Reconfigurable tiles of computing-in-memory SRAM architecture for scalable vectorization

Roman Gauchi, Valentin Egloff, Maha Kooli, Jean-Philippe Noel, Bastien Giraud, Pascal Vivet, Subhasish Mitra, Henri-Pierre Charles

Reconfigurable Tiles of Computing-In-Memory SRAM Architecture for Scalable Vectorization

R. Gauchi[1], V. Egloff[1] M. Kooli[1], J.-P. Noel[1], B. Giraud[1], P. Vivet[1], S. Mitra[2] and H.-P. Charles[1]
[1]Université Grenoble Alpes, CEA List, Grenoble, France [2]Stanford University, Palo Alto, CA, USA
{*roman.gauchi, valentin.egloff, maha.kooli, jean-philippe.noel, pascal.vivet, henri-pierre.charles*}*@cea.fr, subh@stanford.edu*

ABSTRACT

For big data applications, bringing computation to the memory is expected to reduce drastically data transfers, which can be done using recent concepts of Computing-In-Memory (CIM). To address kernels with larger memory data sets, we propose a reconfigurable tile-based architecture composed of Computational-SRAM (C-SRAM) tiles, each enabling arithmetic and logic operations within the memory. The proposed horizontal scalability and vertical data communication are combined to select the optimal vector width for maximum performance. These schemes allow to use vector-based kernels available on existing SIMD engines onto the targeted CIM architecture. For architecture exploration, we propose an instruction-accurate simulation platform using SystemC/TLM to quantify performance and energy of various kernels. For detailed performance evaluation, the platform is calibrated with data extracted from the Place&Route C-SRAM circuit, designed in 22nm FDSOI technology. Compared to 512-bit SIMD architecture, the proposed CIM architecture achieves an EDP reduction up to 60× and 34× for memory bound kernels and for compute bound kernels, respectively.

KEYWORDS

Computing-In-Memory, SRAM, Instruction Set Simulator, SystemC/TLM, SIMD, PolyBench.

1 INTRODUCTION

As artificial intelligence progresses, today's applications are becoming data and computationally intensive. The cost of transferring data from the memory to the processing element is very high compared to the cost of the computation itself [11]. New emerging architectures must be considered to reverse this paradigm by bringing the computation closer to the memory. In-Memory Computing (IMC) has been introduced in [3, 4, 12] and aims at bringing the computation as close as possible to the memory, by directly modifying the SRAM bit-cell design. These IMC designs can perform logic operations (mainly bit-wise) within the memory by selecting multiple word-line of the memory bit-cell array. In order to provide more complex logic and arithmetic operations, the Near-Memory Computing (NMC) provides the necessary additional operators inside the SRAM memory tile, as presented in [7, 12]. From a system integration and software-level, IMC and NMC can be named as Computing-In-Memory (CIM), since all computations are performed within the memory interface. Thus, we refer IMC and NMC as CIM in the rest of this paper and we consider the Computational-SRAM (C-SRAM) concept as introduced in [4]. It represents a SRAM tile integrating additional logic allowing arithmetic and logic operations. In all these schemes, data locality is preserved and acceleration is performed through parallel computing, using vector operations, where vector width corresponding to SRAM width. Recent work on SRAM-based CIM has focused on computing within a single or a few memory tiles, which drastically limits the memory capacity and potential applications. In order to address kernels with larger dataset and real applications, it is required to scale up the architecture with more memory, while proposing the adequate programming model allowing vector acceleration. With more available memory, a question raises: how to organize these tiles of memory to perform large vector operations : more vectors or larger vectors ?

In order to increase the available memory space and associated computing capability, we propose to assemble a set of C-SRAM tiles in a configurable fashion. This allows to extend computing vector in two manners: either a horizontal memory extension allowing larger vectors, or a vertical memory extension allowing more vectors. This scalable vectorization scheme allows to directly re-use existing SIMD (Single Instruction Multiple Data) application kernels onto the targeted C-SRAM architecture. In order to evaluate this architecture, we propose an instruction accurate simulation platform that considers all transaction events between hardware components and its integration into a standard software tool chain. For accurate architecture exploration, all low level performance numbers (cycles and energies) are extracted from Place&Route implementation in 22nm FDSOI technology, as presented in [17]. Using the simulation platform, we evaluate the performance speed up, the energy reduction and the Energy Delay Product (EDP) reduction of the reconfigurable vector-based C-SRAM tiles architecture compared to tightly-coupled 128-bit, 256-bit and 512-bit SIMD architectures, using the same vectorized kernels. Simulation results show that our proposed architecture achieves an EDP reduction up to 60× and 34× for memory bound kernels and for compute bound kernels respectively, compared to 512-bit SIMD architecture. The main contributions of this paper are:

- Proposing a reconfigurable multi-tiles of C-SRAM architecture for scalable vectorization,
- Proposing an instruction accurate and power annotated simulation platform for multi-tile C-SRAM evaluation,
- Comparing performance speed up, memory accesses, energy reduction and energy-delay product of our proposed architecture with regards to modern SIMD architectures.

