calculated based on the gate delay and the number of required gates per adder in the longest path. Area is calculated based on the required number of CMOS gates per adder. The characteristics of a FPGA adder are extracted from ISE and shown in Table I. However, as the FPGA implementation is generated and mapped automatically by synthesis tools, the measurements of a large scale implementation are not scaled up using a single FPGA adder characteristics. Instead, an FPGA implementation of 256 adders is used to estimate larger implementations. CIM architecture uses a 32-bit Carry-Ripple-Adder based on memristor [25]. The delay of an adder is calculated by the number of steps to perform an addition. Each step corresponds to one memristor delay, which is the worst-case estimated as 200ps [31]. Area is calculated based on the required number of memristors per adder [25]. The above memristor-based adder is the fastest memristor-based adder available in literature. Meanwhile, other adder designs (multicore and GPU) are optimized for delay and area.

In order to make a realistic estimation, we investigated two cases: infinite resources and limited resources. In the first case, we assume architectures' resources is infinitely scalable. In the second case, we assume a maximum amount of resources based on existing devices that implement each architecture. In particular, the most recent finfet chip of 22nm data is used for multicore and GPU architecture while Virtex-7 chip data is used for FPGA architecture. When required resources cannot fit on a chip, we assumed multiple chips are used for multicore, GPU and FPGA architecture, respectively. Hence, extra area was counted for extra chips, which made delay and energy increase accordingly. We ignored additional delay and energy for transferring data among chips/GPUs/FPGAs. This makes their delay/area estimation optimistic. Memristor device currently has no implementation, hence we assumed maximum size of a memristor chip was $0.7mm^2$, in which 30% extra area is assumed for inter-chip communication. The chip area defined the limit of multicore architecture's area. Similar numbers for GPU and FPGA were chosen.

B. Performance Metrics

The estimation was performed and verified in Matlab. We consider three metrics: $total\_delay$ (D), $total\_energy$ (E), and $total\_area$ (A) described by Equation 1. Each parameter consists of two components: computation made by adders and communication made by local data memory access. The interconnection and controller are not considered in this estimation. Energy is calculated by delay and power. Both static power and dynamic power are considered for energy estimation. Static power is mainly caused by leakage current while dynamic power is consumed by switching activities. Computational activities related to working adders consume dynamic power while idle adders and WDS consumes mostly static power. CIM architecture is based on memristor; hence, it is claimed to consume no static power [39]. From the three above metrics, we derived three performance metrics: computation efficiency (η_C), energy efficiency (η_E) and area efficiency (η_A). Equations for these parameters are described in Equation 2.

$$\begin{aligned} D &= D_{comp} + D_{comm} \\ E &= E_{comp} + E_{mem} \\ A &= A_{comp} + A_{comm} \end{aligned} \quad (1)$$

$$\begin{aligned} \eta_C &= \frac{D * E}{\#ops} \\ \eta_E &= \frac{\#ops}{E} \\ \eta_A &= \frac{\#ops}{A} \end{aligned} \quad (2)$$

IV. Results

A. Infinite Resources

Fig. 3 shows the performance of the four architectures when resources are assumed to scale up infinitely.

With respect to delay, Fig. 3 shows that CIM architecture performs slowest among four architecture. As all four architectures perform the same *n*-additions using binary tree algorithm, the delay for computation is *log(n)* stages. The differences among four architectures are memory access delay and the amount of time to perform a single addition. As architectures with adders organized in a binary tree network (FPGA and CIM architecture) benefit from fewer data loads and stores, they have lower delay in memory accessing. In

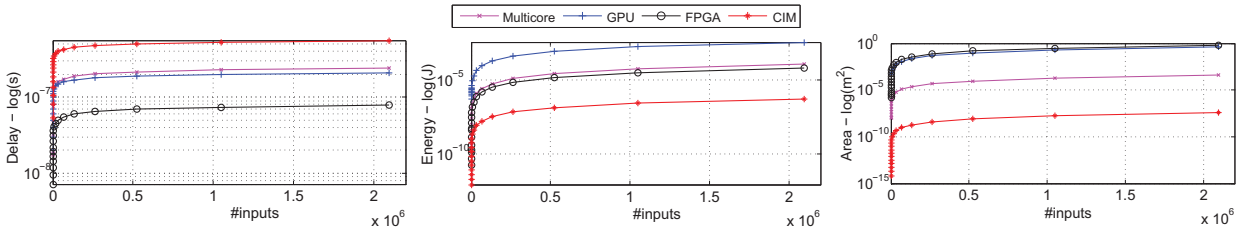


Fig. 3: Multicore vs GPU vs FPGA vs CIM parallel adders without resource constraints (intrinsic parameters)

