KrakenOnMem

A Memristor-Augmented HW/SW Framework for Taxonomic Profiling

Shahroodi, Taha; Zahedi, Mahdi; Singh, Abhairaj; Wong, Stephan; Hamdioui, Said

DOI
10.1145/3524059.3532367

Publication date
2022

Document Version
Final published version

Published in
Proceedings of the 36th ACM International Conference on Supercomputing, ICS 2022

KrakenOnMem: A Memristor-Augmented HW/SW Framework for Taxonomic Profiling

Taha Shahroodi
TU Delft
Delft, Netherlands
T.Shahroodi@tudelft.nl

Mahdi Zahedi
TU Delft
Delft, Netherlands
M.Z.Zahedi@tudelft.nl

Abhairaj Singh
TU Delft
Delft, Netherlands
A.Singh-5@tudelft.nl

Stephan Wong
TU Delft
Delft, Netherlands
J.S.S.M.Wong@tudelft.nl

Said Hamdioui
TU Delft
Delft, Netherlands
S.Hamdioui@tudelft.nl

ABSTRACT

State-of-the-art taxonomic profilers that comprise the first step in larger-context metagenomic studies have proven to be computationally intensive, i.e., while accurate, they come at the cost of high latency and energy consumption. *Table Lookup* operation is a primary bottleneck of today's profilers. In this paper, we first propose TL-PIM, a hardware accelerator based on the processing-in-memory (PIM) paradigm to accelerate *Table Lookup*. TL-PIM leverages the in-memory compute capability of emerging memory technologies along with intelligent data mapping. Then, we integrate TL-PIM into Kraken2, a state-of-the-art metagenomic profiler, and build an HW/SW co-designed profiler, called KrakenOnMem. Results from a silicon-based prototype of our emerging memory validate the design and required operations on a smaller scale. Our large-scale calibrated simulations show that KrakenOnMem can provide an average of 61.3% speedup compared to original Kraken2 for end-to-end profiling. Additionally, our design improves the energy consumption by orders of magnitude compared to the original Kraken2 while incurring a negligible area overhead.

CCS CONCEPTS

• **Hardware → Memory and dense storage**; **Bio-embedded electronics**.

KEYWORDS

Taxonomic Profiling, Emerging Memories, In Memory Processing, (Hash) *Table Lookup*, Kraken2

ACM Reference Format:
Taha Shahroodi, Mahdi Zahedi, Abhairaj Singh, Stephan Wong, and Said Hamdioui. 2022. KrakenOnMem: A Memristor-Augmented HW/SW Framework for Taxonomic Profiling . In *2022 International Conference on Supercomputing (ICS '22), June 28–30, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3524059.3532367

ICS '22, June 28–30, 2022, Virtual Event, USA

ACM ISBN 978-1-4503-9281-5/22/06.
https://doi.org/10.1145/3524059.3532367

1 INTRODUCTION

Taxonomic profiling determines the relative abundance of existing species in a (biological) sample under study. It constitutes the first and most compute-intensive step of larger-context metagenomic studies (metagenomics for short). The goal of metagenomics is to better understand the role of each organism in our environment to improve our quality of life, e.g., by enhancing drugs [86]. The ability to improve the performance of taxonomic profilers will, therefore, have a huge impact on the overall speed of metagenomic studies and will remain a crucial line of research for decades to come.

Many recent works have improved the speed and/or accuracy of taxonomic profiling by various means, e.g., directly as heuristics in pre- and post-processing steps of profiling [40, 41], indirectly as pre-alignment filters [2] or innovative hardware designs for alignment [4, 16, 34]. However, the memory bandwidth and the (limited) cache capacities remain the two main bottlenecks even in these approaches [33, 83]. This is because *Table Lookup* (i.e., key matching and label retrieval) is a critical kernel in today's profilers and it is performed on data structures that are hundreds of gigabytes in size that cannot fit in caches of even high-performance computing (HPC) servers [51, 83, 87]. Note that it is also estimated that the working datasets that metagenomic studies should deal with scale faster than those produced by YouTube and Twitter by 2025 [5, 20, 82, 87], exacerbating this problem. Consequently, we need a fast, energy-efficient, scalable, and yet accurate design for taxonomic profilers (with an emphasis on their bottlenecks) to expedite the metagenomic studies and keep up with the fast data generation rate.

Our goal is to build the first hardware/software co-designed framework for taxonomic profiling that exploits *real* memristor (i.e., STT-MRAM) devices and the processing in-memory (PIM) paradigm. Using the PIM paradigm helps to prevent the high cost of data movement between memory and different levels of caches by performing the bottlenecked operations completely inside the memory, where the data resides. An in-memory solution can also scale up the active computational units without the need for expensive scale-up in the computational units of the server. **To this end,** we propose KrakenOnMem, an optimized framework for accelerating Kraken2[1] that notably improves execution time and energy consumption of taxonomic profiling with a negligible area

[1]Kraken2 is currently the most widely-used and one of the most promising taxonomic profilers based on recent metagenomics challenges.

overhead. KrakenOnMem is based on two main observations: (1) reference genomes rarely change, and (2) memristors are inherently capable of performing Vector-Matrix Multiplication (VMM). KrakenOnMem exploits these observations and addresses the bottlenecks of Kraken2 (and many other profilers), *Table Lookup*, using a memristor-based substrate, called TL-PIM hereafter. We perform an extensive design exploration for (1) data mapping, (2) logical operations, (3) array sizes, and (4) peripheral supports to optimize TL-PIM for *Table Lookup* and KrakenOnMem for taxonomic profiling based on Kraken2's algorithm. Our evaluations show that KrakenOnMem can provide up to 61.3% end-to-end speedup compared to original Kraken2 implementation.

Our paper makes the following contributions:

- To our knowledge, KrakenOnMem is the first HW/SW co-designed framework to accelerate taxonomic profiling using memristor devices and the PIM paradigm. We design KrakenOnMem to target the key bottleneck of SOTA taxonomic profilers.
- We propose TL-PIM, an in-memory accelerator that executes *Table Lookup* efficiently using an intelligent data duplication and hybrid row-major and column-major data mapping. TL-PIM is designed to harvest the maximum parallelism and performance of the underlying hardware (Section 4).
- We rigorously compared TL-PIM and KrakenOnMem to (1) Kraken2 (open-sourced) as a SOTA profiler and (2) Optimized Sieve as the latest in-memory accelerator[2]. We use a real small-scale prototype to validate our memory design. Our large-scale evaluations show that TL-PIM achieves an average 1386× and 111× speedup for *Table Lookup* operation compared to Kraken2 and Sieve, respectively. To capture the full potential of KrakenOnMem, we also perform a second set of analyses for end-to-end taxonomic profiling. KrakenOnMem achieves 61.3% and 1.17% speedup compared to Kraken2 and Sieve, respectively, for end-to-end taxonomic profiling. These improvements all come with the same level of accuracy as Kraken2 (Section 6).
- We investigate the possibility to adopt our designs in future (or non-heuristic-based) taxonomic profilers. TL-PIM integrated into Metalign, a SOTA alignment-based taxonomic profiler, show a 23.01% improvement for end-to-end profiling compared to original Metalign. This is achieved despite the fact that the bottleneck of Metalign does not lay on *Table Lookup* (Section 6).