The remainder of this paper is organized as follows: Section 2 discusses the state-of-the-art. Section 3 introduces the proposed architecture, Section 4 details the simulation platform, Section 5 explains architecture exploration results, and finally, Section 6 concludes the paper.

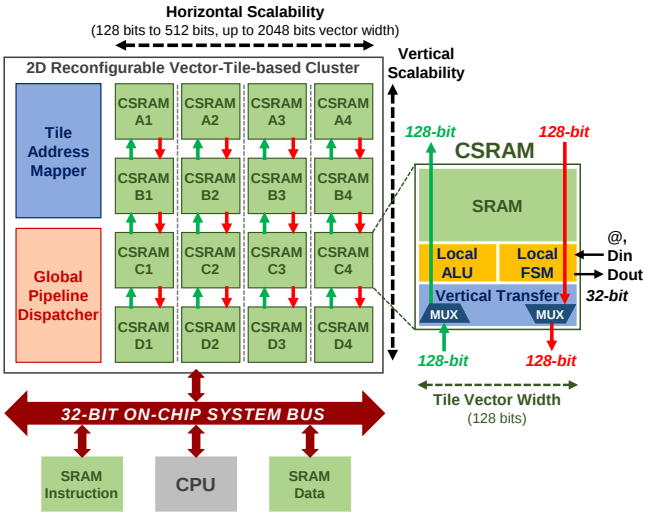

Figure 1: Reconfigurable Tiles of C-SRAM Architecture, integrated with a CPU on a standard system bus.

2 RELATED WORK

2.1 In and Near Memory Computing

IMC and NMC concepts have been introduced recently and propose different kind of data-centric architectures. From a circuit design point of view, many recent works [3, 4, 6, 12] have shown that IMC provides logic operators using pre-computation capabilities within the memory array. Recent works [10] also explored IMC using emerging technologies such as Non Volatile Memory (NVM) to perform Multiply And Accumulate (MAC) operations by exploiting analog current laws. Such circuits targets neural network applications, while our architecture uses existing current CMOS technology SRAM memories to target general purpose applications, specifically with vector level parallelism. Compared to IMC, NMC consists of an optimized design placed close to the memory which performs operation between several memory tiles. But, this "near memory logic" is usually not tightly integrated within the memory, nor respecting a standard memory interface for smooth software integration, as presented in [21]. A specific IMC type, using bit serial scheme has shown up 1.9× speed up gain and 2.4× energy reduction, but requires a high data parallelism (all words active) to achieve these gains, as presented in [7, 12]. For further system integration, IMC feature has been integrated within cache architecture as proposed in [20], nevertheless, the programming model of IMC through the cache is not explicit and large vector operations are not supported. For ease of software integration, specific Instruction Set Architecture (ISA) for IMC has been proposed and evaluated at functional-level on a single IMC memory instance in [14]. For larger dataset, integration of multiple tiles of IMC or NMC has not been explored in a systematic way in terms of memory organization and performance trade-offs. In this paper, we propose a reconfigurable organization of CIM memory tiles, for vector based parallelism acceleration using a standard programming model.

2.2 Reconfigurable Architecture

For reconfigurable computing, FPGAs are composed of tiles of logic, using Look-Up-Table (LUT) and tiles of SRAM, called Block RAM (BRAM). FPGAs are designed for emulating any logic and memory functions while the proposed reconfigurable C-SRAM tiles are made of rather similar elements, but organized to act as an energy efficient vector accelerator. In past years, many vector processors and associated configurable architectures have been proposed [22], but they still rely on a register semantic and their vector width is limited to their local SRAM data bus. The proposed reconfigurable C-SRAM tiles architecture avoids register transfers by maintaining data within the local SRAMs and the vector width can be configured to the maximum number of available tiles.

3 RECONFIGURABLE CIM ARCHITECTURE

3.1 Architecture Overview

As shown in Figure 1, the C-SRAM cluster consists of multiple tiles of C-SRAM disposed in a physical array. Each C-SRAM tile is composed of : the SRAM itself, a *Local ALU* (Arithmetic Logic Unit) for near-memory operation, a *Local FSM* (Finite State Machine) for pipeline control and finally a *Vertical Transfer* unit. For application mapping facility, the physical C-SRAM array can be logically reconfigured in two manners : either for horizontal vector extension to create larger vectors using the *Tile Address Mapper* unit ; or for vertical vector extension to create more vectors using the vertical connections and the *Vertical Transfer* unit. Each C-SRAM tile stores and computes data on a 128-bit vector width in one instruction. Using the horizontal scalability, up to 2048-bit vectors can be handled in a single cycle. We detail these mechanisms in section 3.2.

