Storage Class Memory with Computing Row Buffer: A Design Space Exploration

Valentin Egloff, Jean-Philippe Noel, Maha Kooli, Bastien Giraud, Lorenzo Ciampolini, Roman Gauchi, César Fuguet, Éric Guthmuller, Mathieu Moreau[†], Jean-Michel Portal[†]

Univ. Grenoble Alpes, CEA, List
F-38000 Grenoble, France
firstname.lastname@cea.fr

[†]*IM2NP, Univ. Aix-Marseille et Toulon, CNRS, France.*
Marseille, France
firstname.lastname@univ-amu.fr

***Abstract*—Today computing centric von Neumann architectures face strong limitations in the data-intensive context of numerous applications, such as deep learning. One of these limitations corresponds to the well known von Neumann bottleneck. To overcome this bottleneck, the concepts of In-Memory Computing (IMC) and Near-Memory Computing (NMC) have been proposed. IMC solutions based on volatile memories, such as SRAM and DRAM, with nearly infinite endurance, solve only partially the data transfer problem from the Storage Class Memory (SCM). Computing in SCM is extremely limited by the intrinsic poor endurance of the Non-Volatile Memory (NVM) technologies. In this paper, we propose to take the best of both solutions, by introducing a Computing Row Buffer (C-RB), using a Computing SRAM (C-SRAM) model, in place of the standard Row Buffer (RB) in the SCM. The principle is to keep operations on large vectors in the C-RB of the SCM, minimizing data movement to and from the CPU, thus drastically reducing energy consumption of the overall system. To evaluate the proposed architecture, we use an instruction accurate platform based on Intel Pin software. Pin instruments run time binaries in order to get applications' full memory traces of our solution. We achieve energy reduction up to 7.9x on average and up to 45x for the best case and speedup up to 3.8x on average and up to 13x for the best case, and a reduction of write accesses in the SCM up to 18%, compared to SIMD 512-bit architecture.**

***Index Terms*—Near-Memory Computing, In-Memory Computing, von Neumann bottleneck, Storage Class Memory, row-buffer, C-SRAM**

I. Introduction

Conventional computer architectures are based on the traditional von Neumann architecture, where storage and computing are clearly separated. This separation implies an intense traffic between the memory and the Central Processing Unit (CPU) for loading data and storing results. This is dramatically increased by today's data centric applications such as database or deep learning. This phenomenon is known as the von Neumann bottleneck. Moreover, the entire memory hierarchy, from cache memory to storage memory, is designed with different memory technologies which exhibit decades difference in access time. This introduces timing gap in the von Neumann architecture at each levels and leads to the so called memory wall. Furthermore, this data movement can represent up to 80% [1], [2] of the overall total energy in the system. To overcome these latency and energy issues, solutions based on In-Memory Computing (IMC) and/or Near-Memory Computing (NMC) are currently studied. Indeed, such solutions bring computation directly into the memory circuit, avoiding a large part of the data exchange with the CPU. Numerous IMC/NMC solutions have been proposed so far on the different levels of the memory hierarchy, from cache memory with Computing SRAM (C-SRAM) solutions [3], [4], primary memory with Computing DRAM (C-DRAM) [5], [6] and finally to storage class memory (SCM) [7]–[9] with Computing SCM (C-SCM) based on different technologies (Flash, PCM, MRAM, RRAM). All these solutions take advantage of the large memory array organisation to compute operations on very large vectors, close to the Single Instruction Multiple Data (SIMD) computation concept. However, solutions close to the CPU, like C-SRAM, only avoid a small part of the data transfers, since large sets of data are mainly stored in SCM. C-DRAM architectures solve a part of the problem, but DRAM destructive read and high energy consumption limits the benefits of these solutions. Moreover, data transfers between SCM and DRAM are still necessary. Thereby, IMC solutions developed for SCM appear as promising solutions to reduce data transfers. However since endurance of SCM bitcells are limited, C-SCM solutions might strongly reduce their lifespan. To summarize the best of all the proposed solutions:

- IMC solutions should target SCM to avoid large data transfers to the rest of the memory hierarchy.
- IMC solutions should target SRAM technology to fully take advantage of its high endurance and low energy consumption.