2 PRELIMINARIES

In this section, we provide a high-level overview of metagenomics and taxonomic profiling, Kraken2, the CIM paradigm, and memristor devices along with their computation capability. For further details, we refer the reader to past review works on similar topics [7, 18, 23, 43, 54, 59].

[2]Sieve is a SOTA k-mer (a substring of length k) matching accelerator that we tuned for a similar profiling approach.

2.1 Metagenomics and Taxonomic Profiling

Recent advances in high-throughput sequencing (HTS), namely producing sequenced data with high-throughput and low cost, initiated metagenomics [24, 67, 86]. In metagenomics, researchers study the behavior of many species altogether in a sample taken directly from an environment. The results of such a study help researchers to capture the complex relationship between different species without cultivating or isolating them individually in a very costly or yet impossible procedure for some species.

Taxonomic profiling is the first step of any metagenomic study [40, 83] and it determines the relative abundance of different taxonomy ranks (e.g., species, genus, family) in a given sample. Taxonomic profiling is divided into two main categories [63]: reference-free and reference-based profilers.

Reference-free Profilers. MetaPhlAn [69, 75], PhymmBL [6], and PhyloPythiaS+ [22] are a few examples of reference-free profilers. These profilers are typically slow. They also require a relatively long query sequence for their composition feature needed for the classification. Therefore, although they are actively being investigated, researchers and industry do not currently use them for taxonomic profiling.

Reference-based Profilers. Reference-based taxonomic profilers can be further divided into two main classes: alignment-based and non-alignment-based (also known as heuristics). The alignment-based profilers are highly accurate (especially at the species rank). However, alignment-based profilers are very slow due to the high computational cost of their alignment. A few examples of such profilers are Metalign [40], MG- RASTv.4 [57], MEGAN6 [28], and Taxator-tk [13]. Non-alignment-based profilers, or heuristics, replace the time-consuming alignment operation with a faster *Table Lookup* operation. This way, taxonomic profilers in this group trade the required execution time with the memory needed for their table and lookup operation, i.e., they gain speed but require more memory. Kraken [85], Kraken2 [83], and CLARK [60] are a few examples of such profilers.

Kraken2+Bracken[3] always stands among the top taxonomic profilers and binners, varying just a little from dataset to dataset, based on the most comprehensive benchmark for metagenomics, Critical Assessment of Metagenome Interpretation (CAMI) challenge [56, 68]. It is worth noting that most of the highly ranked profilers in the CAMI challenge are non-alignment-based profilers.

2.2 Kraken, Kraken2, and Bracken

Kraken [85] is a non-alignment-based taxonomic profiler that utilizes exact-match database queries of small substrings from the main read, called k-mers. Kraken, first, stores all k-mers within the sequence into a set. It then maps each k-mer inside the set into the lowest common ancestor (LCA) taxon of all the genomes in the reference database that have the special k-mer. This LCA texa and its ancestors in the taxonomy tree form the classification tree used to classify the input sequence. The classification path is defined as the maximum scoring root-to-leaf (RTL) path in the classification

[3]Bracken [51] is an orthogonal method to Kraken2 and other profilers to re-distribute reads in the taxonomic tree and improve the accuracy. We discuss Bracken further in Section 2.2.

tree, and the sequence S is assigned to the label of the corresponding leaf. Kraken owns its efficiency not only to the classification algorithm but also to its database creation, in which it uses the notion of minimizers [66] and two tables for performing an efficient search. Kraken groups similar k-mers using minimizers, defined as the smallest M-mers among all M-mers in a k-mer when sorted lexicographically. Since adjacent k-mers share the same minimizers in practice, Kraken stored the k-mers with similar minimizers consecutively and sorted them in the lexicographical order of their canonical representations. This enables Kraken to query a k-mer by looking up in an index position and finding the range where k-mer with the same minimizer as the query has been stored and then perform a binary search within the region to find the exact k-mer and corresponding taxonomy ID.

Kraken, while effective, uses a memory-intensive algorithm for assigning queries to the lowest common ancestor (LCA) taxonomic label. Kraken2 [83] improves performance and memory consumption of Kraken by building a more compact reference database using probabilistic hash functions. Kraken2 significantly reduces the memory requirement to a third while maintaining accuracy. Kraken2 also takes advantage of block-based and batch-based parsing within the critical sections to further improve thread scaling, similar to what has been done in Bowtie [39]. Both Kraken and Kraken2 use extensive index (hash) tables to store a pre-built data structure to help accelerate their assignment. Therefore, they both perform multiple *Table Lookup* operations to map a DNA read to an LCA.

To prevent underestimating the abundance of some species, we typically use Bracken [51] (Bayesian Reestimation of Abundance after Classification with KrakEN) along with Kraken2. Bracken proposes to probabilistically re-distribute reads in the taxonomic tree so that estimating the abundance of species will become possible. The re-distribution in Bracken works in both directions: 1) Reads that are originally assigned to nodes above species levels will be re-distributed to this level, 2) Reads that are originally assigned to nodes in the strain level will be re-distributed to their parent species node. Bracken is orthogonal to Kraken and Kraken2.

2.3 Processing in Memory (PIM)

Processing-in-Memory (PIM)[4] is an old paradigm that is reignited as a promising solution to alleviate the data movement bottleneck in today's data-intensive applications being run upon processor-centric architectures. The PIM paradigm advocates for redesigning our systems such that they prevent unnecessary data movement between memory units and computational units in the first place [9, 17, 42, 71].

Previous PIM-enabled proposals can be categorized based on their underlying technology, type of operation/function they support, and the location they perform the operation/function and produce the output. They target various memory technologies, naming DRAM [44, 71], SRAM [1, 15, 19], and memristors [45, 91]. These proposals also vary from supporting simple logical operations (e.g., AND, OR, XOR) inside the memory arrays [71, 89] to a kernel or full application considering the whole memory, its peripheral and sometimes also help from an external processing unit like CPU [11, 32, 72].

