Evaluating Energy Efficiency of Genomics Algorithms on Processing-in-Memory Architectures

Meven Mognol[1], Florestan De Moor[1], Erwan Drezen[2], Yann Falevoz[3], and Dominique Lavenier[1][0000−0003−2557−680X]

[1] Univ. Rennes, CNRS-IRISA, Inria, Campus de beaulieu, Rennes, France
lavenier@irisa.fr
[2] Institut Pasteur, Paris, France
[3] UPMEM, Grenoble, France

Abstract. Processing-in-Memory (PiM) is a novel computing paradigm for reducing data movements between memory and processing units, and thus minimizing energy consumption. PiMs are particularly well-suited to data-intensive applications, where traditional systems are often limited by memory bandwidth. Genomics is a representative example of such a domain, involving massive datasets and repetitive access patterns. In this paper, we evaluate the energy efficiency improvements achieved by running several genomic algorithms on a PiM-based system. Our experiments focus on realistic workloads and highlight the challenges and opportunities of parallelizing genomic tasks for PiM. The most significant gains are observed in large-scale database search applications, which naturally map to the parallel structure of PiM and benefit greatly from reduced data movement.

Keywords: Processing-in-Memory · Parallelism · Power consumption · Genomics

1 Introduction

Computer applications have become ubiquitous in all aspects of our daily lives, from communication, entertainment, and commerce, to scientific and medical domains. This ubiquity comes with a fundamental shift in the nature of these applications: they are no longer primarily focused on computation, but rather on data manipulation.

This evolution presents significant technological challenges. Traditional computer architectures, such as the Von Neumann model developed in the 1940s, are increasingly ill-suited to the demands of modern computing. By separating the processing units (CPUs) from memory, the Von Neumann architecture proved highly efficient for decades, especially in tasks involving extensive data reuse, like matrix multiplication, where data read from memory could be reused many times in complex calculations. However, this model is no longer well adapted to the performance requirements of contemporary applications.

Data-centric applications often involve less regular memory access patterns, with simpler computations and limited data reuse. This diminishes the effectiveness of cache hierarchies and prefetching mechanisms. As a result, performance degrades significantly not because processors lack computing power, but because they spend an increasing amount of time waiting for data. This phenomenon is known as the memory wall [1].

The memory wall refers to the growing performance gap between processors, whose speed and capabilities have improved dramatically over the past decades, and memory systems, whose latency and bandwidth have not progressed at the same rate. This asymmetry results in processors being chronically underutilized, as they wait for data to be transferred from main memory. The energy cost of these data transfers is also significant, often hundreds of times greater than that of arithmetic operations on the same data. For example, adding two 32-bit floating point numbers costs around 1 picojoules, while accessing just one of those numbers in DRAM consumes more than 1000 picojoules [2].

To address these limitations, the Processing-in-Memory (PiM) paradigm emerged [3]. The core idea of PiM is to move processing closer to the data by integrating computation units directly into memory components. This model significantly reduces data transfers between memory and processor, increases effective bandwidth, lowers latency, and above all, decreases overall energy consumption. PiM is not intended to replace traditional processors but rather to offer a complementary solution, particularly well suited to massively parallel tasks and localized processing on large datasets.

Among the practical industrial initiatives in this area, the French company UPMEM offers an innovative PiM solution based on integrating small processing cores, called DPUs (Data Processing Units), directly into DRAM memory chips [4]. These units can execute processing algorithms directly on the data stored within their own memory segment, thereby avoiding costly transfers to the central processor. UPMEM's technology enables massive parallelism and can be integrated into standard servers, providing a pragmatic solution for data-heavy applications.

This article builds on this technological innovation and aims to concretely evaluate the energy gains enabled by PiM in a particularly data-intensive field: genomics. With the democratization of sequencing technologies and the rapid drop in costs, the volume of genetic data being produced has exploded. Processing this data, although not always requiring extreme computational power, involves complex tasks such as sequence comparison, database searching, and genome assembly. These operations are inherently data-intensive, involve many memory access and need limited arithmetic operations, making them ideal candidates for execution on PiM architectures.

