POLITECNICO DI TORINO
Repository ISTITUZIONALE

Scalability analysis of multi-bank near-memory computing in low-power SoCs

Original
Scalability analysis of multi-bank near-memory computing in low-power SoCs / Giuffrida, L., Schiavone, P.D., Caon, M., Masera, G., Martina, M., Atienza, D.. - (2025), pp. 1-4. (33rd IFIP/IEEE International Conference on Very Large Scale Integration, VLSI-SoC 2025 Puerto Varas (Chl) 12-15 October 2025) [10.1109/vlsi-soc64688.2025.11421768].

Availability:
This version is available at: 11583/3015728 since: 2026-10-06T11:45:52Z

Publisher:
IEEE

Published
DOI:10.1109/vlsi-soc64688.2025.11421768

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2025 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

20 November 2026

Scalability analysis of multi-bank near-memory computing in low-power SoCs

Luigi Giuffrida* Pasquale Davide Schiavone† Michele Caon† Guido Masera* Maurizio Martina* David Atienza†

*VLSI Lab
Politecnico di Torino, Italy
{luigi.giuffrida,guido.masera,maurizio.martina}@polito.it

†Embedded Systems Laboratory
EPFL, Switzerland
{davide.schiavone,michele.caon,david.atienza}@epfl.ch

***Abstract*—Machine learning and artificial intelligence are moving towards the edge, where the need for high throughput with a constrained energy budget is more urgent than ever. During the last few years, near-memory computing has emerged as a promising solution to address the memory bandwidth and energy efficiency limitations of conventional von Neumann systems. The recently proposed NM-Carus architecture combines vector-oriented computing capabilities within a RISC-V programmable, configurable, and autonomous memory macro, addressing the usability of near-memory computing from a software deployment standpoint. In this paper, we explore the scalability of NM-Carus in terms of computation parallelism, memory size and energy consumption, As a benchmarking platform, we rely on a low-power microcontroller that features multiple instances of NM-Carus that target the execution of biomedical applications. This exploration was performed on 16nm TSMC NM-Carus implmentation, and we highlighted the benefits of technology scaling for a previous implementation on 65nm with respect to the overhead of replacing conventional on-chip data SRAMs with near-memory computing banks. Overall, the paper presents a solid baseline regarding the trade-offs in terms of area, performance, and energy efficiency of integrating programmable near-memory computing in an existing edge-oriented system on chip towards efficient edge AI architectures at the system level.**

***Index Terms*—Near-memory computing, Machine Learning, RISC-V**

I. Introduction

The increasing prevalence of data-driven workloads executed close to sensors necessitates a paradigm shift in computing toward enhanced energy efficiency and performance. The conventional von Neumann architecture, the foundation of computing for decades, exhibits inherent inefficiencies for data-intensive tasks due to constant data and instruction movement between the memory hierarchy and processing units [1]. This issue is compounded by the ”memory wall”, where the advancements of SRAM integration technology lag behind logic scaling, resulting in SRAM accesses consuming significantly more energy than arithmetic operations [2].