[4]Interchangeably referred to as Computation-in-Memory (CIM).

2.4 Memristor Devices

A resistive memory or memristive device is a non-volatile emerging memory technology that can store the data in the form of its resistance level. ReRAM [80], PCM [42], and STT-RAM [78] are just a few examples of memristive devices [42, 80]. Memristive devices have recently been shown as a suitable candidate for both storage and computation units. In these devices, one can alter the resistance states of the device with suitable voltage or current pulses. We use SET to denote a transition from a low resistive state (LRS) to a high resistive state (HRS). Term RESET presents a reverse transition. These two states (LRS and HRS) then can present logical "1" and "0", respectively [59].

Recent works exploit an array of memristors to perform matrix-vector [88] and bulk bit-wise logical [10, 37, 89] operations efficiently since these devices follow Kirchhoff's law inherently. Therefore, many works propose memristor-based PIM-enabled designs for applications that require such operations heavily. Memristor devices also enjoy non-volatility, high-density, and near-zero standby power, making them suitable as future memory technologies.

3 MOTIVATION

In this section, we investigate the potential bottlenecks in Kraken2 and limitations of previous solutions for taxonomic profiling.

3.1 Kraken2's Execution Breakdown

Methodology. We evaluate Kraken2 on a high-end server and measure the execution time of separate functions for end-to-end profiling of our input files. We use the default parameters of the tool for our study. Query reads come from the CAMI challenge, and Kraken2's standard (default) reference genome dataset is used for the references. We detail our evaluation methodology further in Section 5.

Results & Analysis. Fig. 1 depicts the percentage breakdown regarding execution time of Kraken2. We classify the various functions of Kraken2's implementation [84] into four main groups: (1) Building Taxonomy Tree, (2) Key Extraction, (3) *Table Lookup*, (4) Profiling. The building taxonomy tree function is run only once for each reference genome database. This function does not exist in the profiling phase and can be considered as a pre-processing function. We included the breakdown to show the relative time proportion to other frequently run functions. The key extraction function is responsible for reading the query files, extracting minimizers, performing hash functions, and producing the keys. The *Table Lookup* function tests each query key against all the keys in the table and returns the associate value (label) if it finds a match. The profiling function is responsible for aggregating the results of the *Table Lookup* function and performs the final profiling processes alongside writing the data to a file.

Based on Fig. 1, Kraken2 spends more than 60% of its total execution time on performing the *Table Lookup* function. Therefore, *Table Lookup* is currently the bottleneck of Kraken2 as a SOTA taxonomic profiler. Our evaluations show that this humongous share does not

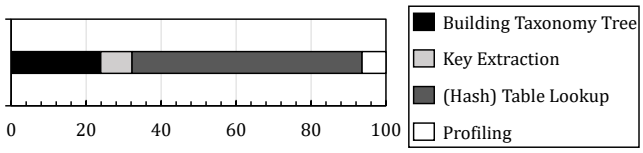

Figure 1: Kraken2's Execution Time Breakdown.

change significantly as we increase the number of active threads in the system and *Table Lookup* remains the bottleneck of Kraken2.

Moreover, it is important to note that increasing the available memory bandwidth does not improve the performance of Kraken2 significantly [87]. This is simply because memory bandwidth is highly underutilized in the *Table Lookup* function as miss status holding registers (MSHR) in caches will be used up quickly and prevent using the memory bandwidth fully. Using cores with more MSHRs (e.g., Broadwell cores) is also not a suitable solution as they come with massive energy consumption (i.e., cost) and still waste DRAM bandwidth as Kraken2 still uses a small number of the retrieved cache lines for each *Table Lookup* [87].

We conclude that *Table Lookup* is currently the bottleneck in SOTA Kraken2 profiler and will likely remain the main bottleneck of future non-alignment-based profilers for the same reasons unless hardware support is provided or profilers experience a cost-efficient, dramatic algorithmic change.

3.2 Limitation of Previous PIM-enabled Designs

A recent work, Sieve [87], proposes a high-throughput k-mer matching mechanism that uses in-DRAM processing. Sieve presents an Early Termination Mechanism (ETM) method that can interrupt the matching procedure of two k-mers as soon as the first mismatch occurs. This way Sieve can reduce the row activation required for its matching mechanism and reduce the latency and energy overheads compared to a naive implementation for k-mer matching. Technically, one can perform a *Table Lookup* function using Sieve's matching mechanism with a simple value retrieval approach. Therefore, it is reasonable to consider such an approach a suitable candidate for accelerating taxonomic profilers.

However, Sieve comes with four main limitations. First, high and unacceptable area overhead for a DRAM chip. Sieve requires up to 10.75% for its type III design which achieves the highest performance. Since DRAM chips are optimized for die area, this makes Sieve unlikely to be adopted in future systems. We compare Sieve's area with the proposal in Section 6.2. Second, Sieve requires considerable data duplication and high #writes for vertical placement of each query k-mer and its duplicates inside group patterns in Region 1 defined in the original manuscript. Based on the examples provided in the original manuscript [87], this can be up to 4× higher than the actual number of query k-mers. Since query k-mers are extracted directly from reads/queries and vary per input sample, this is not a one-time cost and cannot be justified. Moreover, although the chosen memory technology in Sieve, namely DRAM, does not suffer from the endurance problem, such a decision still comes with endurance problems and a high cost (energy consumption and time) for each query. This limitation also prevents applying ideas presented by Sieve to emerging memory technologies that still suffer from low endurance. Third, Sieve only builds on DRAM as its underlying PIM infrastructure. Sieve justifies this by the technology maturity, availability of simulation tools, and cost advantages compared to SRAM. However, it left out exploring NVM-based technologies entirely. Such memory technologies have been the focus of many recent accelerators since they enjoy non-volatility, high-density, near-zero standby power, and low-cost logical operations [49, 50]. Fourth, Sieve incurs a significant amount of internal data movement associated with the multi-row activation needed for matching. This is unavoidable because Sieve requires copying the operand rows to designated ones.

We argue that Sieve's limitations are more than what can be expected for the cost of a taxonomic profiler preventing it from being adopted in future systems. Sieve's limitations and our experimental observations motivate us to develop an in-situ *Table Lookup* accelerator integrated with a host processing unit that accelerates taxonomic profiling. Our design has four key objectives: (1) It should provide high *Table Lookup* performance. (2). It should scale linearly with the required memory for Kraken2, rather than the parameters of the (hash) table. (3) It should not impose any significant overheads for its additional hardware, such as logic circuits in the periphery. (4) It should incur minimum #writes, data movement, and data duplication.