As part of the European BioPIM [20] and GenoPIM [21] projects, several bioinformatics algorithms have been adapted and parallelized to run on UPMEM's PiM architecture. This work presents the experimental results obtained from these applications, highlighting the energy savings achieved.

2 PiM architecture

As stated before, Processing-in-Memory architectures aim to address the inherent limitations of classical von Neumann systems by bringing computation closer to memory. This proximity reduces data transfers, improves effective bandwidth, and decreases energy consumption. Two main PiM architecture types are commonly identified: in-memory architectures and near-memory architectures [9]. These approaches differ in how closely the processing units are integrated with the memory subsystem.

2.1 In-Memory vs. Near-Memory Architectures

In-memory architectures represent the most radical form of the PiM paradigm. In this configuration, the computing elements—often simple logic gates or basic arithmetic units—are directly embedded inside memory cells, typically within DRAM or SRAM arrays. This extreme co-location enables data to be processed without physically moving it, effectively eliminating memory transfers. The key advantage of this model is its extremely low latency and minimal energy consumption. However, the tight integration of logic into memory arrays imposes significant constraints on memory density and fabrication complexity. As a result, in-memory PiM is often limited to specialized operations such as bitwise logic, counting, or simple vector arithmetic.

By contrast, **near-memory architectures** adopt a more modular strategy. In these designs, processing units are placed close to the memory, but not within the memory arrays themselves. Typically, they reside on the same die or memory module (e.g., DIMMs). This approach supports more complex processing units, such as full RISC cores, while still maintaining low latency and high bandwidth thanks to tight coupling with the memory. While energy savings are generally smaller than those of in-memory PiM, near-memory architectures offer better programmability, greater architectural flexibility, and compatibility with existing fabrication processes.

2.2 The UPMEM PiM Architecture

Among the most advanced industrial realizations of the near-memory paradigm, the architecture developed by the French company UPMEM offers a compelling and functional implementation. It is based on a hybrid Processing-in-Memory model integrated into standard DDR4 DRAM modules, allowing seamless compatibility with x86 servers as a drop-in replacement for conventional DIMMs.

The UPMEM PiM module adopts a dual-rank architecture, with each rank containing eight DRAM chips, for a total of 16 chips per module (cf fig. 1) . Each chip is structured into eight independent memory banks of 64 MB, resulting in an overall capacity of 8 GB. What sets this module apart from traditional memory architectures is the integration of Data Processing Units (DPUs) that are lightweight RISC processors embedded near the memory arrays, one per memory bank. This results in 128 DPUs per module, each associated with a

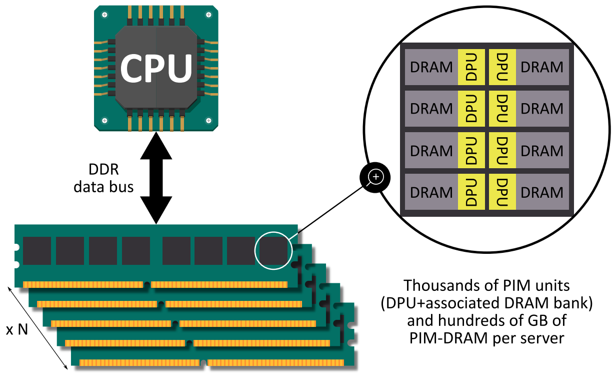
CPU
DDR data bus
x N
DRAM
DPU
Thousands of PIM units (DPU+associated DRAM bank) and hundreds of GB of PIM-DRAM per server

Fig. 1. Server with PiM DIMMs.

dedicated 64 MB local memory bank, referred to as Main RAM (MRAM) in UPMEM's terminology.

Each DPU is a general-purpose, energy-efficient core, capable of running up to 16 hardware threads (tasklets) concurrently. This high level of parallelism enables efficient in-memory data processing, particularly well-suited for workloads involving massive data sets and localized computation. The triadic instruction set architecture (ISA) implemented on the DPUs is specifically optimized for control flow and branching. Furthermore, DPUs operate independently, without any interconnection, reinforcing their role as autonomous, parallel workers within the memory system.