For overall control at cluster level, the total vector width is configured by the *Tile Address Mapper* unit, while the *Global Pipeline Dispatcher* unit resolves data hazards and conflicts between interleaved SRAM accesses and CIM instructions. Every CIM instruction passes through a 5-stage pipeline made up of (1) Decode, (2) Read left operand, (3) Read right operand, (4) Execute the operation and (5) Write back into the memory. This C-SRAM cluster array is seen as an co-processor unit, and is integrated onto a standard system bus. It is controlled by a CPU (Central Processing Unit) engine with its tightly coupled data and instruction memories.

3.2 Inter-tiles Reconfiguration and Vertical Communication

For data-centric kernel, such as matrix multiplication shown in Listing 1, reduction operations (perform the same operation between the elements composing the vector) can be costly when computing on large vectors. The proposed architecture can dynamically resize the vector width during execution for each instruction. Actually, as shown in Figure 2, the kernel defines logical vector width used as in the layout ❸, then the next instruction uses sub-vectors as in the layout ❷. After performing an addition between these sub-vectors, the last instruction splits the vector as in ❶, to complete the operation reduction in a 512-bit vector. To maintain instruction throughput, the *Tile Address Mapper* ensures the vertical data movement between tiles and the *Global Pipeline Dispatcher* orders instruction to avoid pipeline data hazards.

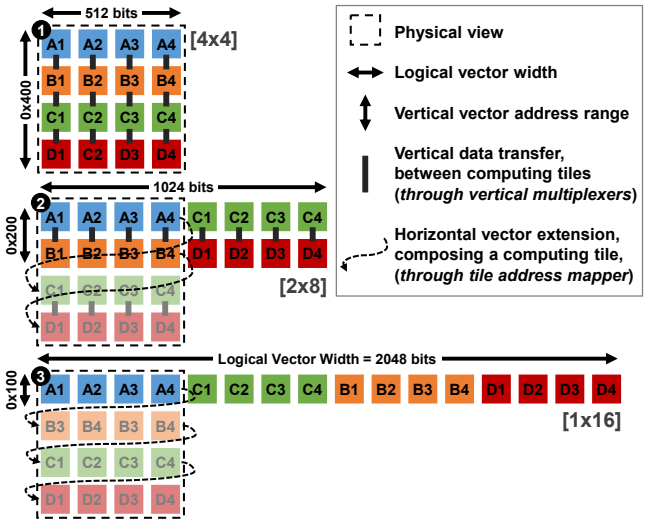

Figure 2: Physical and logical view for 3 vector configurations: ❶ 512-bit, ❷ 1024-bit and ❸ 2048-bit wide.

Tile Address Mapper – Assembling these C-SRAM tiles horizontally permits to realize vectorized computation by distributing the same CIM instruction to all the tiles. Moreover, the *Tile Address Mapper* allows 32-bit sequential access to all C-SRAM in a horizontal way. In Figure 2, the vector size is scalable from 512 bits up to 2048 bits with a 128-bit step, thus offering an address range that extends from 1024 vectors of 512 bits to 256 vectors of 2048 bits. Thanks to this vector approach, computations are performed in an aligned and vertical way. By using a memory partitioning into smaller tiles, the *Tile Address Mapper* allows to scale the architecture with some limits: energy cost of individual accesses is inversely proportional to the read access time. In terms of physical design, the *Tile Address Mapper* provides an energy/performance trade-off as long as the number of tiles is limited [9].

Vertical Transfer Multiplexer – For data transfer between tiles, two additional vertical interconnections of 128 bits width each redirect a large vector of data respectively in the upper or lower direction. For example, in tiling configuration ❷, a vector that moves from tiles A to tiles B implies that the 1024-bit data will be transferred through rows of multiplexer via the down path of eight C-SRAM tiles. The vertical transfer is done in the second pipeline stage, within the read cycle, before the computation cycle.

Data placement and off-chip memory – To read or write a 32-bit data from the CPU, the *Tile Address Mapper* selects the aligned C-SRAM according to various layout set-up, as presented in Figure 2. At software-level, the *Tile Address Mapper* can stride addresses in memory to optimize internal data transfer which impacts the vectorization performance. In the proposed architecture, all memories (SRAM and C-SRAM) are on-chip memories in order to evaluate kernel acceleration. Nevertheless, data movement from off-chip memories should also be considered to evaluate larger datasets.