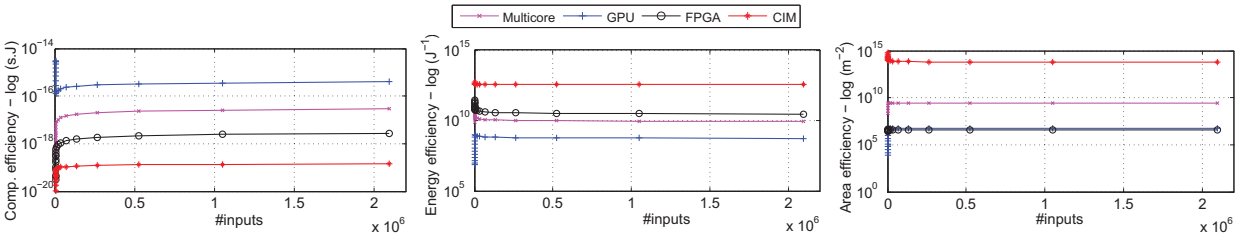


Fig. 4: Multicore vs GPU vs FPGA vs CIM parallel adders without resource constraints (derived parameters)

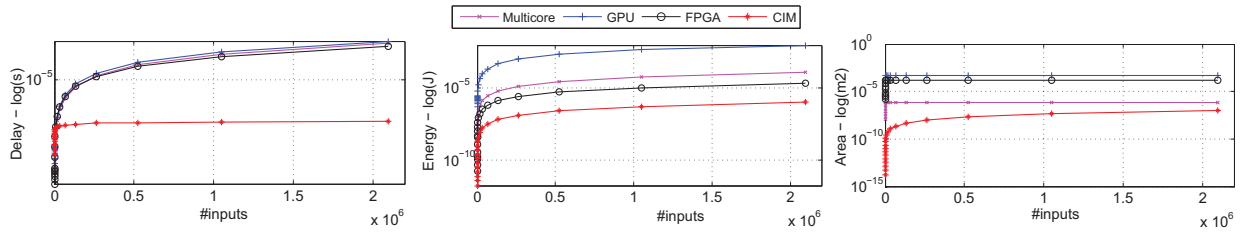


Fig. 5: Multicore vs GPU vs FPGA vs CIM parallel adders with resource constraints (intrinsic parameters)

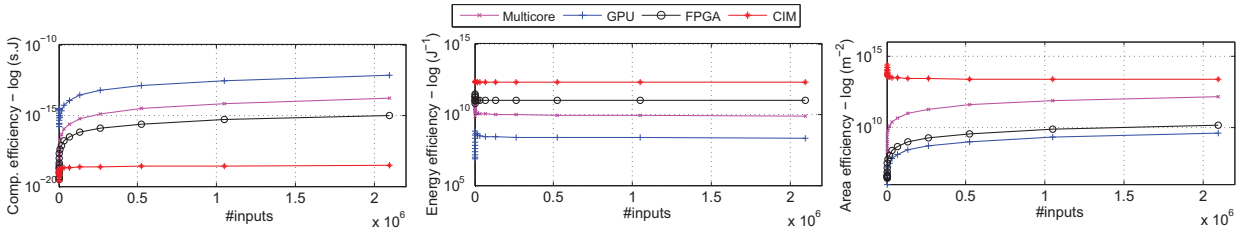


Fig. 6: Multicore vs GPU vs FPGA vs CIM parallel adders with resource constraints (derived parameters)

addition, the FPGA architecture has no cache and data is loaded from registers, while multicore architecture has the highest miss rate in comparison with other architectures with cache. However, memristor has slower switching speed [28] and large number of addition steps [25]. Hence, an addition using CIM takes much more time than FPGA, multicore and GPU architecture. CIM performs nearly four times slower than FPGA architecture while GPU and multicore architecture perform twice faster than CIM architecture.