4 KRAKENONMEM DESIGN

The low arithmetic intensity and high energy inefficiency of *Table Lookup*, the primary bottlenecks in Kraken2, limit the maximum attainable performance and increase the energy consumption on server clusters typically used for profiling. This sub-optimal performance and energy consumption happens for three reasons. First, the extensive indexes used for taxonomic profiling. Second, irregular memory access and poor cache hit rate of profilers. Third, unnecessary data movement between memory (where the indexes initially reside) and the system's rigid cache hierarchy. In a nutshell, taxonomic profilers that use *Table Lookup* do not fully utilize the available bandwidth of memory systems [87] for their operation. We mitigate this problem by proposing a PIM-enabled accelerator for *Table Lookup* using memristor devices. We call this design TL-PIM hereafter. We integrate TL-PIM into a full system and propose an HW/SW co-designed framework for taxonomic profiling based on Kraken2's algorithm. This framework is called KrakenOnMem henceforth.

4.1 A High-Level Overview

Fig. 2 presents a high-level overview of our entire KrakenOnMem framework, i.e., TL-PIM and its integration with the host CPU and storage unit. KrakenOnMem consists of 4 main components: ❶ Host CPU, ❷ Main Memory, ❸ Storage, and ❹ TL-PIM. Current taxonomic profilers share the first three components with KrakenOnMem, i.e., KrakenOnMem only adds TL-PIM.

Host CPU is responsible for the non-bottleneck steps of Kraken2. This includes ❶ building the reference (hash) table, ❷ loading the reference table into TL-PIM, ❸ reading the query read sequences from Fastq files and generating keys (e.g., extracting minimizers and calculating the hash values), and ❼ aggregating the retrieved taxonomic labels as the profiling result. Host CPU also sends the

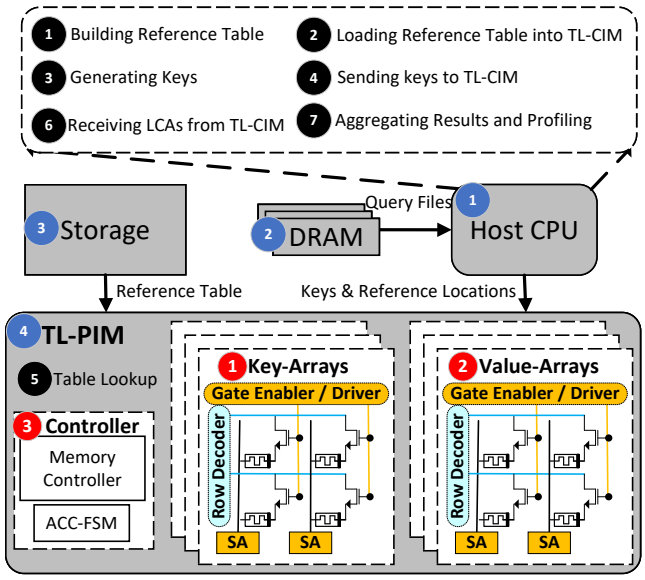

Figure 2: An Overview of KrakenOnMem.

keys to TL-PIM (❹) and receives the results back from the TL-PIM (❻).

TL-PIM accelerates *Table Lookup* ❺ (the bottleneck) and subsequently helps the overall performance and energy consumption. TL-PIM itself consists of 3 key components: (1) Key-Arrays: Memory arrays for matching units ❶, (2) LCA-Arrays: Memory arrays for taxonomic labels retrieval ❷, and (3) Controller ❸. Memory arrays are memristor arrays and their periphery circuits.

When designing the matching and retrieval units of TL-PIM, a designer faces three highly-correlated design choices and challenges, namely (1) data mapping, (2) matching mechanism, and (3) additional required hardware resources. For data mapping, TL-PIM should choose among possible options:

- Data layout: Row-major vs. Column-major
- Data distribution (i.e., keys or labels for queries or references): inter arrays vs. intra arrays vs. near arrays[5]

For the matching mechanism and its place, TL-PIM again have several options to choose from:

- Inside an array (e.g., using MAGIC [37] or other stateless methods)
- Outside the array but inside peripheries such as Sense Amplifiers (SAs) or Analoge Analog-to-Digital Converter (ADCs) (e.g., Pinatubo [45] or other stateful methods)
- Outside but near the array using simple logic after SAs (e.g., XOR and shift registers)
- A hybrid of previous options

Finally, TL-PIM should consider the required additional hardware resources for the matching or retrieval. These resources are for:

- The general control flow logic
- Modified peripheries (SAs or ADCs in the case of scouting or VMM operation)
- Fine-grained and complex control logic for analog operations in stateless mechanisms

We consider all possible (and logical) combinations of such design choices. KrakenOnMem is built based on the most efficient designs for each component and we discuss the reasonings in Sections 4.2 and 4.3. The controller is the brain of TL-PIM and orchestrates all necessary operations (Section 4.4). All components are highly efficient regarding their performance, area, and power. We discuss these in detail in Section 6.

4.2 TL-PIM: Matching Mechanism

For matching a query and reference in memory arrays, previous works took three main approaches:

(1) Approach 1:
  (a) Store both (or at least the reference) inside the memory
  (b) Read out the stored values
  (c) Perform an XOR/XNOR using logical gates
  (d) Perform a pop-count to determine the exact match
(2) Approach 2:
  (a) Store both inside the memory
  (b) Read them into a modified SAs/ADCs to perform XOR/XNOR (e.g., Ambit [71] for DRAM, Pinatubo [45] for memristors)
  (c) Perform a pop-count on the output of SAs/ADCs
(3) Approach 3:
  (a) Store both inside the memory
  (b) Perform XOR/XNOR inside the memory using analog computing (e.g., MAGIC [37])
  (c) Read out the result
  (d) Perform a pop-count

Unfortunately, all these techniques have one or more shortcomings in matching two short sub-strings or keys. We provide two examples of the inefficiency of these methods. First, storing queries inside memory incurs unnecessary write operations that take time, waste energy, and hurt endurance. Note that the query keys will change every time one wants to profile a new metagenomic sample exacerbating the problem. Second, the necessary logical gates to perform the pop-count operation after having the results of XOR/XNOR are energy and area inefficient (e.g., a tree of logical OR gates) or time inefficient (e.g., shift registers and counters), depending on the implementation. Note that, as also stated in previous works [32, 72], the additional logic units in peripheries are already responsible for most of the energy and area consumption of the whole design.