3.3 ISA and Programming Model

To use an abstract model from the user's point of view, a dedicated ISA for the IMC architecture is detailed in [14]. In this paper, we demonstrate the extensibility of this ISA with a multi-tile implementation, without changing the design of the CPU and couple our multi-tile architecture on a standard 32-bit system bus. From the execution point a view, sending a compute instruction to the C-SRAM is equivalent to writing a specific data to a specific address in a memory (`store` instruction). From the software-level point of view, we implement our C-SRAM ISA as a library written in C/C++, as written in Listing 2, that can be included in any C/C++ compiler tool chain (clang, gcc, riscv-gcc). In addition, we define a new vector format type (`CSRAM_Vect_t`) which ensures the data alignment inside the targeted memory area, as proposed in Listing 1. This method is used for both SIMD and CIM software evaluation. Thus, macro definitions as in Listing 1 and inlined functions allow to be SIMD-compatible whatever the vector size is. Since the same level of vector parallelism is targeted, the proposed programming model permits to reuse directly low level vectorized kernels already written for SIMD architecture for our proposed CIM architecture. This avoids the tedious work of application mapping and parallelism extraction.

Listing 1: Example of Matrix Multiply kernel in language C.

```
1  /* Typedef used for C-SRAM scalable vector size */
2  typedef CSRAM_Vect_t Mat[N][N / VECTOR_SIZE];
3  /* Memory map define with sections */
4  __attribute__((section(".csram"))) Mat A, B, C;
5  /* Exemple of NxN Matrix Multiply for SIMD & C-SRAM */
6  for(int i = 0; i < N; i++)
7      for(int j = 0; j < N; j++) {
8          int sum = 0;
9          for(int k = 0; k < N / VECTOR_SIZE; k++) {
10             vmul8(tmp[k].v, A[i][k].v, B[j][k].v);
11             vreduce_add8(sum, tmp[k].v);
12         }
13         C[i][j/VECTOR_SIZE].i8[j%VECTOR_SIZE] = sum;
14     }
```

Listing 2: 8-bit chunk vector multiply macro applied per tiles for a configured vector size, up to 2048 bits.

```
1  #define vmul8(D, A, B) do {                                  \
2    volatile uintptr_t* cm_addr = MAKE_ADDR(OPC_MUL8, D); \
3    uintptr_t           cm_data = MAKE_DATA(A, B);         \
4    *cm_addr = cm_data;       /* <=> store instruction */ \
5    asm("" ::: "memory");     /* memory fence */          \
6  } while(0)
```

4 SIMULATION PLATFORM

Previous works on IMC have used Low Level Virtual Machine (LLVM) as a high level compiler tool chain for high level performance profiling of the IMC [13]. Nevertheless, it lacks detailed modeling of instruction control flow and memory sizing with regards to application needs. In order to provide a fast and flexible modeling platform of the architecture, we propose in this paper a simulation platform using SystemC Transaction-Level Modeling (TLM) abstraction [1]. In order to perform an accurate exploration at system-level, we extract all performance and energy values from Place&Route design experiments using Global Foundries (GF) 22nm FDSOI technology node.

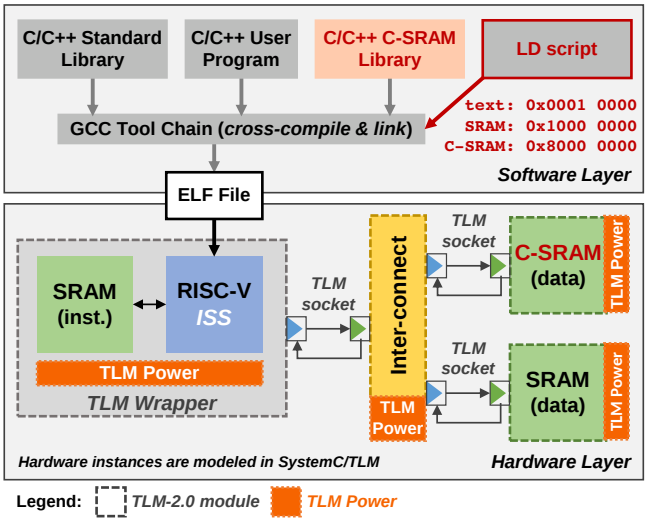

Figure 3: Hardware/Software Simulation Platform.