For energy and area, CIM architecture consumes the least while GPU and multicore architecture consume much more in energy and area (as shown in Fig. 3). The high energy and area consumption was caused mostly by the cache and memory. As unlimited available resources are assumed, area and energy scale up with the input size. For multicore architecture, only a small cache (8kB) was included in each cluster. However, as the number of inputs scales up, the amount of caches in the whole architecture increases. For GPU architecture, the WDS contain 6GB of global memory and 64kB of cache in a default configuration. However, this increased amount of memories does not scale up linearly as in multicore architecture. The default amount of GPU WDS corresponds to 1536 adders. When the number of inputs requires more adders, a scale-up-ratio is applied on the default WDS size. This practically means more GPU platforms have to be used, hence more energy and area are consumed. For FPGA architecture, the same principle applies as if resources are required more than a single platform (default size is taken from Virtex-7 platform) can support, multiple platforms are used. CIM architecture has an advantage of low power consumption, no static power, and small area. The advantage reflects clearly in the results as CIM architecture achieve low energy consumption and area cost. GPU architecture consumes energy linearly with the input size while other architectures' energy scales up 2 to 4 order magnitude less than GPU architecture. GPU and FPGA architecture also consume more area in comparison with multicore and CIM a factor of five and ten order of magnitude, respectively.

Using performance metrics (η_C, η_E and η_A), we observed that although CIM architecture has the highest delay metric in comparison with other architectures (as shown in Fig. 4), it performs the best with respect to the three performance metrics. Note that for computation efficiency, the lower values are better.

B. Limited Resources

Fig. 5 shows the performance of four architectures when resources are assumed to be limited. The maximum resources are dependent on the latest available chip size for each architecture. For multicore architecture, a chip size of $700mm^2$ is used for 22nm technology. For GPU architecture, a chip size of $300mm^2$ is used with 1536 adders. For FPGA architecture, a chip size of $400mm^2$ is used corresponding to Virtex-7 platform. For CIM architecture, no fabrication data is available yet; hence we assume the area constraint for a memristor chip around $0.7mm^2$ (1000 times smaller than the multicore architecture as memristor has advantage of physical size). With these resource constraints, area constantly stays at a particular input size. If the required resources are larger than the provided resources, the architecture has to reuse resources several times to perform the same amount of additions. Hence, the delay increases, which leads to increasing static power consumption of local memories and dynamic power consumption of adders.

Fig. 6 shows that CIM performs better than other architectures in all performance metrics. There is a large gap among CIM and other architectures. For computational performance, CIM architecture performs three order magnitude better than FPGA and seven order magnitude than multicore and GPU architecture. Indeed, CIM architecture achieves even lower delay in the case of limited resources. This gain comes from the advantage of smaller devices and lower energy consumption. Even though, the delay of a single memristor and memristor-based adder are high, an efficient architecture without the WDS (e.g. caches) shows significant performance improvement.

V. Conclusion

In this paper, we have presented a CIM-based parallel adder and estimated its performance. Despite the simplicity of the case study, the results clearly show that CIM architecture has a huge potential and orders of magnitude improvements. This is mainly due to reducing/eliminating memory accesses, using the non-volatile technology, and exploiting the high level of parallelism. CIM architecture seems to be very promising and could enable computation of current infeasible big data applications, fuelling important societal changes.

References

[1] H. Esmaeilzadeh *et al.*, "Dark silicon and the end of multicore scaling," *SIGARCH Comput. Archit. News*, vol. 39, pp. 365–376, 2011.

[2] S. Borkar, "Design challenges of technology scaling," *IEEE MICRO*, vol. 19, pp. 23–29, Jul 1999.

[3] J. W. McPherson, "Reliability trends with advanced CMOS scaling and the implications for design," in *IEEE CICC*, 2007, pp. 405–412.

[4] S. Borkar, "Design perspectives on 22nm CMOS and beyond," in *DAC*. ACM, 2009, p. 9394.

[5] G. Declerck, "A look into the future of nanoelectronics," in *Symposium on VLSI Technology, Digest of Technical Papers*, 2005, pp. 6–10.

[6] G. Gielen *et al.*, "Emerging yield and reliability challenges in nanometer CMOS technologies," in *DATE*. ACM, 2008, p. 13221327.

[7] S. Kaxiras, *Architecture at the End of Moore*, ser. Advances in Atom and Single Molecule Machines. Springer Berlin Heidelberg, 2013, pp. 1–10.

[8] J. W. Janneck, "Computing in the age of parallelism: Challenges and opportunities," Keynote talk, 2013.

[9] A. W. Burks *et al.*, *Preliminary discussion of the logical design of an electronic computing instrument (1946)*. Ablex Publishing Corp., 1989, pp. 39–48.

[10] S. A. McKee, "Reflections on the memory wall," in *CF*. ACM, 2004.