By relocating computation directly to the memory subsystem, this architecture drastically reduces data movement between memory and CPU, which is one of the primary causes of energy inefficiency in conventional systems. As a result, memory bandwidth usage drops, the host CPU is relieved of massively parallel or repetitive tasks, and overall system energy efficiency improves. The CPU retains the role of orchestrator, coordinating task dispatching and global execution flow, while the DPUs perform distributed, fine-grained computation independently.

UPMEM offers a dedicated development environment to harness the full potential of its PiM architecture. Based on a heterogeneous computing model, the host CPU manages and communicates with DPUs, which execute data-parallel tasks using C-based kernels. The environment includes an LLVM-based compiler toolchain, a runtime API, and debugging tools. DPUs access local MRAM and can perform efficient data transfers with the host. Execution typically involves initialization, kernel execution, and result retrieval. UPMEM also provides tools for profiling and power monitoring, helping developers optimize application performance and energy efficiency on PiM-enabled systems.

2.3 Fine-grain parallelism challenge

While Processing-in-Memory architectures such as UPMEM's offer promising performance and energy efficiency gains, they also introduce a major programming challenge: scaling applications to thousands of DPUs, each capable of running 12 to 16 concurrent tasklets. Achieving optimal performance requires fine-grain parallelization, where the application must be decomposed into tens of thousands of lightweight threads. This level of concurrency demands not only efficient workload distribution but also careful management of memory access patterns and synchronization. Designing algorithms that can leverage such massive parallelism remains an open and active area of research, especially for irregular and data-intensive applications.

3 Experimental Setup

This study aims to assess the energy savings enabled by Processing-in-Memory architectures through the implementation and parallelization of several algorithms drawn from the field of genomics, a representative class of data-intensive applications. This section presents the hardware platform used for experimentation, describes the various programs selected for the evaluation, and outlines the methodology used to calculate energy savings.

3.1 Hardware

On the hardware side, we run all benchmarks on the same server with an Intel® Xeon® Silver 4215 processor (16 hyper-threaded cores) at 2.5 GHz (Skylake architecture), 256 GB DDR4 RAM at 2.4 GHz and 20 UPMEM DIMMs at 350 MHz (corresponding to 40 ranks, or 2,560 DPUs, and a total of 160 GB MRAM memory). The server runs under Debian 10 and uses version 2023.2.0 of the UPMEM SDK.

To evaluate energy consumption, we connected a power meter that samples power consumption of the server every 10 milliseconds. The wattmeter comes from AdecWatts [18], a company that designs professional equipment for measuring energy consumption. The measurement accuracy error rate is less than 1%, which is more than sufficient for our purposes. This device provides an accurate trace (from an USB port) of the server's energy consumption over time, as shown in Figure 2. From this trace, the average energy consumption of an application execution can be calculated: it is computed as the product of its execution time with its average power consumption.

3.2 Genomic algorithms

The algorithms analyzed in this study originate from the field of genomics. Their parallelization on a UPMEM server was carried out as part of the BioPIM [20] and GenoPIM [21] projects. All evaluations were performed under real usage

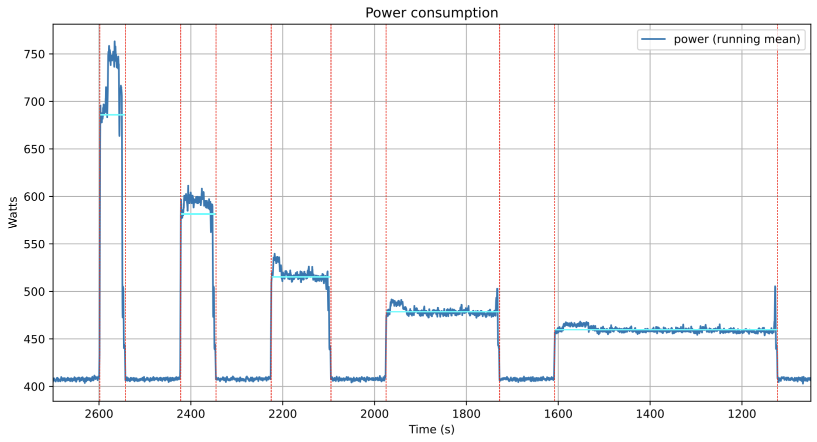

Fig. 2. Example of the trace of the server's energy consumption over time

conditions and should not be considered estimations. A selection of algorithms with diverse characteristics, particularly in terms of memory access patterns, was made in order to assess their impact from an energy efficiency perspective. The algorithms evaluated are the following:

- **Sorting.** In genomics, k-mer counting represents a fundamental computational task. K-mers are short nucleotide sequences of length k, extracted from DNA sequences, and serve as the foundation for constructing more complex data structures widely used in genomic algorithms. The k-mer counting process relies heavily on sorting algorithms, whose parallel implementations have been evaluated in this study.
- **Compression of sequencing data.** Sequencing costs have been steadily decreasing, while throughput capabilities have increased proportionally. As a result, the volume of sequencing data to be stored continues to grow, highlighting the need for highly efficient storage solutions. Given the frequent use of compression and decompression in genomic workflows, these operations need be highly optimized.
- **Mapping.** Mapping involves identifying the occurrence of short sequences, typically a few hundred base pairs, within reference genomes that are several orders of magnitude larger. Although computationally intensive, this process is routinely employed in a wide range of genomic applications, particularly within the healthcare domain.
- **Sequence Comparison.** Sequence comparison is another essential task in genomics, aimed at computing distances between two or more DNA or protein sequences. It is a core component of numerous bioinformatics pipelines and often constitutes a major computational bottleneck due to the high algorithmic complexity involved.

– **Protein database search.** Protein database search facilitates the identification of relationships between unknown sequences and known genes by detecting significant sequence similarities. This task is routinely performed by biologists as part of genomic and proteomic workflows. Among bioinformatics applications, database search is particularly well suited to Processing-in-Memory (PiM) architectures. In this paradigm, the database is permanently stored in memory, and query sequences are broadcast across all DPUs, each of which independently searches for relevant similarities and returns the corresponding results.

3.3 Evaluation of the energy gain

Energy gains are calculated as the ratio of the energy consumption of the reference software to that of its corresponding PiM implementation. Both implementations process the same dataset and produce identical or comparable results. They are also executed on the same server. The reference software is run in parallel, utilizing all available cores whenever possible.

When the server is in idle mode—meaning no user applications are running—it still consumes a significant amount of power, approximately 410 Watts in this case. While this may seem surprising, such idle consumption is consistent with findings from other studies on the topic [10].

To accurately assess the energy savings, it would be necessary to measure software reference power consumption without PiM modules. The UPMEM server contains 20 modules (40 ranks). When these modules are not in use, according to UPMEM designers, the power consumption per module is around 5 Watts. We can therefore estimate that the energy consumption of this server without PiM components, and when no user application is running, is about 310 Watts (410 - 20 $\times$ 5). This estimate is in line with the study published in [5]. Similarly, when the number of PiM modules used is less than 20, the idle power of unused PiM modules must be subtracted.

Hence, from real measurements on our 20 PiM DIMM server, the energy gain (EG) can be computed as follows:

$$EG = \frac{ET_{REF} \times (PW_{REF} - 20 \times PW_{PD})}{ET_{PiM} \times (PW_{PiM} - (20 - K) \times PW_{PD})}$$

PW_{PD}	Estimated power of a PiM module in idle mode (5 Watts)
K	Number of PiM modules used
PW_{REF} / ET_{REF}	Average power measured / Exec. time of reference software
PW_{PiM} / ET_{PiM}	Average power measured / Exec. time of PiM software

The numerator represents the energy consumed on a server without PiM modules. The denominator represents the energy consumed on a server equipped with K PiM modules.