Having a close look at the SCM architecture, one can notice that the communication between the SCM and the rest of the hierarchy is performed through a volatile Row-Buffer (RB) [10]. Our key idea is to replace the RB of the SCM by a Computing-RB (C-RB), following a C-SRAM architecture [11], to achieve significant energy reduction and speedup. By doing so, we get the best of both worlds: high data density from SCM and fast and low energy accesses from SRAM.

The main contributions of this paper are:

- Replace the existing row buffer of the SCM with a computing row buffer preventing data movements through the whole memory hierarchy and enabling large vector computation.
- Explore design features and trade-offs of the C-RB (vector width and total memory size).
- Compare our solution to a SIMD 512 bits architecture and

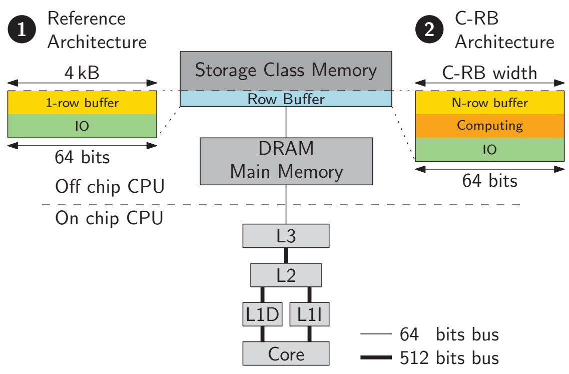


Figure 1: Detailed memory hierarchy. 1 is the reference architecture where the row buffer acts as a temporary storage for read data. 2 is the C-RB architecture where the row buffer is turned into a computing element.

show energy and timing reduction averaged to 7.9× and 3.8× respectively, together with a write access reduction in the SCM up to 18%, on basic benchmarks with linear and quadratic complexity.

The rest of the paper is organized as follow: Section II presents the state of the art of IMC/NMC solutions at circuit and architecture levels with different technologies, Section III details the architecture along with the benchmarks, Section IV describes our simulation platform, Section V presents the results, and finally Section VI concludes the paper.

II. Related Work

IMC concept has been widely studied in the last years since it is a promising solution to overcome the von Neumann bottleneck. Indeed, this concept is highly efficient for vectorizable kernels with massive data. It is also compliant with many memory technologies at every levels of the hierarchy.

Several works focus on computing cache, such as [12], where energy gain is up to 6× and raises up to 7.9× when cache is based on FeFET-RAM. However, when replacing SRAM with FeFET-RAM, the endurance aspect is not discussed. Similarly, [4] transforms the caches into compute units and show performance improvement of 1.9× while observing a reduction of 2.4× in average energy consumption. The IMC SRAM circuit proposed in [3], allowing logic operations and addition, reaches up to 6× speedup compared to a NEON CPU. Reconfigurable architecture using SRAM memories is considered in [13] to meet various requirements either from the application or from the data pattern used during different application phases. The programming model of this reconfigurable architecture is compatible with SIMD programming and show a 60× EDP reduction compared to a SIMD 512 bits architecture. Nevertheless, the considered architecture is a reduced memory hierarchy close to embedded systems. Despite showing impressive results at cache level, these papers do not address the data movements through the memory hierarchy.

Moving computing into the cache hierarchy is the first in-memory computing step. The second step is to compute inside

Table I: Memories configurations

L1 I/D	32 kB, 8-way 64 B block
L2 Unified Cache	256 kB, 4-way 64 B block
L3 Unified Cache	8 MB, 16-way 64 B block
DRAM	512 MB, 64-way 4 kB page
SCM (PCM)	4 GB, 64 B→4 kB RB
C-RB	16 kB→8 MB, 64 B→4 kB block

the DRAM main memory. A system study has been performed in [5] for bitwise operations in DRAM memory. Using triple row activation and a dual-contact cell, AND, OR & NOT operations are directly computed inside the DRAM. The reported performances reached 32× speedup and 35× energy savings on database applications but considering that data are already inside the DRAM and not coming from another external memory. Commercial solutions combining DRAM with a scalar RISC CPU are already available like [6]. Yet, DRAM is one of the biggest energy consumer in data centers [2] and computing inside this technology, although showing great performances, only solve partially the von Neumann bottleneck.