To address this challenge, Compute-In-Memory (CIM) paradigm, which includes In-Memory Computing (IMC) and Near-Memory Computing (NMC), has been proposed to reduce data movement and optimize utilization of the available memory bandwidth [3]. NMCNMC does not require the memory-array layout to be redesigned, but relies on commercially available memories to store data tightly connected to specialized data paths. However, controlling operations on IMC and NMC IPs presents difficulties. While a simple approach involves streaming micro-operations to the CIM blocks, complex runtime control demands either significant CPU overhead for encoding operations or reliance on predefined command sequences. This limitation has been addressed or partially mitigated by Single Instruction Multiple Data (SIMD) vector capabilities as in [4], [5], [6], [7], [8]. In particular, the NM-Carus NMC IP [7], integrates a CPU-based controller supporting a custom, vector-inspired CIM-oriented RISC-V ISA extension (`xmnmc`) to handle and autonomously execute virtually any data processing kernel, while the host system is only responsible for storing the data in the NMC memory, adhering to the kernel data placement requirements. Depending on the target physical and performance constraints, selecting the best combination of memory size and amount of computing resources of the NMC blocks requires exploration of the trade-off of wiring costs, performance, and energy [6]. In this paper, we provide an exploration of the scalability of NM-Carus in terms of throughput, area, and energy consumption. Unlike [7], where the performance assessment was limited to a single 32kB instance with 4 lanes, this work proposes a broader design space exploration, whose major contributions are: (1) scalability analysis of NM-Carus single-instance throughput and energy efficiency with varying memory size and amount of computing resources; (2) Analysis of multi--instance performance scalability; (3) Assessment of the area overhead at the instance and system level; (4) evaluation of the technological benefits of a 16nm NM-Carus implementation compared to its original 65nm variant. Section II analyses the timing, area, and energy scalability of different configurations of NM-Carus, while Section III evaluates the performance improvement enabled by a system featuring multiple instances of NM-Carus. Finally, concluding remarks and a summary of the findings are presented in Section IV.

II. Exploration

In this Section, different configurations in terms of memory size and number of lanes of the NM-Carus IP [7] are explored to find the best trade-offs and scalability figures.

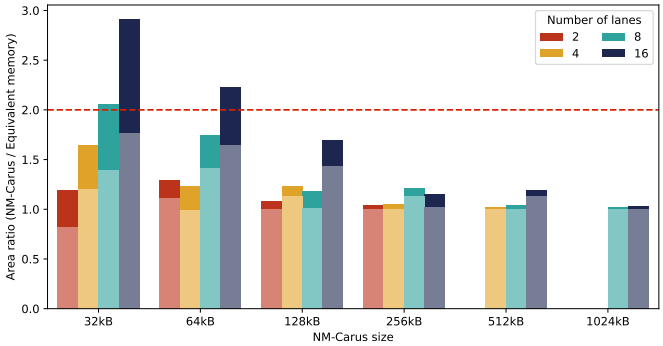


Figure 1: Ratio between the area of differently configured NM-Carus instances and conventional single-bank SRAM memories with equivalent sizes. The dashed line represents the NMC overhead threshold defined in [7] as equal to 2. Bars are split into internal SRAM area (lighter color) and the logic cell area (darker color).

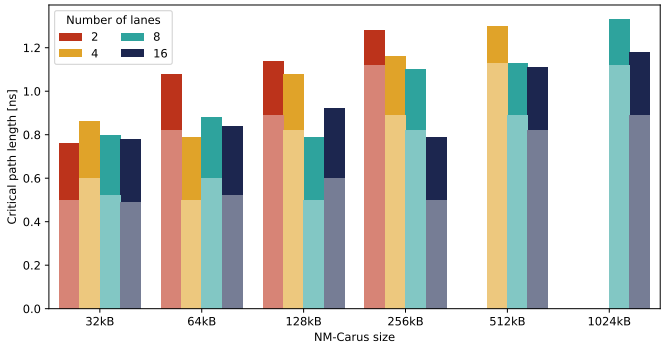


Figure 2: Critical path of the NM-Carus configurations. Bars are split into two: the lighter part represents the portion of the timing portion taken by the SRAM banks, while the darker part represents the delay introduced by logic cells.

The NM-Carus architecture can be configured with two main parameters: the number of processing lanes and the total memory size. The architecture is built such that the number of lanes is equal to the number of internal SRAM banks. Increasing the number of lanes not only determines a linear replication of the computational datapath (i.e., ALUs) but also results in an increase in total area due to the inherent overhead associated with memory sub-banking (i.e., the combined area of multiple smaller memory banks exceeds that of a single larger bank with equivalent size). Conversely, for kernels exhibiting optimal mapping onto SIMD architectures, like matrix multiplications, performance demonstrates a near-linear scaling with the number of processing lanes, directly increasing the throughput in terms of operations per cycle. For instance, when running a matrix multiplication kernel, 4-, 8-, and 16-lane configurations are $1.93\times$, $3.61\times$, and $6.38\times$ faster than a configuration with 2 lanes, respectively. The size of each sub-bank, M_{LANE} is determined by:

$$M_{LANE} = M_{TOT}/N_{LANES}, \tag{1}$$

where M_{TOT} is the total NM-Carus memory size and N_{LANES} is the number of lanes. The area overhead of NM-Carus, relative to a standard SRAM bank of equivalent capacity, is the ratio between the area of an NM-Carus instance and that of a conventional single-bank SRAM with the same capacity. Adopting the same threshold established in [7], a ratio exceeding 2 indicates excessive area overhead, making the NM-Carus IP instance impractical as a direct replacement for a conventional SRAM bank. Figure 1 depicts the area overhead of NM-Carus instances with different combinations of memory capacity and lane count, subdivided in SRAM area and logic cells area. To keep the total NM-Carus area under constraints, both the internal sub-banks and the reference single-bank SRAM memories were generated with the PDK compiler targeting the minimal area. As shown in Figure 1, the 32 KiB 16-lane NM-Carus configuration exhibits the highest area overhead, at $2.9\times$. This substantial overhead can be attributed to two primary factors: the area increase resulting from partitioning the 32 KiB memory into 16 individual 2 KiB sub-banks ($2.08\times$) and the area contribution of the 16 processing elements, including the integrated CPU controller and its instruction memory, which constitutes 39.7% of the total area. In general, the area overhead of the considered NM-Carus variants demonstrates an inverse relationship with their memory capacity. This trend arises from the diminishing relative overhead associated with sub-bank splitting as the total memory size increases. Furthermore, the area of the computing logic is solely determined by the number of lanes and remains constant regardless of the total memory size, thus contributing to a smaller relative overhead for larger configurations. Consequently, for memory capacities of 128 KiB and greater, the 16-lane configurations exhibit lower area overhead compared to their counterparts with smaller total memory sizes.

Increasing the complexity of the near-memory processing logic not only affects the silicon area but also the maximum operating frequency, which is still primarily influenced by both the number and the timing characteristics of the sub-banks. As illustrated in Figure 2, the 128 KiB, 16-lane configuration, comprising 16 8 KiB sub-banks, exhibits a $1.14\times$ reduction in operating frequency compared to a 128 KiB variant configured with 8 lanes and thus 8 larger 16 KiB sub-banks. This disparity arises from a combination of factors: the higher speed of the larger 16 KiB sub-banks (to minimize the area, banks are generated with wider rows, thus slower decoders and increased depth of multiplexers) and the increased latency introduced by the more complex vector slide logic required to move data across different lanes. In contrast, for a 32 KiB instance, the configuration with 2 banks of 16 KiB achieves the highest operating frequency, running $1.02\times$ faster than the 32 KiB variant with 16 smaller 2 KiB banks. In this scenario, while the smaller 2 KiB sub-banks exhibit better timing characteristics, the deeper vector slide logic associated with the larger number of banks ultimately limits the overall performance. Figure 2 highlights how the critical path is dominated by the clock-to-output delay of the memories (lighter color). Configurations that share the same internal memory bank sizes have the same critical path, as the 32 KiB 4-lane and the 64 KiB 8-lane variants, both using 8 KiB internal memory sub-banks. Figure