4 Results

4.1 Sorting

Three sorting algorithms (Quick Sort, Heap Sort, and Radix Sort) were evaluated. The input array is partitioned into sub-arrays, each assigned to a DPU tasklet for local sorting, ensuring balanced workload distribution. Following the local sort, the sub-arrays are merged to produce the final sorted output. Since the only variation between the CPU and PiM implementations lies in the local sorting phase, the benchmark focuses exclusively on this step. Each tasklet processes a sub-array of 512K 32-bit integers. With 16 tasklets per DPU, this results in 8 million integers sorted per DPU, and scales up to sorting an array of 20 billion elements in total.

The performance of the PiM implementations is benchmarked against their CPU counterparts, i.e. three state-of-the-art implementations. For Quick sort, the Intel AVX-optimized version [23] is considered. For Heap sort, we designed an home made optimized software. For Radix sort, an optimized parallel version, called PARADIS [16], have been used. The following table reports the mean consumption in Watts together with the energy saving. Execution times (in second) are indicated in parentheses.

	CPU	**PiM**	**Energy**	**Speed**
	32 threads	40 ranks	**Gain**	**Up**
Quick	527 W (21.2 sec.)	861 W (20.5 sec.)	0.51x	1.03x
Heap	503 W (169.8 sec.)	847 W (122.3 sec.)	0.66x	1.38x
Radix	666 W (31.1 sec.)	821 W (24.5 sec.)	0.87x	1.26x

These results clearly demonstrate that implementing this type of computation on PiM offers no added value, either in terms of processing speed or energy efficiency. This can be attributed to the availability of highly optimized implementations of sorting algorithms that take advantage of (1) the efficient SIMD instructions available on modern processors; (2) the exploitation by cache memories of the locality of data.

4.2 Compression of sequencing data

We selected Genozip [14] as the reference software, given its status as one of the leading tools for efficient and fast compression of sequencing data. Its counterpart, MiMyCS [15], was specifically developed to leverage PiM architectures. However, only a portion of the compression algorithm has been offloaded to PiM, specifically the part that is both highly parallelizable and the most computationally intensive.

The dataset used for compression originates from a whole genome sequencing project of Homo sapiens, obtained through the Illumina NovaSeq 6000 technology (SRR14724533). This dataset is accessible for download on the European Nucleotide Archive browser [8].

Genozip was executed in fast mode and allocates 1.1 threads per core in order to maximize usage of all available cores. In this configuration, it achieves a compression rate comparable to that of MiMyCS. For both software applications, a reference genome is employed to ensure optimal compression. In this experiment, we utilized the GRCh38 reference human genome.

To compress the dataset corresponding to human sequencing data, MiMyCS requires only 8 ranks of PiM memory out of the 40 available. The following table presents the average power consumption and execution time for both implementations.

CPU 35 threads	**PiM** 8 ranks	**Energy Gain**	**Speed Up**
526 W (231 sec.)	525 W (162 sec.)	1.4x	1.4x

MiMyCS demonstrates slightly higher speed and lower power consumption than its counterpart. The increase in the number of ranks does not translate into higher speed or energy savings, as the bottleneck here comes from accessing the data from disk. Compression is nearly performed at maximum disk bandwidth.

4.3 Mapping

Bowtie2 [11], a state-of-the-art mapping tool, serves as the CPU reference implementation. In this context, millions of sequencing reads (short DNA fragments) are aligned against a reference composed of one or more genomes.

To evaluate mapping under different conditions, two metagenomic datasets are employed, each representing a distinct mapping scenario: low and high mapping rates. The first dataset, originates from the TARA Oceans expedition and includes sequencing reads along with corresponding Metagenome-Assembled Genomes (MAGs) used as references. Due to the extensive microbial diversity in marine environments, this dataset represents a low-mapping scenario, where only a small proportion of reads align to any given reference genome. The second dataset, UHGG (Unified Human Gut Genome), is derived from the Human gut microbiome. In this case, reads from a single individual are mapped against thousands of bacterial genomes. The relatively low genetic divergence results in a high-mapping scenario, characterized by a large proportion of successful alignments.