Next and last step is to go on top of the memory hierarchy by computing directly in the SCM, where massive data storage and large vectors are available. 3D NAND Flash memory computing has been developed by [7] to improve deep neural networks power consumption. Indeed, the Flash memory is used to store the constant weights of the network. Computing multiply-and-accumulate operation is performed with an analog approach by summing currents from Flash bitcells. This solution reaches up 40 TOPS/W although the computation result is never written back into the Flash to preserve its endurance and to avoid the energy hungry write operations. Unfortunately, to the best of our knowledge, this solution has never been benchmarked at architecture level, considering the full hierarchy. Using PCM, in [8], a TCAM, which can be considered as a kind of memory computing (pattern matching), reach 500 Msearch/s. Equivalent design can be found with RRAM technology such as [9] where IMC is only used for logic operation. Some works are technology agnostics [15]. This permits to take advantage of the similarities between RRAM, PCM and STT-MRAM. The proposed solution focus only on bitwise operations thanks to large alteration of the internal memory organization. Reported gains are up to 1.12× and 1.11× for speedup and energy saving respectively, but here also, endurance aspect is not discussed. Streaming architecture is proposed in [14] where an application data path is split into several memory processors (MP) designed with NVM technology. Data go from one MP to the next as each MP computes a part of the application. Using convolutional neural network, they achieve up to 235 image/s/W and a speedup of 48× compared to a mobile GPU. However, this solution is close to a FPGA architecture and could face challenges to be introduced in a generic hierarchy.

All these papers show that IMC can be introduced at every level of the memory hierarchy and in a wide range of technology, including emerging NVM. Moreover, all the proposed solutions exhibit gains in terms of performances and energy,

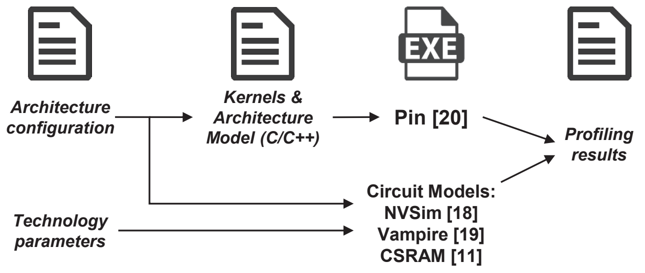


Figure 2: Simulation methodology: main steps

compared to standard von Neumann architecture. But most of IMC solutions based on NVM do not address the endurance limitation of these technologies. Some do not benchmark at architecture level or consider that data are already present in the memory level they work on.

III. Architecture Overview and Benchmarks

A. Architectures and C-RB integration

In this section, we present the HPC architecture and the benchmarks used to assess our solution. We propose to use existing row buffer [10] present in SCM to alleviate access times and repurpose it as a computing element with a privileged wide IO access to the SCM. Our solution and the reference architecture only differ by the SCM RB.

The considered architecture, illustrated in Fig. 1, is composed of an off-chip full memory stack, including a SCM and a DRAM and of an on-chip CPU core with a 3 levels SRAM cache. The main memory configurations are reported in Table I. This architecture is based on a single, in-order core, supporting x86-64 ISA with SIMD capacity of 64B (512 bits). The cache system is a 3 levels cache hierarchy, with a level one cache dedicated to data (L1D) and one dedicated to instruction (L1I), both are 32 kB with 8 ways. The second level (L2) of cache is 256 kB with 4 ways. Finally, the third level of cache (L3) is an inclusive cache of 8 MB with 16 ways. All the levels of cache use 64 B blocks. The buses between every cache level are 64 B (512 bits) wide, so that a cache line can be transferred in one cycle. The main memory is handled as a DRAM cache of 512 MB with 4 kB page size and 64 ways, to reduce memory accesses to the SCM. The buses between the DRAM and cache, as well as between the DRAM and the SCM are 8 B (64 bits). The non-volatile SCM at the top is a PCM of 4 GB. At each access to the SCM, a full 4 kB block is transferred to simplify cache management.