4.1 Simulation Platform Overview

The proposed simulation platform is composed of both hardware and software related layers, as shown in Figure 3. On the top, the software layer relies on the C/C++ C-SRAM library, a Linker Descriptor (LD) script and the user program in a GCC tool chain to create an Executable and Linkable Format (ELF) file. On the bottom, the hardware layer simulates the architecture behaviour and memory transactions between hardware components, as presented in Section 3.

Software Layer – To run our target applications on our platform, we use the same riscv-gcc compiler tool chain as the actual hardware. The C-SRAM ISA library consists of macro definitions and inlined functions to emulate a vector format such as SIMD, but expandable up to 8192 bits wide. The LD script file define the memory map by specifying the program (`text`), the data (`SRAM`) and the CIM (`C-SRAM`) sections.

Hardware Layer – For RISC-V modeling, we use an open source instruction-accurate Instruction Set Simulator (ISS) developed by Western Digital [2]. This ISS is wrapped in SystemC/TLM platform and execute the binary ELF file given by the compiler. By using SystemC/TLM, we model all other hardware components with cycle accuracy, including the C-SRAM, the *Global Pipeline Dispatcher*, the *Tile Adress Mapper*, and all other sub-system accesses and associated latencies. For power modeling, we inherit the TLM Power library [15], allowing to annotate power values of all elements, including idle state and leakage values.

4.2 Simulation Platform Calibration

Regarding the physical scalability of the multi-tile memory architecture, we have evaluated the wiring cost and the correct trade-off between C-SRAM tile size and tile performance. As presented in [9], a wiring cost and energy model based on Place&Route shows the scalability of multiple SRAM tiles: for a 256 kB of total memory size, composing an 4×16 array of 4 kB tile, the wiring cost between tiles is about 50% in read access time, while partitioning in array allows to save massive dynamic power. The C-SRAM architecture

Table 1: Energy details in 22nm FDSOI used for simulations.

Module name	Average of extracted energy values (pJ)			
	Core		Memory	
	Compute(s)	Fetch	Read	Write
Inter-connect (on-chip)	–	1.63	1.63	1.63
RISC-V core	[2.24, 4.83]	1.92	–	–
SRAM (4kB, 32-bit access)	–	–	4.20	4.20
128-bit C-SRAM (4kB, 32-bit access)	[3.41, 7.21]	–	4.32	6.48
128-bit SIMD (128-bit access)	[8.96, 19.32]	1.92	16.80*	16.80*
256-bit SIMD (256-bit access)	[17.92, 38.64]	1.92	33.60*	33.60*
512-bit SIMD (512-bit access)	[35.84, 77.28]	1.92	67.20*	67.20*

*SRAM memory access in one instruction. *Notes:* RISC-V numbers are adapted from [5] in GF 22nm, C-SRAM numbers are from Place&Route results under GF 22nm in [17] and SIMD instruction numbers are adapted according to the bus width as explained in Section 5.2.

integrates both IMC operations (bitwise logic) and NMC operations (*ADD*, *MULT*). It has been designed in a GF 22nm FDSOI technology and implemented with different floor-plans configurations up to final Place&Route [17]. We have selected a 4 kB SRAM configuration: the C-SRAM single tile performance numbers (timing and energy access) are extracted for the multi-tile architecture exploration. In this paper, we ensure that all evaluated architectures operate at the same frequency and each timings (for memories and core) are constrained using the same parameters, in order to extract energy values summarized in the Table 1.

As a summary of all related design performances, each row of Table 1 represents a hardware component converted into TLM module that have different energy states (*Compute*, *Fetch*, *Read*, *Write*). For instance, all instructions perform by the RISC-V core (*Compute*) are fetched (*Fetch*) from the *SRAM (instruction)* memory and all CIM instructions are executed in each C-SRAM (*Compute*). These energy values are given according to the instruction types.

5 ARCHITECTURE BENCHMARKING

Using the proposed simulation platform, we study five kernels representing different application profiles in terms of memory patterns and computing requirement. All our application evaluations use the same vectorized code using gcc (7.4.0) and riscv-gcc (7.2.0) compilers at optimization level 3, forcing SIMD vectorization by using intrinsics on Intel's Xeon processor.

