Improving Post-Quantum Cryptography coupling Near-Memory Computing and RISC-V Cores

Maria Ramirez Corrales, Emanuele Valea and Jean Philippe Noel

Univ. Grenoble Alpes, CEA, List, F-38000 Grenoble, France

Abstract

Near-Memory Computing (NMC) is a promising architectural approach to accelerate and improve the efficiency of matrix products, since it drastically reduces the transfer of data between the CPU and the data memory. In this paper, we propose to couple a NMC co-processor with a RISC-V based CPU to accelerate the matrix product in Post-Quantum Cryptography (PQC) algorithms. Experimental results on the matrix product of FrodoKEM PQC algorithm show a 4X improvement in performance with respect to the same implementation without the NMC approach.

Introduction

Near-memory computing (NMC) is an architectural approach with the aim of overcoming the huge gap, existing in modern computing applications, between the time spent to transfer data between the host processor and the data memory and the time spent for the computation itself. This is particularly relevant for data-intensive applications centered on matrix products (e.g. machine learning, image processing, etc.). Thanks to NMC approaches, we limit data exchanges between the host processor and the data memory, so the processor will only execute one custom command, in order to make the memory start the computation in parallel on all data that are already stored on the same memory vector-line (at least 128 bits). Digital modules are added close to the data memory to process the data after it is read. For this reason, the memory array keeps unmodified and all the computation is performed in its periphery, which eases the integration in larger systems, such as RISC-V architectures, and makes it technology agnostic [1].

Post-Quantum Cryptography (PQC) is an interesting use case that handles matrix-vector products. Currently, researchers have investigated software implementations and hardware accelerators for PQC algorithms, including FrodoKEM-640 that is based on matrix-vector products. Instead of dedicated accelerators, NMC-based co-processors, coupled with a host processor, could be used to efficiently speed-up PQC algorithms [2]. On the other hand, as they are not application-specific but domain-specific, they serve as a "partial" accelerator as they can just accelerate matrix-vector products in some algorithms, such as the previously mentioned FrodoKEM-640, where these operations take around 50% of the execution time [3].

In this paper, we show the potential of a computer architecture combining a NMC-based co-processor and a RISC-V CPU to accelerate the matrix product of a PQC application. The NMC-based co-processor in this paper is called Computational SRAM (C-SRAM). It is based on an SRAM memory with a *Near-Memory Processing Unit* (NMPU) that is able to decode specific commands and to perform arithmetic operations on any vector-line of the SRAM memory array.

RISC-V-based NMC architecture

Figure 1 shows the structure of the proposed system. We used a CV32E40 CPU, based on the RISC-V architecture. The CPU is connected to the memories through a system bus based on the Open Bus Interface standard. An instruction memory of 16KB and a data memory of 256KB (i.e., the C-SRAM) are connected to the CPU.

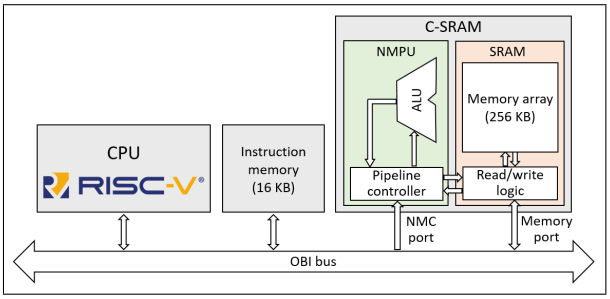

Figure 1: *High-level schematic of the system.*

The C-SRAM is an SRAM memory with NMC capabilities. SRAM memories are connected to the system bus and they can be accessed by the processor for either reading/writing data or fetching program instructions. We use the C-SRAM as both data memory and NMC co-processor. It can be accessed by the *memory port* or the *NMC port*. A vector *arithmetic logic unit* (ALU) is the core of the NMPU. The read/write logic is shared between the *memory port* and the NMPU for reading the operands and writing the results of NMC operations. The C-SRAM contains a memory array made of 128-bit lines. The memory array of the C-SRAM can be generated by a generic memory compiler, making the NMPU compatible with any memory technology. The *memory port* serves to access the C-SRAM as a traditional memory. For this

purpose, the whole memory is mapped to a range of addresses that, when accessed through load/store instructions, allow to read/write data inside the SRAM memory array. The *NMC port* is used to access the NMPU, that executes the NMC instructions. A range of addresses at system-level is reserved to NMC operations. Each time the RISC-V runs a store instruction on NMC addresses, the received values are decoded and processed by the NMPU. An NMC operation is composed of three elements that must be encoded on the bus transaction: the *opcode* defines the NMC operation; the *source* and *destination* addresses define the memory vector-lines that store the operands and the result. C-macros are used in order to generate NMC instructions at compile-time while allowing programmers to use an assembly-like syntax for inserting NMC instructions in their program. This method makes the software support for the C-SRAM straightforward, without the need of modifying the RISC-V compilation toolchain.