We propose a 1-cycle key matching (XNOR and pop-count in one cycle) using memristor devices in a typical memory array structure to mitigate previous shortcomings without introducing a customized memory layout. We exploit the inherent capability of memristor-based memory arrays for performing VMM operations. We also exploit the fact that the XNOR of two keys ($k1 \odot k2$) is functionally equivalent to $k1.k2 + \bar{k1}.\bar{k2}$. Therefore, one can use a column-major data mapping to achieve the intended operations. We call these memory units Key-Arrays hereafter.

[5]The inter-array distribution is defined as when data (keys or LCA labels) is stored in a single memory array. On the other hand, the intra-array distribution is when query- and reference-related data is stored in different/separate memory arrays. The case for near array data distribution happens when either query keys/labels or reference keys/labels are in a separate buffer next to the memory array that stores the other.

The VMM function takes the query key and its complement as the input vector and a matrix of reference keys each placed with their complements in a single column as the input matrix. Fig. 3 demonstrates the placement of input vector and one column of input matrix. Subsequently, by performing VMM operation and reading the results out using an SA that uses a customized reference voltage, one can perform the pop-count of $(k1|\bar{k1}) \,.\, (k2|\bar{k2})$ of multiple reference keys with a query key in one cycle. For this purpose, the reference of SA is set to recognize any current higher than length (key) as logical 1 and lower currents as logical 0. This way, a 1 in the output of an SA shows a match between the query key and the reference key in the same column, and a 0 shows a mismatch.

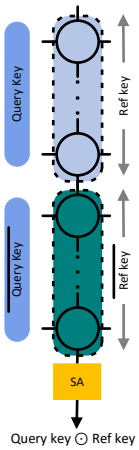

Figure 3: Matcher.

There are three points worth mentioning regarding our proposal. First, this method doubles the required memory cells for reference keys. However, as our evaluations in Section 6 and analysis in Section 3 show, our approach brings even further performance improvement than doubling (1) the whole available bandwidth and (2) compute power in the baselines. Additionally, a quick analysis of Kraken2 shows that reference keys are the smaller part of the hash table.

Second, unlike previous designs that require ADCs for their VMM operation, our method works with SAs. This simplification is because a Key-Array does not need to perform a complete VMM and get the exact number. We are simply interested in whether there is a match at all positions (which produces a current equal to the length of a key). ADCs represent a significant area overhead compared to the memory cells [3, 72]. Therefore, the current design overcomes a critical source of inefficiency if someone replaces this design with those in previous works.

Third, in theory, the same results can be achieved using CAMs (but not TCAMs). However, using CAMs instead of our design have two main limitations TL-PIM aims to prevent: (1) The memory cells in a CAMs are different and fixed and cannot be reconfigured for other types of memory arrays (like those we use for taxonomic retrieval in Section 4.3). In other words, one cannot reuse Key-Arrays for other memory units if they use CAMs for matching. Such a choice limits the design, for example, for a case where one wants to support and load different reference tables; (2) CAMs require more complex controllers due to their different cell designs. Our design does not change the control circuits other than we already have in typical memories.

4.3 TL-PIM: Taxonomic Retrieval

After finding a match between a query key and one of the stored reference keys in Key-Arrays, TL-PIM needs to retrieve the corresponding taxonomic label and send it back to the host CPU. LCA-Arrays are the memory units that store taxonomic labels. A LCA-Array uses row-major data mapping to store the labels. This way, it can retrieve the full label by only reading one row, which is impossible with a column-major mapping. In addition, LCA-Array applies a revised data mapping allowing the design to retrieve the label in 1 cycle. This mapping is due to the limitation of shared SAs among columns in emerging memories. Fig. 4 shows the proposed interleaved, row-major data mapping for each LCA-Array.

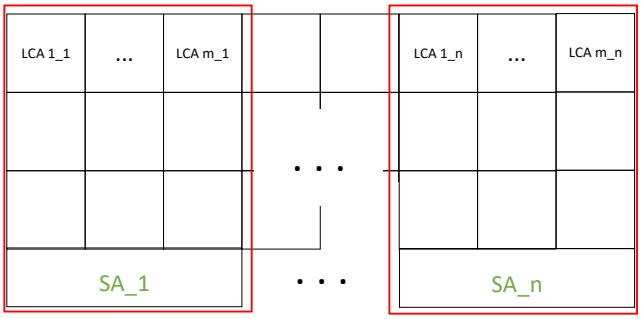

Figure 4: Taxonomic Label Retrieval.

Since SAs are shared among columns of a LCA-Array, to ensure TL-PIM can retrieve all the bits of one label in a single cycle, the LCA-Array needs to distribute the bits of a label and store them among columns that use different SAs. In other words, LCA-Arrays interleave bits of each LCA label among different SAs. *LCA* X_Y in Fig. 4 presents the Y^{th} bit of *LCA X*. For example, assume a 512×512 array where every 16 columns share one SA. Additionally, assume that each taxonomic label has 17 bits. In this case, an LCA-Array puts the 1st bit of Label#1 in column#1, the 2nd bit of Label#1 in column#17 (=16+1), and 17th bit of Label#1 in column#273 (=17*16+1). It does the same for Label#2, i.e., it puts 1st bit of Label#2 in column#2, the 2nd bit of Label#2 in column#18 (=16+2), and so on. This way, for retrieving Label#1, TL-PIM only needs to read out the first bit of the first 17 SAs, which can be achieved in 1 cycle simultaneously.

4.4 TL-PIM: Controller

The Controller unit is the mind behind TL-PIM. The controller first receives the query key as a PCIe packet from the host CPU through the PCIe (Peripheral Component Interconnect Express) [52, 61]. Subsequently, it unpacks the package and distributes the query key to appropriate Key-Arrays. Additionally, the controller sends the proper signals to all memory units (Key-Arrays and LCA-Arrays).

Examples of such signals are those for the VMM operation in Key-Arrays, select signals of MUXes in LCA-Arrays, and decoders' signals. Once TL-PIM compared the query key against all possible reference keys, it sets the PCIe interconnects response ready queue (RRQ). The finished requests will be forwarded to PCIe Out Queue (POQ). Each response can be either a taxonomic label or a NULL, meaning that the query key did not exist in the index table. The controller sends an interrupt signal to the host CPU when a packet is ready in POQ or empty slots in PIQ. The controller is a simple FSM machine and can be easily modified for future taxonomic profilers if the hardware requires the same sets of supported operations by other units.