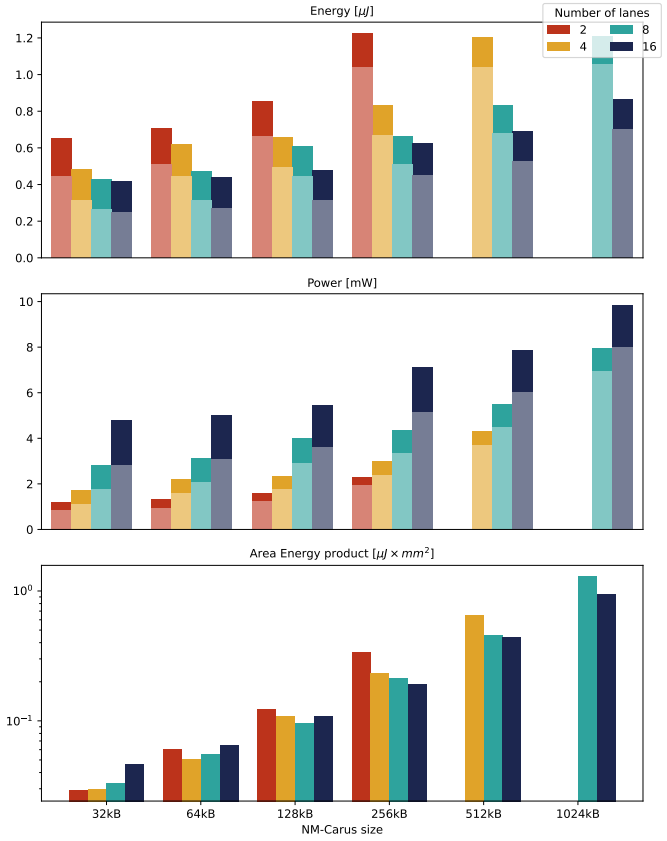


Figure 3: Power, energy and area energy product of NM-Carus for different configurations when running at 100 MHz an 8-bit matrix multiplication ($A[10 \times 10] \times B[10 \times 1024]$). Power and energy bars are split into two: the lighter part represents the portion of the power dissipated by SRAM banks, while the darker part represents the logic cells.

3 shows the power consumption of NM-Carus configurations when running an 8-bit matrix multiplication at 100 MHz under typical conditions (0.8 V, 25 °C). Power consumption is dominated by the internal memory sub-banks (lighter color). Given a fixed memory capacity, a higher number of lanes is associated with a near-linear increase in computational logic power, mostly due to its dynamic power contribution, whereas the embedded CPU controller's consumption remains negligible, accounting for only 5% to 7% of the total power depending on the configuration. Regarding energy consumption, Figure 3 illustrates how the increase in power consumption associated with a higher number of processing lanes is compensated by the improved performance of the matrix multiplication kernel executions, showing that a higher number of lanes improves performance more than the power overhead in all configurations. However, when taking into account the area overhead, which is crucial for considering NM-Carus an SRAM replacement, the best compromise depends on the configuration. Figure 3 shows the area-energy product on a logarithmic scale, revealing that 2, 4, and 8 lanes are optimal for 32 KiB, 64 KiB, and 128 KiB respectively. While for 256 KiB and beyond, 16 lanes offer the best trade-off.

Comparing the 32 KiB 4-lane 16 nm NM-Carus configuration from Figures 1 to 3 with its 65 nm counterpart from [7], it is possible to appreciate the benefits of technological scaling on such memory-oriented architectures. In the former case, the area overhead is 27 %, significantly lower than the 58 % overhead in 65 nm. The lower scaling trend in SRAM manufacturing compared to logic scaling ensures more headroom for cost-performance optimization. From a timing perspective, the 16 nm implementation features a much higher operating frequency at 1.16 GHz compared to the 330 MHz in 65 nm. Such timing characteristics support the exploitation of NM-Carus in integrated last-level data cache, as proposed in [8]. Similarly, technological scaling allows for an estimated 2.1 × reduction in power consumption at 100 MHz.