In the reference architecture, the RB is a single block memory matching the IO width of the SCM. It holds the read data from the SCM temporarily while the IO stage sends it over the bus. We replace it with a Computing RB (C-RB), that is a SRAM-based memory with a digital wrapper enabling computing capabilities. When computing, the C-RB loads data from the SCM or can get it from the DRAM and the result is written back to the C-RB memory. This avoids writing back to the SCM and also allows the C-RB to reuse previously computed results without loading from the SCM. When memory capacity of the C-RB is insufficient to contain all data, we remove non-modified data or send modified data to the DRAM. When accessing memory blocks that are shared by both the CPU and the C-RB, data coherency must be ensured. To do that, we invalidate unconditionally blocks in the caches that are written by the C-RB. Conversely, we invalidate blocks in the C-RB that are written by the CPU. Before invalidating from the caches or the C-RB, modified blocks are first sent back to the DRAM which has also the function to merge blocks of different sizes into a 4 kB memory page. This situation must be avoided as much as possible as it incurs a lot of data transfers. Hence, benchmarks should be written in a way that no accesses to the same blocks occurs in an interleaved manner.

Table II: Memories parameters

	Energy	**Timing**	**Leakage**
L1 I/D	Hit: 20 pJ Miss: 24 pJ	3.4 ns	36 mW
L2 Unified Cache	Hit: 52 pJ Miss: 55 pJ	12 ns	261 mW
L3 Unified Cache	Hit: 1.47 nJ Miss: 1.5 nJ	48 ns	815 mW
DRAM [19]	Read*: 39 nJ Write*: 37.5 nJ	Read*: 59 ns Write*: 59 ns	1 W
SCM (PCM) [17] 4 kB block	Read: 308 nJ Write: 1.29 µJ	Read: 1.7 µs Write: 4.67 µs	N/A
C-RB [11] 32 kB 64 B block†	Read:159 pJ Write: 193 pJ	Read: 0.93 ns Write: 0.93 ns	24 mW

* Including Activate & Precharge for a 64 bytes access
† Other size and width are scaled using [13]

B. Benchmarks

To evaluate our proposed architecture (C-RB) versus the reference SIMD 512 bits architecture (REF), four benchmarks are considered: Hamming Weight and shift-or as custom kernels, and two kernels from Polybench [22]: atax and gesummv. These benchmarks are described below:

- *Hamming Weight (hw)* is an algorithm used in information theory to compute the number of symbols that differs in two vectors of same length, with bitwise XOR operation. It is also used in Binary Neural Network as the `popcnt` operation where the second vector is replaced by a null vector. It is a linear algorithm in both memory and computing.
- *Shift-or (so)* is an implementation of the *bitap* algorithm used to match pattern in DNA sequence. It is a linear algorithm in both memory and computing.
- *atax* is a Polybench kernel computing a $A^T(Ax)$ with A a matrix and x a vector. It has a complexity of $\mathcal{O}(n^2)$ in both memory and computing.
- *gesummv* is a Polybench kernel computing a summed matrix-vector multiplications as $\alpha Ax + \beta Bx$ with A, B square matrices, x vector and α, β scalars. It has a complexity of $\mathcal{O}(n^2)$ in both memory and computing.

For all the benchmarks, we used a dataset between 1 GB and 2 GB thus with an amount of data superior to the DRAM capacity to force transfers between DRAM and SCM, which is representative of today's application's dataset size.