5.1 Kernels with Scalable Vectorization

Hamming Weight (**`hw`**) [8] is a kernel used in information theory applications to count the number of ones after performing a *XOR* operation between two vector of bits (using *AND*, *SRL*, *ADD*, *SUB* operators). Shift-OR (**`so`**) [8], is a kernel used in bio-informatic applications to match a pattern in a DNA sequence. It prepares a set of bit-masks saving each position of the pattern element and shifts those bit-masks through the DNA sequence with bitwise operators (*OR*, *AND*, *SRL*, *SLL*) to match the position. PolyBench [18] is a collection of benchmarks used for evaluating polyhedral problems and improve auto-parallelization of compilers. By using 8-bit fixed-point addition and multiplication instructions, we assess three Basic Linear Algebra Sub-programs as: **`atax`**, a matrix transpose and vector multiplication kernel as $A^T.A_x$, **`gemm`**, a matrix-multiply as $C = \alpha.A.B + \beta.C$ kernel and **`3mm`**, three matrix-multiply

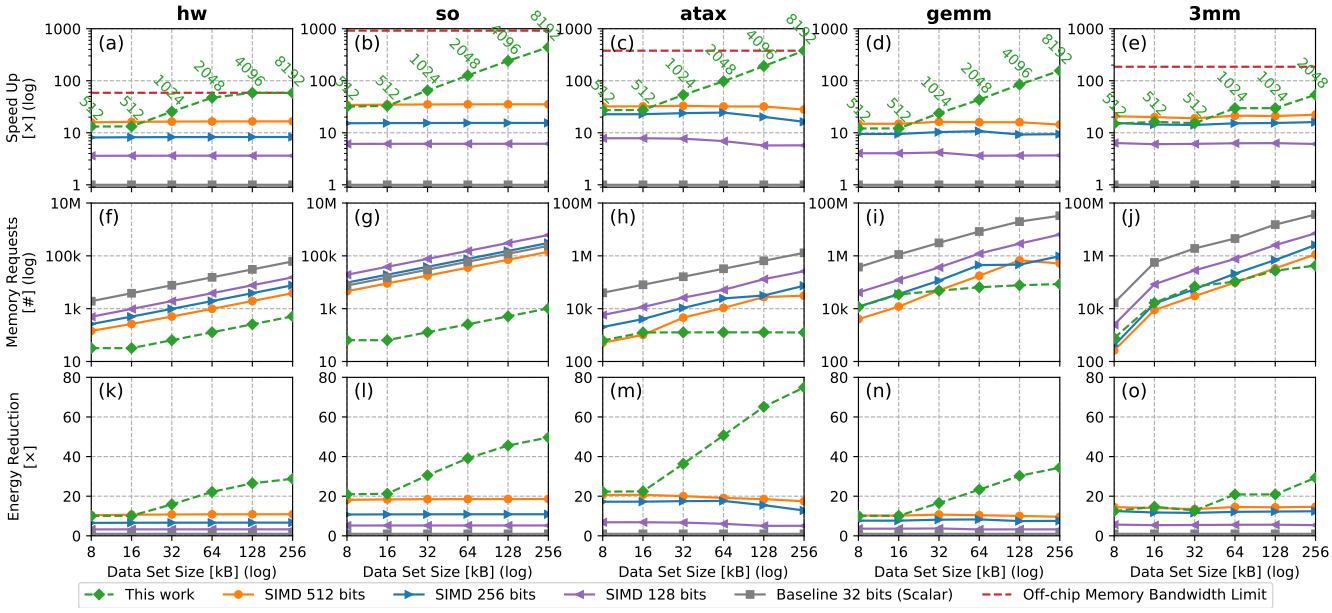

Figure 4: Simulation results of execution speed up, memory requests and energy reduction of five kernels, compared to the scalar architecture (RISC-V with SRAM). Each label (in *Speed Up*) represents the CIM vector width (in bit) for each data set size.

as $E = A.B; F = C.D; G = E.F$. These five kernels represent different application profiles in terms of memory patterns and computing requirement.

5.2 SIMD Comparison Methodology

To evaluate SIMD programs on Intel's SIMD processors, we use *Pin-3.11* [16], a dynamic binary instrumentation program developed by Intel. Thanks to the set of analysis tools provided, we extract and log CPU/SIMD instructions and memory accesses sorted by vector width for SSE2 (SIMD 128 bits), AVX-2 (SIMD 256 bits) and AVX-512 (SIMD 512 bits). Using this instruction level profiling, we obtain an instruction level analysis and, based on values established in Table 1, we evaluate each SIMD power state proportional to the bus width access. For example, an instruction of 512-bit SIMD access to 16 data of 32 bits is equivalent in energy to 16 times the memory access of the RISC-V core. All SIMD instructions including control flow and standard memory accesses are also counted and their cost evaluated.

5.3 Architecture Exploration Results

In Figure 4, we compare four architectures against the baseline RISC-V scalar architecture for different data set sizes and report execution speed up, memory requests and energy reduction. For both SIMD and CIM architectures, all data are pre-loaded in their 256 kB associated SRAMs, and the CIM cluster set-up is a 4×16 array of 4 kB tile enabling an horizontal scalability of 512-bit up to 8192-bit vector width. For each data set size, the maximum vectorization width is configured for each architecture and the labels in the speed up results represent the CIM vector width used. For small data set (8 kB and 16 kB), vectorization is limited to 512 bits, where CIM and SIMD architectures are compared for the same 512-bit width.