4.5 Relation between LCA-Arrays and Key-Arrays

The ratio between the number of LCA-Arrays and Key-Arrays is not necessarily 1-to-1. A LCA-Array receives the results of N key-to-key comparisons per cycle per each Key-Array, where N is #SAs per Key-Array. Only one of all these N comparisons can be an exact match (Section 2.2). Therefore, in the worst case, or for the highest performance, TL-PIM needs to be able to retrieve the corresponding taxonomic label for that 1 match out of N possible cases in 1 cycle. This way TL-PIM can overlap retrieval operation of the previous matching with finding the next exact match of the same Key-Array. This scenario may require more than 1 LCA-Array per each Key-Array. In other words, the ratio between the required number of LCA-Arrays per each Key-Array is a design choice and tradeoff between the number of required LCA-Arrays and performance. This design choice also affects the utilization of each LCA-Array as each LCA-Array may end up not using all of its SAs or some columns in each SA (Section 6.2).

The ratio for the highest performance highly depends on the #SAs per Key-Array (or the number of evaluated keys per cycle), length of LCA labels, the capacity of SAs in a LCA-Array, and #SAs per LCA-Array. In other words, it depends not only on the device characteristics of memory arrays and their peripheries but also on the length of values (labels) in the reference table. This is the main reason that having the configurability between Key-Array and LCA-Array is favorable, and we use a typical memory layout for Key-Arrays instead of CAMs (Section 4.2). Fig. 5 presents a case were TL-PIM uses 2 LCA-Arrays per each Key-Array for achieving the highest performance.

Fig. 5 also demonstrates the expected utilization for Key-Arrays and LCA-Arrays by diagonal gray patterns. For minimizing this inefficiency in Key-Arrays, TL-PIM places keys and their complement in each column, i.e., all columns are used. However, depending on the #Rows and required bits per key, TL-PIM can only fill $\lfloor \frac{\#Rows}{bits\ per\ key} \rfloor$. This means that some rows will remain empty (underutilized). For LCA-Arrays, memory utilization is lower, especially when opting for the maximum achievable performance. This means each LCA-Array may have to leave out a few SAs or columns of each SA depending on the number of results produced by corresponding Key-Array in each cycle, size of LCA-Array itself, and #columns that share SAs in the LCA-Array. Each LCA-Array may also have to not use some of its rows depending on expected #keys checked by corresponding Key-Array in total. We evaluate array utilizations in Section 6.2.

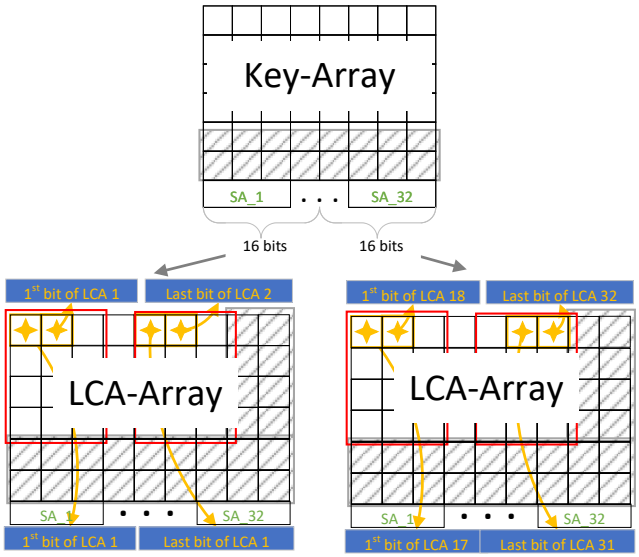

Figure 5: Relation and connection between one Key-Array and multiple LCA-Arrays.

4.6 Optimizations

We apply several design optimizations to further boost the performance and/or energy consumption of our framework that we discuss here.

Optimization 1. We did not add any buffer or additional network among memory arrays in TL-PIM other than those that exist in typical memories. As alluded before (Sections 4.2 and 4.3), each Key-Array produces at most 1 bit consumed by corresponding LCA-Arrays. In addition, not all LCA-Arrays will consume the results of each Key-Array. This means that TL-PIM requires no network among its arrays. This allows TL-PIM to be less constrained regarding the area budget compared to all previous ML-related PIM-enabled architectures using memristor devices [3, 72].

Optimization 2. KrakenOnMem prevents broadcasting each query key to all Key-Arrays, and subsequently, it decreases the data movement in TL-PIM significantly. To this end, KrakenOnMem sorts the required hash table based on the keys alphanumerically and then stores them in Key-Arrays of TL-PIM according. Similar to Sieve [87], KrakenOnMem stores an 8-byte ID consisting of first and last reference keys in an array. Such a table will remain under 2 MB for a 500 GB reference table, and the host CPU or controller can easily store it. For each query key, KrakenOnMem first consults this table to find the correct Key-Array to send the request to in TL-PIM. This mechanism scales linearly with the size of memory considered for KrakenOnMem (equivalently the size of the hash table for references) rather than the length of the considered keys.

Optimization 3. We clock gate the LCA-Arrays that have no potential match for a particular query key. This is possible since we only get at most one output "1" from all Key-Arrays in each cycle because every query key can match only with one reference key

by definition of the key in a hash table. This optimization saves the static energy of our system.

4.7 KrakenOnMem Profiling Walk Through

KrakenOnMem performs an accurate and high-performance taxonomic profiling. In the boot up, the host CPU of KrakenOnMem loads the reference indexes (hash table) into TL-PIM's memory units based their required data mapping (Sections 4.2, 4.3). This is a one-time job, and we do not need to repeat it unless one changes the reference database that rarely happens. Fig. 6 summarizes how KrakenOnMem translates an existing Kraken2's database into appropriate data mapping in Key-Arrays and LCA-Arrays.

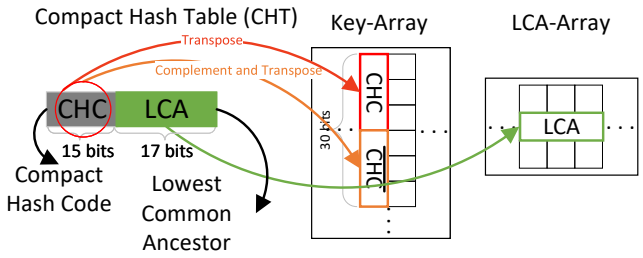

Figure 6: Mapping of Kraken2's Hash Table into TL-PIM for KrakenOnMem.