NMC instructions are processed by the NMPU through a pipeline made of 6 stages:

- *Decode*: the NMPU receives the instruction and it parses bus transactions in order to decode the instruction opcode and the operands.
- *Memory Read* (two pipeline stages): the addresses retrieved from the NMC instruction are used by the NMPU to read the memory array and fetch the content of two memory vector-lines.
- *Execution* (two pipeline stages): a vectorial ALU is placed inside the NMPU and, according to the opcode, it performs a specific operation on the two operands. Since the operands are memory vector-lines, they can be interpreted as vectors of values of different sizes. For instance, as the memory lines are 128-bit long, and the instructions correspond to 16-bit additions, the ALU executes eight operations in parallel. The second stage is used for *multiply-accumulate* (MAC) operations.
- *Writeback*: the resulting vector is written inside the destination memory vector line.

Results on PQC Application

In this Section, we show the results in terms of performance of the proposed architecture using the C-SRAM as a data memory and as a co-processor when running FrodoKEM-640 Key Generation function. 47% of the execution time of this algorithm is spent on executing matrix-vector products. The SW implementation has been compiled with the `riscv32-unknown-elf-gcc` compiler, with `-O1` optimization level. Figure 2 compares the execution time of the RISC-V implementation of FrodoKEM-640 between the reference implementation (i.e., where the C-SRAM is used only as data memory) and the NMC implementation (i.e., where the C-SRAM is used as a co-processor). If we

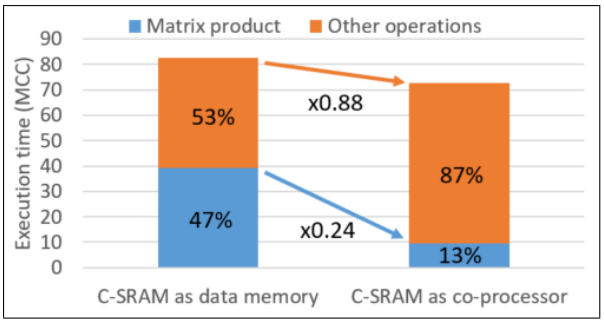

Figure 2: *FrodoKEM-640 execution times in terms of millions of clock cycles.*

compare the reference RISC-V SW implementation to the state of the art implementations on ARM M4 microprocessors [3], we observe a 3% gain on the execution time. The SW implementation, coupled with the NMC approach, provides a gain of 76% in the matrix-vector product execution time and a 12% gain on the whole function with respect to the reference implementation. This corresponds to a 4x acceleration factor in the matrix product execution time. Since eight MAC operations are performed in parallel, a performance gain of a factor eight was expected. However, the reality of the proposed NMC-based implementation is impacted by some overhead, explained by the post-processing of the MAC results. In fact, when eight MAC operations are performed in parallel on the NMPU, eight partial results are obtained. These intermediate values are then summed up together by the RISC-V to obtain the final result.

Conclusions

We showed the interest of NMC-based architectures coupled with RISC-V processors to improve the performance of PQC algorithms such as FrodoKEM-640. We have achieved a 12% speedup on the whole algorithm by accelerating the matrix product (where a major acceleration of 76% is achieved). This work shows the interest of further exploring the potentiality of the NMC approach for cryptographic algorithms, since the proposed C-SRAM shows a very smooth integration with the RISC-V architecture and its toolchain.

References

[1] M. Kooli et al. "Towards a Truly Integrated Vector Processing Unit for Memory-Bound Applications Based on a Cost-Competitive Computational SRAM Design Solution". In: 18.2 (Apr. 2022).

[2] D. Bellizia et al. "Post-Quantum Cryptography: Challenges and Opportunities for Robust and Secure HW Design". In: *DFTS 2021*. 2021, pp. 1–6.

[3] J. Howe et al. "Standard Lattice-Based Key Encapsulation on Embedded Devices". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018.3 (Aug. 2018), pp. 372–393.