Table 2: Memory accesses / instruction analysis of the scalar baseline to identify memory and compute bound kernels.

Kernel	`hw`	`so`	`atax`	`gemm`	`3mm`
Memory Accesses (%)	9.1	2.3	27.6	37.1	28.4
Compute Instructions (%)	90.9	97.7	72.4	62.9	71.6

For larger data sets, the largest CIM vector width is selected (up to 8192 bits) while compared to the maximum SIMD available width. Regarding the Table 2, the `so` and `hw` kernels refer as compute bound kernels and `atax`, `gemm` and `3mm` refer as memory bound kernels.

Speed Up Exploration (a,b,c,d,e) – At equivalent vector width (8 kB and 16 kB), the execution speed up shows that the CIM architecture has gains close to the 512-bit SIMD architecture. This minor difference is due to the additional control flow of the proposed ISA, necessary to execute the instructions in memory (constants, loop preparation). However, the CIM architecture continues to increase its speed up using larger vectors, wider than 512 bits for larger data sets contrary to SIMD architectures, limited to 512 bits. Data dependencies in memory bound kernel are mostly resolved by using the dynamic tiling configurations from 8192-bit down to 512-bit for reduction operations as explained in 3.2.

External Memory Bandwidth Limitation (a,b,c,e) – For modeling external memory access, we propose a simple bandwidth limitation using an 10 GB/s memory bandwidth (LPDDR4). By considering the data movement from an external memory, the top-line represents the minimum time to transfer the data set in the cluster compared to the execution time. For all kernels besides `hw` and `atax`, this limitation is not reached, which means that enough memory bandwidth is available to feed the kernel. Even using CIM architecture, off-chip memory accesses is still a concern.

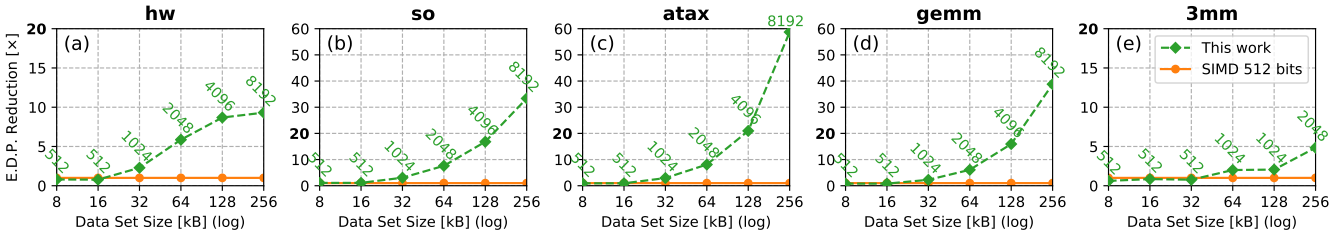

Figure 5: Simulation results of EDP reduction compared to the 512-bit SIMD. Each label is the CIM vector width used (in bit).

Memory Requests Exploration (f,g,h,i,j) – The CIM architecture strongly reduces the number of standard memory requests (read/write) with the CPU. However, for memory bound kernel, additional memory accesses are still required to complete the reduction operation. For a data set size of 256 kB, the vectorization of the `3mm` kernel allows a maximum vector of 2048 bits with the CIM architecture, thus having more reduction operation and smaller speed up gain than the other kernel for this data set size.

Energy Reduction Exploration (k,l,m,n,o) – Contrary to the SIMD architecture, the CIM architecture continues to increase its speed up while still maintaining a high energy reduction, thanks to the reduction of data movement between processor and memory replaced by the vertical data transfers between C-SRAM tiles. By using an 8192-bit vector width on `atax` kernel, the CIM architecture achieves energy reduction of 75× and 4.3× compared to the scalar baseline and the 512-bit SIMD architectures respectively.

Energy Delay Product Reduction Exploration – The Figure 5 presents the relative EDP reduction of the CIM architecture compared to SIMD 512 bits architecture using the same exploration methodology as in Figure 4. For memory bound kernels, an EDP reduction up to 60× is achieved, while for compute bound kernels, where more control flow is still required, an EDP reduction of 34× is achieved. Finally, the EDP gain of `hw` kernel for the CIM architecture is limited due to the external memory bandwidth limitation as discussed before.