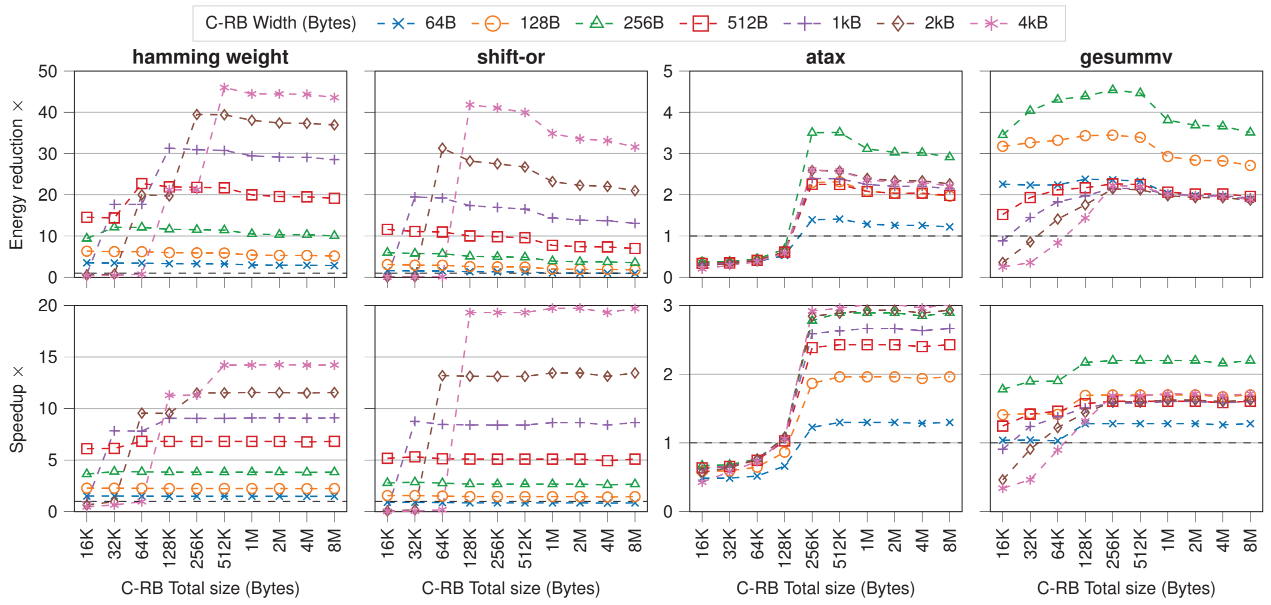


Figure 3: Energy reduction and speedup for all benchmarks compared to the reference architecture according to the total C-RB memory size and vector width (higher is better).

IV. Simulation Platform

This section is dedicated to our simulation platform description, through workflow and used tools, as represented in Fig. 2. First, the reference and the proposed architecture are modeled in C++ and provided as input to Intel Pin software [20]. This model, also called a pintool, implements the two architectures with coherency management for our solution. In addition it collects all memory accesses and statistics. Pin allows us to instrument applications at instruction level, as such, our platform is instruction accurate. When the application ends, the data are written back to the SCM. Two versions of the benchmarks are compiled and executed: one for the reference architecture using AVX-512 intrinsics; one with the C-RB instructions using a dedicated Instruction Set Architecture (ISA), proposed in [21] and extended to 64-bit architecture. In order for the C-RB to perform operations, these dedicated instructions are encoded and emitted by the CPU. To execute a remote instruction, the CPU must store a 64-bit word to a given address region. Concatenation of the address and data fields serves as a representation of the instruction opcode and the operands' addresses. These instructions are prepared on the fly by the CPU and can be stored on the stack to be reused later. This saves a lot of preparation time and instructions, especially for loops iterating over an array where instructions are always the same and only one or more operands' addresses change. Address and data fields are then decoded by the remote compute unit. Both versions, AVX-512 and C-RB, are functionally similar and written using C-style macros that encapsulates vector instructions.

The energy and timing parameters of all memory levels are extracted from either circuit model tools or publications from the state of the art. Estimation of the energy and timing cost of all the different cache levels are performed using NVSim [18] for a 22nm node with the architecture configuration given in Table I. L1 caches are optimized for latency while L3 is optimized for area. DRAM parameters are taken from [19]. SCM's bitcell parameters, named Technology parameters in Fig. 2, such as set and reset pulses width and currents as well as read voltage, are extracted from [17] and fed to NVSim. Finally, C-RB numbers, considering a C-SRAM design approach, are taken from [11] and adapted to the different width and memory capacity using the tiling model from [16]. Obtained memories parameters are given in Table II.