At the recipient of a metagenomics sample, the host CPU in KrakenOnMem reads queries from Fastq files, extracts the k-mers, and calculates the corresponding keys. It then transfers the results to TL-PIM as queries. TL-PIM connects to the CPU host using PCIe [52, 61]. KrakenOnMem uses this connection to (1) send TL-PIM the necessary signals and query keys and (2) receive taxonomic labels from TL-PIM. KrakenOnMem hides the transfer latency of PCIe between host CPU and TL-PIM using the double-buffering technique [53]. TL-PIM unpacks the PCIe packets it gets from the PCIe input queue and distributes the keys to possible target Key-Arrays. After retrieving a taxonomic label, TL-PIM creates a response packet and stores it in PCIe's RRQ. A batch of these packets will be sent to POQ and eventually to the host CPU. The host CPU reads the response and aggregates them to achieve the final profiling of the read a key belongs to.

5 EVALUATION METHODOLOGY

We build KrakenOnMem based on Kraken2 to perform end-to-end taxonomic profiling. TL-PIM replaces the Kraken2's *Table Lookup* operation, which accelerates this significant bottleneck. We verify KrakenOnMem architecture using a cycle-accurate RTL model of the complete CMOS design with equivalent throughput based on the architecture in previous works [91, 92]. The memory model is validated and based on a small 4Gbit STT-MRAM chip prototype in TSMC 28nm CMOS technology [36, 64]. We use an analytical model based on this small prototype and extend the memory to the required size. The model is acquired from the results of the EU project MNEMOSENE [58], led and concluded by TU Delft in 2020. In other words, without access to large-scale production-level memristive devices, we evaluated our design using the next best approach: We implemented and prototyped many parts in FPGA and connected them to existing (small) memories and a high-performance AFE (analog -front-end) board for DACs, power supplies, and voltage and current references, to faithfully build first a small scale (using real size memory) and then an analytical model for our evaluation. We run all of our software experiments on a 128-core server running on AMD EPYC 7742 processors that operate at 2.25 GHz. We have 512 GB of 3200 MHz DDR4 DRAM available on this server.

Baselines. We compare KrakenOnMem mainly with Kraken2, the state-of-the-art taxonomic profiler. Kraken2 accompanied by Bracken [51] is one of the promising approaches for taxonomic profiling on different datasets once based on the latest results of the CAMI challenge [56]. We analyze the accuracy of KrakenOnMem by comparing its output results with only the profiling outputs of Kraken2. We also compare our platform with a state-of-the-art in-memory k-mer matcher, Sieve [87]. We consider Sieve only for the k-mer matching and query retrieval part (*Table Lookup*) and assume that the CPU takes care of the rest for a taxonomic profiling, similar to our platform. Finally, we compare the proposal with Metalign, an alignment-based taxonomic profiler. This study demonstrates the potential of KrakenOnMem in general or TL-PIM in particular in other and future taxonomic profilers that are not bottlenecked by *Table Lookup* but still suffer from a similar inefficient operation.

Note that our evaluations do not include a GPU-based baseline for three reasons. First, currently, there is no GPU-based taxonomic profiler for metagenomics, let alone a GPU-based profiler based on Kraken2. Second, Sieve already outperforms GPUs in simple k-mer matching operation regarding performance and energy consumption. Third, GPUs are power-hungry and require significant data movement for reference index tables making them less likely to be adopted by the experts in the near future.

Performance Model. Kraken2 and Metalign open-sourced implementations report performance (execution time) directly when run on our servers. We use statistics of our synthesized design using TSMC 28nm technology node in Synopsys Design Compiler [74] to obtain the latency of main hardware components of KrakenOnMem, namely TL-PIM. We obtain the execution time of other steps by running each necessary steps of the Kraken2 on the host CPU. We take a similar approach for estimating Sieve's performance, considering improvements stated in the original paper for the matching.

Area and Power. Similar to performance, we also acquire the area and power consumption of KrakenOnMem's components from our synthesized design and memory model. This design considers a typical operation condition of temperature 25°and voltage 1.2V for power consumption evaluations. We measure the power consumed by our CPU host using Intel's PCM power utility [30].

Datasets. We use DustMasked MiniKraken for our reference database when testing our small prototype. This is a pre-built 4GB database from dustmasked bacterial, archaeal, and viral genomes in Refseq. For all other experiments, we used the default reference database of Kraken2 and Metalign. For query sequences, we use 3 datasets from the CAMI challenge: CAMI-low (RL), CAMI-medium (RM), and CAMI-high (RH).

6 EXPERIMENTAL RESULTS

KrakenOnMem produces the same list of matches, LCAs, and ultimately taxonomic profiling results as the original Kraken2. This

was expected since KrakenOnMem does not change the order of steps in Kraken2 but it accelerates the bottleneck. Note that the order or rate at which we perform the matching and retrieval does not affect the profiling results as long as we ensure we have all the results before the final aggregation step. The same holds for the accuracy results of Sieve. From the results of the latest CAMI challenge [56], we know that Kraken2+Bracken stands among the high accuracy taxonomy profilers. Therefore, we conclude that KrakenOnMem also has high accuracy.

6.1 Performance Analysis

Our performance analysis consists of two separate sets of experiments. In the first set, we compare the performance of TL-PIM for *Table Lookup* with that in Sieve and Kraken2. Subsequently, we evaluate the end-to-end effect of our accelerator when employed for taxonomic profiling. In the second set, we slightly modify TL-PIM to be used for counting the matched k-mers instead of *Table Lookup*. Afterward, we compare this design with KMC3 [35] used in Metalign. Finally, we evaluate the end-to-end effect of using such a design in Metalign.

***Table Lookup* and Heuristic-based Profiling.** Fig. 7 depicts the performance of Kraken2, Sieve, and TL-PIM when performing *Table Lookup*. The y-axis utilizes a logarithmic scale.

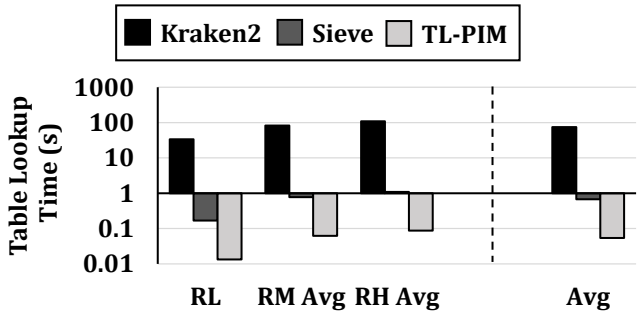

Figure 7: Performance comparison for *Table Lookup* between Kraken2, Sieve, and TL-PIM.