[11] W. H. Kautz, "Cellular logic-in-memory arrays," *IEEE Transactions on Computers*, vol. C-18, pp. 719–727, 1969.

[12] D. G. Elliott *et al.*, "Computational RAM: implementing processors in memory," *IEEE Design Test of Computers*, vol. 16, pp. 32–41, 1999.

[13] P. M. Kogge *et al.*, "Pursuing a petaflop: point designs for 100 TF computers using PIM technologies," in *Symposium on the Frontiers of Massively Parallel Computing*, 1996, pp. 88–97.

[14] D. Keitel-Schulz and N. Wehn, "Embedded DRAM development: Technology, physical design, and application issues," *IEEE Des. Test*, vol. 18, pp. 7–15, 2001.

[15] P. M. Kogge, "EXECUBE-a new architecture for scaleable mpps," in *ICPP*, vol. 1, 1994, pp. 77–84.

[16] D. Patterson *et al.*, "A case for intelligent RAM," *IEEE Micro*, vol. 17, pp. 34–44, 1997.

[17] Y. Kang *et al.*, "FlexRAM: Toward an advanced intelligent memory system," in *ICCD*, pp. 5–14.

[18] J. Draper *et al.*, "The architecture of the diva processing-in-memory chip," in *ICS*. ACM, pp. 14–25.

[19] T. L. Sterling and H. P. Zima, "Gilgamesh: A multithreaded processor-in-memory architecture for petaflops computing," in *Supercomputing Conference*, pp. 48–48.

[20] N. Venkateswaran *et al.*, "Memory in Processor: A novel design paradigm for supercomputing architectures," ser. MEDEA '03. New York, NY, USA: ACM, pp. 19–26.

[21] E. Upchurch *et al.*, "Analysis and modeling of advanced PIM architecture design tradeoffs," in *Innovative Architecture for Future Generation High-Performance Processors and Systems*, 2003, pp. 66–75.

[22] S. Hamdioui *et al.*, "Memristor based Computation-in-Memory architecture for data-intensive applications," in *DATE*, 2015.

[23] L. O. Chua, "The fourth element," *JPROC*, vol. 100, pp. 1920–1927, 2012.

[24] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press, 2000.

[25] A. Siemon *et al.*, "A complementary resistive switch-based crossbar array adder," *IEEE JETCAS*, 2014.

[26] A. A. El-Slehdar *et al.*, "Memristor based N-bits redundant binary adder," *Microelectronics Journal*, vol. 46, pp. 207–213, 2015.

[27] Xilinx, "Xilinx Design tools," 2014.

[28] "The international technology roadmap for semiconductors ITRS," 2011.

[29] C. Meinhardt and R. Reis, "FinFET basic cells evaluation for regular layouts," in *IEEE LASCAS*, 2013, pp. 1–4.

[30] A. Muttreja *et al.*, "CMOS logic design with independent-gate FinFETs," in *ICCD*, 2007, pp. 560–567.

[31] C. T. Antonio *et al.*, "Sub-nanosecond switching of a tantalum oxide memristor," *Nanotechnology*, vol. 22, p. 485203, 2011.

[32] M. Bohr and K. Mistry, "Intels revolutionary 22nm transistor technology," Tech. Rep., 2011.

[33] V. Volkov, "Better performance at lower occupancy," UC Berkeley, Tech. Rep., 2010.

[34] C. Woolley, "GPU optimization fundamentals," Tech. Rep., 2013.

[35] D. Levinthal, "Performance analysis guide for Intel core i7 processor and intel xeon 5500 processors," Tech. Rep., 2008.

[36] E. Karl *et al.*, "A 4.6ghz 162mb SRAM design in 22nm tri-gate CMOS technology with integrated active VMIN-enhancing assist circuitry," in *ISSCC*, 2012, pp. 230–232.

[37] J. Zhao *et al.*, "Optimizing GPU energy efficiency with 3d die-stacking graphics memory and reconfigurable memory interface," *ACM Trans. Archit. Code Optim.*, vol. 10, pp. 1–25, 2013.

[38] L. Chun-Yi and N. K. Jha, "CACTI-FinFET: An integrated delay and power modeling framework for FinFET-based caches under process variations," in *DAC*, pp. 866–871.

[39] X. Yuan, "Modeling, architecture, and applications for emerging memory technologies," *IEEE Design & Test of Computers*, vol. 28, pp. 44–51, 2011.