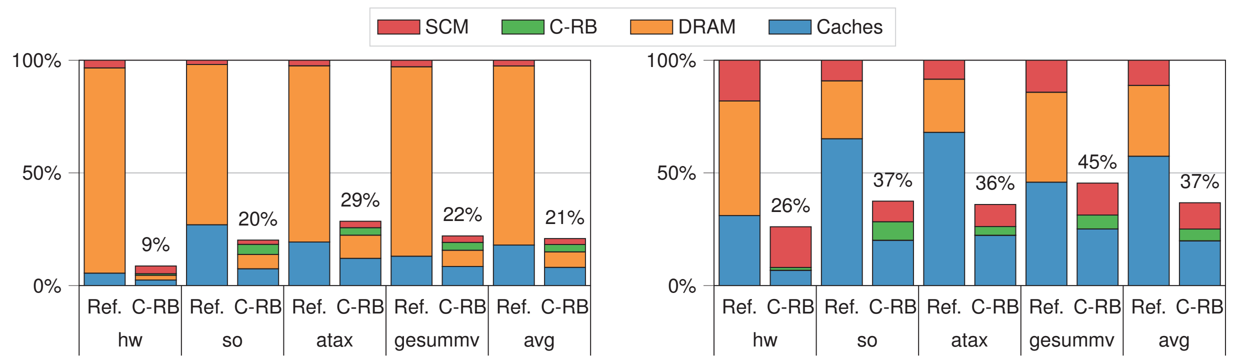
During a benchmark execution with Pin, all the memories accesses are extracted together with their types (Read/Write for all memories plus hit/miss for caches). These accesses are then associated with their energy and timing parameters corresponding to the acceded memory, after which they are summed to obtain the global contribution of each memory in the hierarchy for the complete execution of a given benchmark.

V. Simulation Results

A. C-RB design space exploration

We vary the total memory size and the width of the C-RB to explore different configurations. The total memory size range from 16 kB which corresponds to a small cache, to 8 MB, the size of the level 3 cache. The width of the row buffer always matches the IO width of the SCM and ranges from 64 bytes, the width of cache and vector extension in the CPU, to 4 kB which is the usual size of a memory page in a modern system. The exploration results are given in Fig. 3 with respectively the energy reduction and the speedup obtained with our solution versus the reference system.

First of all, regarding the C-RB total memory size, it clearly appears that an optimum in term of energy reduction and



(a) Energy repartition

(b) Timing repartition

Figure 4: Energy and timing repartition in the memory hierarchy with a 256 kB C-RB and 256 B vector width normalized to the SIMD 512-bit architecture (lower C-RB is better)

speedup can be found between 128 kB and 512 kB depending on the considered benchmark. The optimum is found at 512 kB for *hamming weight*, 128 kB for *shift-or* and 256 kB for *atax* and *gesummv*. After this optimum, the energy reduction slightly decreases due to more important leakage and access cost, whereas speedup gain remains almost constant. This means that the benchmarks do not require more memory for working without having too many non mandatory misses. Thus C-RB total memory size has to be limited to this range to fit the benchmark requirements in term of temporary storage for intermediate results and constants data that are used throughout all the execution.

Secondly, regarding the C-RB width, the complexity of the benchmark plays a major role. Considering the linear benchmark, namely *hamming weight* and *shift-or*, the largest width (4 kB) leads to the best results in energy reduction and in speedup. This trend can be explained by the nature of the benchmark, where data have no dependency and the data access pattern is a streaming one. Thus large C-RB vector width leads to less memory accesses, which in turn reduces energy and execution time. Now, considering quadratic benchmarks, a clear optimum is obtained with a C-RB width of 256 B. Both quadratic benchmarks are based on a matrix-vector product, with a reduction through an addition operation. This reduction operation must be applied along the entire vector width, thus limiting the benefits of large buffer ($>$ 256 B). This last point also explains that the energy reduction and, respectively, the speedup are one decade above for the linear benchmarks compared to the quadratic ones.

Overall, the proposed design space exploration gives some trends on C-RB total memory size, which should be comprised between 128 kB and 512 kB and on the C-RB width, which should be set to 256 B to be compliant with the quadratic benchmark requirements.

B. C-RB impact on memory hierarchy usage

The normalized energy and timing distribution of the whole memory hierarchy is reported Fig. 4 for the four benchmarks and the average case, noted avg. The selected configuration for the C-RB is compliant with the trends drawn in the previous subsection, with a C-RB total memory size of 256 kB and a vector width of 256 B.