	CPU 32 threads	**PiM** 40 ranks	**Energy Gain**	**Speed Up**
TARA	523 W (144 sec.)	504 W (46 sec.)	2.7x	3.1x
UHGG	695 W (375 sec.)	540 W (136 sec.)	3.0x	2.8x

In both cases, the PiM implementation is faster and provides significant energy gain. However, a closer look at the profiling shows that DPUs are not being used to their full potential [17]. Mapping is a relatively fast process which, in this case, induces fast execution times compared to the time taken for exchanges between the host processor and the PiM memories.

4.4 Sequence comparison

Performance evaluation was conducted through a comparative analysis with a multi-threaded CPU implementation based on OpenMP, obtained from the minimap2 GitHub repository [22]. This implementation, which shares components with the KSW2 library [6], is vector-optimized using SSE instructions. The reference software is MiniMap2 [7].

Two real data sets of long DNA sequences were used. The first dataset consists of 9,557 16S ribosomal RNA sequences, extracted from the NCBI bacterial database (August 2022). The processing task is to perform all-to-all sequence comparisons, which is a prerequisite step for constructing a phylogenetic tree. The second dataset comprises 38,512 sets of PacBio DNA sequences. Each set contains between 10 and 30 similar sequences derived from the same genomic region. These raw sequence sets are processed to obtain the most probable genomic sequence in that region. The first step is to perform all-against-all alignments. The second step calculates a consensus sequence based on the alignments provided by the first step. The benchmark only takes into account the first step.

	CPU	**PiM**	**Energy**	**Speed**
	32 threads	40 ranks	**Gain**	**Up**
16S	484 W (5882 sec.)	832 W (632 sec.)	4.3x	9.3x
PacBio	433 W (4044 sec.)	800 W (505 sec.)	3.3x	8.0x

As we can see, parallelizing the process of comparing long DNA sequences on a PiM architecture brings significant energy savings, while also providing a significant acceleration factor. The calculations performed in each DPU are consistent and independent of each other. Memory exchanges are therefore greatly reduced, with the main processor only orchestrating the calculations.

4.5 Protein database search

The Blast [12] software suite remains the most widely used tool for performing DNA and protein database searches. In this context, the protein-specific variant, Blastp, is utilized as the reference software. The PiM implementation, called PANG [13], yields results comparable to those of Blastp in no-gap mode. For the experiment, Blastp has been run with 32 threads and PANG with 32 UPMEM PiM ranks.

The dataset consists of the SwissProt protein database, comprising approximately 560,000 curated protein sequences. The query set includes 10,000 protein sequences randomly selected from a separate sequence repository, ensuring no overlap with the reference database. The table below reports the electric consumption together with the elapsed execution time.

CPU 32 threads	**PiM** 32 ranks	**Energy** **Gain**	**Speed** **Up**
520 W (850 sec.)	686 W (56 sec.)	10.5x	15.2x

Here we are in an extremely favorable situation for PiM architectures. Computation times are greatly reduced, as is power consumption. In this type of application, a great deal of time is spent searching for “hits”, which are then used as a basis for calculating similarities. This search for hits generates a huge number of memory accesses, which do not follow regular patterns. Deporting this search directly into memory brings considerable gains.

5 Conclusion

This study demonstrates that not all genomic applications benefit equally from the PiM architecture, either in terms of computation acceleration or energy reduction. The most favorable use cases are those that can be broken down into thousands of independent tasks, enabling fine-grained parallelism. These applications can fully exploit the parallel processing units available in PiM systems, leading to efficient execution.

A strong correlation was observed between speedup and energy savings. In general, the greater the speedup achieved through parallelization, the more significant the reduction in energy consumption.

While the focus of this study was on genomic workloads, the conclusions can be extended to other domains with similar computational characteristics. Applications that support fine-grained parallelism and that exhibit intensive memory access are particularly well-suited to PiM. In such cases, the architecture effectively minimizes the traditional bottleneck of data movement. The optimal scenario involves a dataset that remains resident in PiM memory and is queried or processed intensively. Under these conditions, PiM enables low-latency, energy-efficient processing by reducing memory transfer overhead.

Overall, PiM shows strong potential as an architectural solution for both high-performance and energy-aware computing, particularly in domains where massive parallelism and data locality can be exploited.