III. System integration use-case

To demonstrate the performance improvement of the NM-Carus instances, two distinct configurations were integrated into the memory subsystem of the X-HEEP [9] RISC-V microcontroller. The X-HEEP system has been configured to include one CV32E40P [10] core, four DMA channels, standard peripherals, and a 352 KiB on-chip SRAM memory subsystem partitioned into 64 KiB for instruction storage, 32 KiB for private data, two 64 KiB banks for extra data, and 128 KiB with interleaving addressing scheme on four banks. The two 64 KiB banks for extra data have been replaced with the NM-Carus instances, configured as one with 4 processing lanes, and one with 8 processing lanes. Unlike Arcane [8], where several homogeneous instances of NM-Carus are used within a cache and exploited by means of host CPU custom ISA extensions, in this work the memories are scratch-pad memories which require explicit data tiling and layout at application level, as well as loading specific near-memory computing kernels for each instance, which allows running heterogeneous kernels in parallel. The two NMC instances can operate either in memory mode or as computational acceleration engines, depending on the operational mode as detailed in [7]. The X-HEEP [9] system was synthesized using the TSMC 16nm LVT process node, targeting a worst-case operating frequency of 330 MHz (0.72 V, 125 °C, slow process conditions). Figure 4 shows the area breakdown of the X-HEEP system, including the NMC IPs. The area overhead of the 4-lane and 8-lane NM-Carus instances relative to equivalent SRAM implementations is 1.23× and 1.74×, respectively, contributing with an increase of 10% of the overall system area. To showcase the speedup that NM-Carus can introduce in an actual system, a matrix multiplication task has been executed on the X-HEEP system with the two NM-Carus instances as a benchmarking application. The size of the two matrix operands are $A[30 \times 15]$ and $B[15 \times 2048]$. Both are stored in an off-chip flash chip and copied with quad SPI to the on-chip SRAN memory. We analyzed three different scenarios: execution of the matrix multiplication on the CPU, execution on a single instance of NM-Carus, and execution on both the NMC instances. For the reference execution on the CPU, first, the DMA copies both matrices from the external flash to the interleaved memory banks, then the CPU executes the kernel, storing the result in another portion of the same memory region. For the single

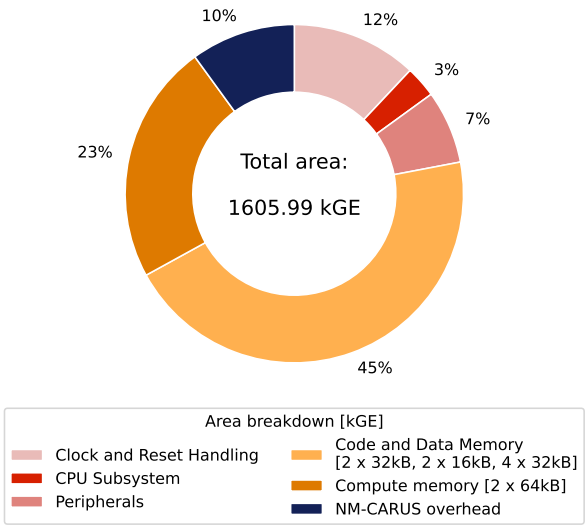


Figure 4: Area breakdown of X-HEEP with two NM-Carus instances. Area reported on the pie-chart is in equivalent kilogates (kGE).

NM-Carus execution, tiling is performed as the output of the matrix multiplication would not fit the NMC memory. After the DMA has copied the 15 rows of A and the complete matrix B, NM-Carus executes the kernel in place. When done, the results are copied to the system memory to leave space for the next tile. This solution requires an extra data-movement operation as the NM-Carus size is smaller than the problem size. While the deployment using both instances of NM-Carus splits the matrix A into two and replicates the matrix B, allowing for the execution of the matrix multiplication using two parallel NM-Carus banks. The multi-channel DMA is used to hide latencies in a double-buffering scheme. Employing configurations with 4 and 8 lanes balances execution. Due to the sequential nature of data transfers from flash memory, the higher speed of the configuration with 8 lanes, is exploited to complete concurrently, despite starting later. The execution of the application with NM-Carus shows a speedup of $19.85\times$ and $29.34\times$ faster than the CPU for the single and double NM-Carus configurations, respectively.