In Fig. 4a, for the REF architecture, the average energy consumption is dominated by the DRAM (79%) followed by caches (18%) and the remaining 3% are due to the SCM. These numbers are in-line with the standard trends of the state of the art. This gives us a great deal of leeway to tap into this energy, so we can easily get considerable energy reduction as shown in Fig. 4a. Indeed, for the C-RB architecture, most of the computation is located into the row-buffer of the SCM, limiting data access to the rest of the hierarchy, and leading to only 21% of energy consumption in the C-RB architecture compared to the REF architecture, in average. In the C-RB architecture, energy consumption is well balanced between DRAM and caches. The DRAM consumption is mainly due to idle consumption (refresh process and leakage). The caches consumption is reduced thanks to the reduction of the number of instructions, with for example, 75% less instructions for *shift-or* and 40% less for *atax*. But this reduction is balanced by the increase of caches usage ($+25\%$) with the storage of the C-RB instructions on the stack.

In Fig. 4b, for the REF architecture, the average timing is dominated by the caches, with 57%, followed by the DRAM (31%) and the SCM (12%). Here also, the introduction of the C-RB solution reduces considerably the timing, down to 37% of the REF architecture. Since the majority of the computation on large vectors is supported by the C-RB, the DRAM is only very sparsely used and acts as a write buffer for the C-RB. The introduction of the C-RB reduces greatly the number of executed instructions, which in turn reduces caches' timing, mainly L1I for the instruction and L3 for the data.

Focusing on the C-RB solution and considering energy and timing values normalized versus the REF architecture, one can notice that:

- the energy distribution is well balanced between the caches and the DRAM, with respectively 8% and 7% followed by the C-RB (3%) and the SCM (3%).
- the timing distribution is mainly shared between the

caches (20%), the SCM (12%), the C-RB (5%) and the DRAM (< 1%).

Comparing these values to the REF architecture, it appears clearly that the introduction of the C-RB has a great impact on the data transfers in the hierarchy, with a clear reduction of the activity of the DRAM (energy is reduced from 79% down to 8%, and timing from 31% to less than 1%) and of the caches (energy is reduced from 18% to 7%, and timing from 57% to 20%), whereas the SCM activity remains stable.

C. C-RB impact on SCM endurance

As mentioned in the previous sub-section, SCM activity remains stable, but it is interesting to focus more in detail on the operations performed on the SCM, since only write operations impact the endurance. In this aim, the numbers of write and read accesses, normalized versus the REF architecture, are presented in Fig. 5 for our case study (C-RB total memory size of 256 kB and vector width of 256 B).

For the linear benchmarks, the same number of accesses is found for the REF architecture and our solution. As these benchmarks have streaming access patterns, which reduce large vectors to scalars, the written blocks stay in the cache in the REF architecture and cannot be evinced before the end of the applications. In the C-RB architecture, the same written blocks stay in the C-RB as they are written at every loop iteration. Thus, only the read data can be removed from both working memory (L1D & C-RB), but as it is only read once, the number of read accesses does not increase. For the quadratic benchmarks, the number of write accesses decrease with our solution. Indeed, in the REF architecture, both read-only and written data compete for available blocks leading to DRAM contention. This is due to the fact, that written data does not fit in the caches and some blocks get evicted before being reused. This phenomenon does not happen in the C-RB architecture, where only the written data from C-RB target DRAM. For the same reason, more reads ensue as there is limited space in the C-RB memory, especially for *atax* where the input matrix is used twice and cannot fit in the C-RB.

To summarize, our solution preserves the endurance of the SCM with linear benchmarks and reduces the number of write accesses for quadratic benchmarks, with 18% less access for *atax* and 10% for *gesummv*.