6 CONCLUSION

In this paper, we proposed a reconfigurable and scalable vector-tile-based architecture and a simulation platform to evaluate the benefits of CIM. Our solution allows to implement computing applications requiring vector acceleration, without changing the overall architecture and reusing vectorized kernels already written for SIMD architectures. For each kernel, performance gains are achieved thanks to large vectorization in line with the application kernel. The proposed horizontal reconfigurability and vertical communication scheme allow to select the optimal vectorization width for maximum performance, while the SIMD is limited with the maximum width of 512 bits. Using equivalent vector-based instructions, compared to 512-bit SIMD architecture, the CIM architecture achieves an EDP reduction up to 60× and 34× for memory bound kernels and for compute bound kernels, respectively. This architecture is a possible option towards refining the N3XT architecture [19], integrating fine grain distributed computing within the 3D memory-computing system. For future works, we will implement the physical design of our reconfigurable vector-tile-based CIM architecture, compare it to other vector architectures, and explore kernels with 3D data movement using larger data sets.

REFERENCES

[1] 2012. IEEE Standard for Standard SystemC Language Reference Manual. *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)* (Jan. 2012).
[2] 2018. *Western Digital's Open Source RISC-V SweRV Instruction Set Simulator.*
[3] A. Agrawal et al. 2018. X-SRAM: Enabling In-Memory Boolean Computations in CMOS Static Random Access Memories. *IEEE Transactions on Circuits and Systems I: Regular Papers* (2018).
[4] K. C. Akyel et al. 2016. DRC2: Dynamically Reconfigurable Computing Circuit based on memory architecture. In *IEEE International Conference on Rebooting Computing (ICRC).*
[5] J.-F. Christmann et al. 2019. A 50.5 ns Wake-Up-Latency 11.2 pJ/Inst Asynchronous Wake-Up Controller in FDSOI 28 nm. *Journal of Low Power Electronics and Applications* (2019).
[6] Q. Dong et al. 2017. A 4 + 2T SRAM for Searching and In-Memory Computing With 0.3-V VDDmin. *IEEE Journal of Solid-State Circuits* (2017).
[7] C. Eckert et al. 2018. Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks. In *ACM/IEEE International Symposium on Computer Architecture (ISCA).*
[8] S. Faro et al. 2013. The Exact Online String Matching Problem: a Review of the Most Recent Results. *ACM Computing Surveys (CSUR)* (2013).
[9] R. Gauchi et al. 2019. Memory Sizing of a Scalable SRAM In-Memory Computing Tile Based Architecture. In *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC).*
[10] S. Hamdioui et al. 2015. Memristor based computation-in-memory architecture for data-intensive applications. In *Design, Automation & Test in Europe Conference & Exhibition (DATE).*
[11] M. Horowitz. 2014. 1.1 Computing's energy problem (and what we can do about it). In *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC).*
[12] J. Wang et al. 2019. A Compute SRAM with Bit-Serial Integer/Floating-Point Operations for Programmable In-Memory Vector Acceleration. In *IEEE International Solid-State Circuits Conference (ISSCC).*
[13] M. Kooli et al. 2017. Software Platform Dedicated for In-Memory Computing Circuit Evaluation. In *IEEE/ACM International Symposium on Rapid System Prototyping (RSP).*
[14] M. Kooli et al. 2018. Smart instruction codes for in-memory computing architectures compatible with standard SRAM interfaces. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)* (Dresden, Germany).
[15] H. Lebreton et al. 2008. Power Modeling in SystemC at Transaction Level, Application to a DVFS Architecture. In *IEEE Computer Society Annual Symposium on VLSI.*
[16] G. Lueck et al. 2012. PinADX: An interface for customizable debugging with dynamic instrumentation. *International Symposium on Code Generation and Optimization (CGO)* (2012).
[17] J.-P. Noel et al. 2020. Computational SRAM Design Automation using Pushed-Rule Bitcells for Energy-Efficient Vector Processing. In *Design, Automation & Test in Europe Conference & Exhibition (DATE).*
[18] L.-N. Pouchet. 2015. *PolyBench/C, the Polyhedral Benchmark suite.*
[19] M. M. Sabry Aly et al. 2019. The N3XT Approach to Energy-Efficient Abundant-Data Computing. *Proc. IEEE* (2019).
[20] William Andrew Simon et al. 2019. BLADE: A BitLine Accelerator for Devices on the Edge. *Proceedings of the ACM Great Lakes Symposium on VLSI (GLSVLSI)* (2019).
[21] G. Singh et al. 2018. A Review of Near-Memory Computing Architectures: Opportunities and Challenges. In *21st Euromicro Conference on Digital System Design (DSD).*
[22] N. Stephens et al. 2017. The ARM Scalable Vector Extension. *IEEE Micro* (2017).