Acknowledgments. The work described in this paper is funded by the European Union’s Horizon Europe Program for research and innovation under grant 101047160 (BioPIM project), and by the French ANR program ANR-21-CE46-0012 GenoPIM.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. McKee, S.A., Wisniewski, R.W.: Memory Wall. In: Padua, D. (eds) Encyclopedia of Parallel Computing. Springer, Boston, MA, 2011.

2. Horowitz, M.: 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), 2014.
3. Asifuzzaman, K., et al.:A survey on processing-in-memory techniques: Advances and challenges, Memories, Materials, Devices, Circuits and Systems, vol. 4, Dec. 2022.
4. Devaux, F.: The true Processing In Memory accelerator, 2019 IEEE Hot Chips 31 Symposium (HCS), Cupertino, CA, USA, 2019.
5. Falevoz, Y., Legriel, J.: Energy Efficiency Impact of Processing in Memory: A Comprehensive Review of Workloads on the UPMEM Architecture. In: Zeinalipour, D., et al. Euro-Par 2023: Parallel Processing Workshops. Euro-Par 2023. Lecture Notes in Computer Science, vol 14352, 2024.
6. Li, H.: Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics, 34:3094-3100, 2018.
7. Li, H.:New strategies to improve minimap2 alignment accuracy. Bioinformatics, 37:4572-4574, 2021.
8. Leinonen, R. and al.: The European Nucleotide Archive, Nucleic Acids Research, vol. 39, no. suppl 1, pp. D28–D31, Jan. 2011.
9. Fujiki, D., Wang, X., Subramaniyan, A., Das, R.: in-/near-Memory Computing, Synthesis Lectures on Computer Architectures, Morgan & Claypool Publishers, 2021
10. Jay, M., Ostapenco, V., Lefèvre, L., Trystram, D., Orgerie A-C., et al.: An experimental comparison of software-based power meters: focus on CPU and GPU, CCGrid 2023 - 23rd IEEE/ACM international symposium on cluster, cloud and internet computing, Bangalore, India, 2023
11. Langmead, B., Wilks, C., Antonescu, V., Charles, R.: Scaling read aligners to hundreds of threads on general-purpose processors, Bioinformatics, Volume 35, Issue 3, Pages 421–432, 2019.
12. Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J.: Gapped BLAST and PSI-BLAST: a new generation of protein database search programs, Nucleic Acids Research, 25(17), 1997.
13. Airault, C., Deltel, C., De Moor, F., Drezen, E., Mognol, M., Lavenier, D.: Protein database search using Processing-in-Memory architecture, IPDPS workshop HiComb 2025, Milan, Italie, 2025.
14. Lan, D., Tobler, R., Souilmi, Y., Llamas, B.: Genozip: a universal extensible genomic data compressor, Bioinformatics, Volume 37, Issue 16, August 2021.
15. De Moor, F., Mognol, M., Deltel, C., Drezen, E., Legriel, J., Lavenier D.: MiMyCS: A Processing-in-Memory Read Mapper for Compressing Next-Gen Sequencing Datasets, BIBM 2024, IEEE International Conference on Bioinformatics and Biomedicine, Lisbon, Portugal, 2024.
16. Cho, M., Brand, D., Bordawekar, R., Finkler, U., Kulandaisamy, V., Puri, R.: PARADIS: an efficient parallel algorithm for in-place radix sort. Proc. VLDB Endow. 8, 12, 2015.
17. Mognol, M.: Acceleration of Bioinformatics Algorithms on a Processing-in-Memory Architecture, PhD These, Rennes, 2025
18. AdecWatts company home page, https://www.adecwatts.fr/
19. CPU HeapSort impelementation gitlab, https://gitlab.inria.fr/pim/org.pim.sort/
20. BioPIM project homepage, https://biopim.eu/
21. GenoPIM project homepage, https://genopim.irisa.fr/
22. MiniMap2 github, https://github.com/lh3/minimap2
23. x86-smd-sort github, https://github.com/intel/x86-simd-sort