IV. Conclusions

This study has presented a scalability analysis of NM-Carus IP [7], investigating the trade-offs between memory capacity and the number of parallel processing lanes. We quantitatively compared the area, maximum frequency, power, and energy characteristics of various NM-Carus configurations against equivalent standard SRAM implementations in the TSMC 16nm LVT technology node. The results indicate that the most significant contributor to area overhead is the partitioning of the total memory into smaller sub-banks. In contrast, the impact on the area of the integrated computational logic decreases for larger total memory capacities. The maximum frequency exhibits a nonlinear relationship with the number of lanes and memory organization. In some configurations, increasing the number of sub-banks (resulting from a higher lane count) leads to a reduction in maximum frequency due to the increased complexity of the addressing. Conversely, in other configurations, the utilization of larger capacity sub-banks with faster access due to area-optimized sub-bank configurations can mitigate or even reverse this trend. In addition, while the power is dominated by the internal sub-banks' consumption, energy shows that more parallelism is always beneficial. However, when combining area and energy, the best configurations depend on the total memory size. Finally, we integrated two heterogeneous configurations into the X-HEEP system, showing that at the system level, using two instances of NM-Carus increases only by 10% the total area, while improving speed up to $29.34\times$.

Acknowledgments

This work was supported in part by the Swiss State Secretariat for Education, Research, and Innovation (SERI) through the SwissChips research project. This work is part of the SERICS project (PE00000014), under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. In addition, we thank Mr. Francesco Poluzzi for supporting the deployment of end-to-end applications.

References

[1] J. Backus, "Can programming be liberated from the von neumann style? A functional style and its algebra of programs," *Commun. ACM*, vol. 21, no. 8, pp. 613–641, aug 1978.

[2] S. A. McKee, "Reflections on the memory wall," in *Proceedings of the 1st Conference on Computing Frontiers*. Association for Computing Machinery, 2004, p. 162.

[3] S. Srinivasa, A. K. Ramanathan, J. Sundaram, D. Kurian, S. Gopal, N. Jain, A. Srinivasan, R. Iyer, V. Narayanan, and T. Karnik, "Trends and opportunities for SRAM based in-memory and near-memory computation," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, 2021, pp. 547–552.

[4] Y. Wang, M. Yang, C.-P. Lo, and J. P. Kulkarni, "30.6 Vecim: A 289.13GOPS/W RISC-V vector co-processor with compute-in-memory vector register file for efficient high-performance computing," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67, 2024, pp. 492–494.

[5] M. Kooli, A. Heraud, H.-P. Charles, B. Giraud, R. Gauchi, M. Ezzadeen, K. Mambu, V. Egloff, and J.-P. Noel, "Towards a truly integrated vector processing unit for memory-bound applications based on a cost-competitive computational SRAM design solution," *J. Emerg. Technol. Comput. Syst.*, vol. 18, no. 2, apr 2022.

[6] R. Gauchi, V. Egloff, M. Kooli, J.-P. Noel, B. Giraud, P. Vivet, S. Mitra, and H.-P. Charles, "Reconfigurable tiles of computing-in-memory SRAM architecture for scalable vectorization," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 121–126.

[7] M. Caon, C. Choné, P. D. Schiavone, A. Levisse, G. Masera, M. Martina, and D. Atienza, "Scalable and risc-v programmable near-memory computing architectures for edge nodes," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–15, 2025.

[8] V. Petrolo, F. Guella, M. Caon, P. D. Schiavone, G. Masera, and M. Martina, "Arcane: Adaptive risc-v cache architecture for near-memory extensions," *arXiv preprint arXiv:2504.02533*, 2025.

[9] S. Machetti, P. D. Schiavone, T. C. Müller, M. Peón-Quirós, and D. Atienza, "X-heep: An open-source, configurable and extendible risc-v microcontroller for the exploration of ultra-low-power edge accelerators," *arXiv preprint arXiv:2401.05548*, 2024.

[10] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 25, no. 10, pp. 2700–2713, 2017.