VI. Conclusion

Today's modern big-data applications such as deep learning are limited by memory bandwidth and memory energy cost. We propose to replace the existing row buffer within an SCM (PCM) with a tightly coupled computing row buffer, based on SRAM technology with a digital wrapper. We achieve energy reduction of 7.9× on average and up to 45× for the best case and speedup of 3.8× on average and up to 13× for the best case. We also show that our solution preserves and even enhances the SCM endurance, with an endurance saving up to 18%. Moving computation closer to the data storage can thus leverage huge benefits and be a step towards tackling the memory wall. This design space exploration can be extended to any SCM based on classical or emerging NVM technology. Future works include exploration of other computation locations within the memory hierarchy; different data management and parallel work between this remote computing unit and the CPU; comparison with other architectures such as GPU and extension to more complex benchmarks.

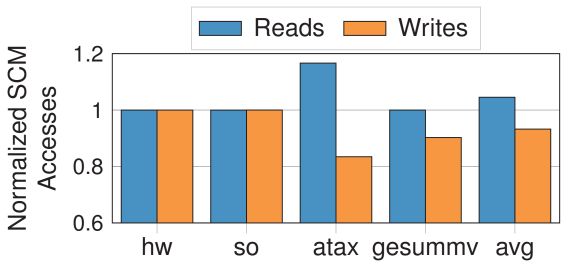


Figure 5: SCM accesses with a 256 kB C-RB and 256 B vector width normalized to the SIMD 512-bit architecture

References

[1] M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," in ISSCC 2014

[2] T. Vogelsang, "Understanding the Energy Consumption of Dynamic Random Access Memories," in MICRO-43, 2010

[3] William Simon et al., "BLADE: A BitLine Accelerator for Devices on the Edge," in GLSVLSI, 2019

[4] S. Aga et al., "Compute Caches", in HPCA 2017

[5] Vivek Seshadri et al., "Ambit: In-memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in MICRO-50, 2017

[6] "UPMEM", 2020. http://www.upmem.com/

[7] H.-T. Lue et al. "Introduction of Non-Volatile Computing In Memory (nvCIM) by 3D NAND Flash for Inference Accelerator of Deep Neural Network (DNN) and the Read Disturb Reliability Evaluation : (Invited Paper)," in 2020 IRPS

[8] J. Li et al., "1 Mb 0.41 µm² 2T-2R Cell Nonvolatile TCAM With Two-Bit Encoding and Clocked Self-Referenced Sensing," in JSSC, Apr. 2014

[9] W. Chen et al., "A 16Mb dual-mode ReRAM macro with sub-14ns computing-in-memory and memory functions enabled by self-write termination scheme," in IEDM 2017

[10] J. Park et al., "Design space exploration of row buffer architecture for phase change memory with LPDDR2-NVM interface", in VLSI-SoC 2015

[11] J.-P. Noel et al., "Computational SRAM Design Automation using Pushed-Rule Bitcells for Energy-Efficient Vector Processing", in DATE 2020

[12] D. Gao et al., "Eva-CiM: A System-Level Performance and Energy Evaluation Framework for Computing-in-Memory Architectures," in TCADICS 2020

[13] R. Gauchi et al., "Reconfigurable tiles of computing-in-memory SRAM architecture for scalable vectorization," in ISLPED 2020

[14] L. Chang et al., "DASM: Data-Streaming-Based Computing in Non-volatile Memory Architecture for Embedded System", in VLSI-S, 2019

[15] Shuangchen Li et al., "Pinatubo: A Processing-in-memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories", in DAC 2016

[16] R. Gauchi et al., "Memory Sizing of a Scalable SRAM In-Memory Computing Tile Based Architecture", in VLSI-SoC 2019

[17] Y. Choi et al., "A 20nm 1.8V 8Gb PRAM with 40MB/s program bandwidth," in ISSCC 2012

[18] X. Dong et al., "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory," in TCADICS 2012

[19] S. Ghose et al., "What Your DRAM Power Models Are Not Telling You: Lessons from a Detailed Experimental Study," Proc. ACM Meas. Anal. Comput. Syst., Dec. 2018

[20] C.-K. Luk et al., "Pin: building customized program analysis tools with dynamic instrumentation," SIGPLAN Not., Jun. 2005

[21] M. Kooli et al., "Smart instruction codes for in-memory computing architectures compatible with standard SRAM interfaces," in DATE 2018

[22] Pouchet, L. N. (2012). Polybench: The polyhedral benchmark suite. http://www.cs.ucla.edu/pouchet/software/polybench