We make two observations. First, TL-PIM provides, on average, 1386× performance improvement over the original *Table Lookup* in Kraken2 and 111× performance improvement over Sieve for the same operation. Second, TL-PIM performs *Table Lookup* faster than Sieve and Kraken2 regardless of the dataset, on all three CAMI datasets by at least 100× and 1250×, respectively. TL-PIM outperforms original *Table Lookup* in Kraken2 due to its minimum data movement, high parallelism, and 1 cycle operation. It also outperforms Sieve due to the inefficiencies that Sieve introduces, such as duplicates and heavy internal data movement (Section 3.2). These results also show that ETM in Sieve, while effective, does not help Sieve outperform TL-PIM, even for queries with different complexity, such as those we used.

Fig. 8 presents end-to-end performance of taxonomic profiling for original Kraken2, Sieve, and KrakenOnMem. We observe that KrakenOnMem provides (1) 61.3% performance improvement over Kraken2, and (2) 1.17% performance improvement over Sieve. As expected, this is lower than improvements considering only *Table Lookup* because although this operation is the bottleneck in taxonomic profiling, KrakenOnMem still incurs some pre-processing and post-processing that become the new bottlenecks, reducing the overall benefit to some extent. In other words, we have diminishing returns due to the sequential nature of other sections of our application (i.e., Amdahl's Law).

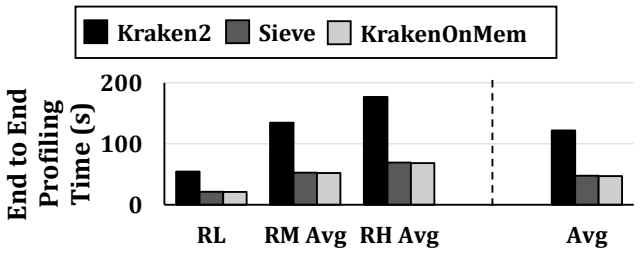

Figure 8: Performance comparison for profiling between Kraken2, Sieve, and KrakenOnMem.

Note that improvements of KrakenOnMem over Sieve are still significant for four reasons. First, although the overall speedup of KrakenOnMem compared to Sieve is small, the accelerated operation (*Table Lookup*) achieves a significant speedup. This opens up the possibility of future work focusing on the next performance bottleneck - a common (iterative) methodology in computer engineering. Second, *Table Lookup* will likely remain in future genomics pipelines for which the sequential part of the application can be different. Third, as we will discuss in Section 6.2, KrakenOnMem is also advantageous over Sieve in other aspects, e.g., area overhead and energy consumption. Fourth, the sped-up operation can and will potentially be used in all the upcoming metagenomics studies and profilers. Therefore, a modest 1.17% improvement can still translate into significant time and cost gains.

k-mer Counting and Alignmnet-based Profiling. Fig. 9 presents the results for pre-filtering stage in Metalign when tweaking TL-PIM to perform k-mer counting compared to the original KMC3 implementation in Metalign. The y-axis utilizes a logarithmic scale.

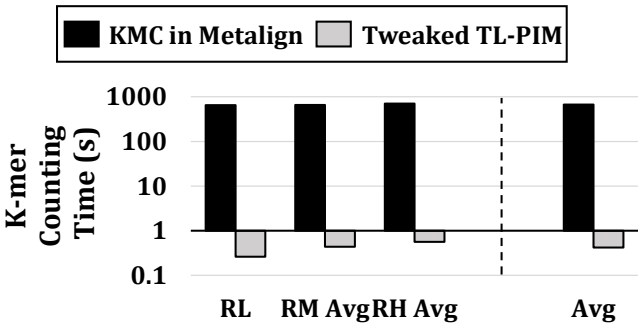

Figure 9: Performance comparison for k-mer counting between KMC and tweaked TL-PIM.

We observe that for all CAMI datasets, the PIM-enabled design provides on average a 1595× improvement compared to an SW

version of the same operation in Metalign. We expected such significant improvements due to advances in data movement reduction and high parallelism in TL-PIM.

Fig. 10 demonstrates the aftermath when we apply the new TL-PIM for end-to-end taxonomic profiling using the Metalign approach as a SOTA alignment-based profiler. We make two observations. First, our proposal provides, on average, 23.01% improvement in execution time for end-to-end profiling. This is expected as the k-mer counting operation in Metalign still affects the overall performance and cannot be masked or parallelized by other operations and steps.

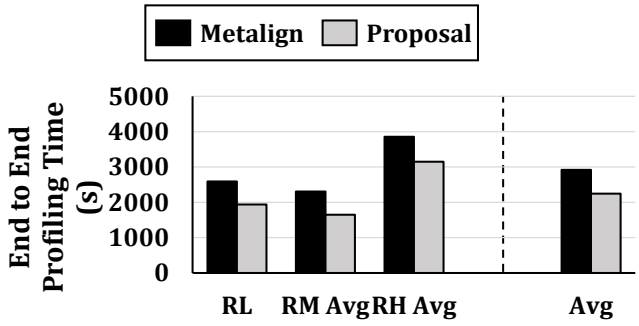

Figure 10: Performance comparison for profiling between original Metalign and Metalign equipped with tweaked TL-PIM.

Second, the end-to-end improvement in an alignment-based mechanism is much less than the achieved improvement over Kraken2 (23.01% vs. 61.3%). This happens because alignment-based taxonomic profilers are not bottlenecked by the *Table Lookup* operation, rather their required alignment operation. However, we argue that this improvement is still significant and shows that the *Table Lookup* procedure is costly for any type of profiler and worth the design, even if it is not the bottleneck.

6.2 Power and Area Analysis

Energy and Power. We compare the expected energy consumption of Sieve with the measured energy consumption of KrakenOnMem for *Table Lookup* operation. We consider the DRAM technology mentioned in the original manuscript for Sieve. We exploit three synthetic datasets to cover all spectrum of possible scenarios for Sieve, as its energy highly depends on the effect of the data-dependent ETM component. **Sc1** is when more than 95% of the mismatch/difference between query and reference keys exists in the first two characters of them. This scenario favors Sieve the most as the ETM component can save Sieve a lot of row activations and unnecessary comparisons. **Sc2** is the case where the average distance of the first mismatch among query and reference keys from their first bit is set to 10 bits. This is the main reported number in Sieve [87] for typical cases. The last scenario, **Sc3**, contains the results for when the mismatches cannot be found until the last two characters of query and reference keys. This is the worst-case scenario for Sieve, in which ETM favors the performance and energy the least. Fig. 11 presents the results.

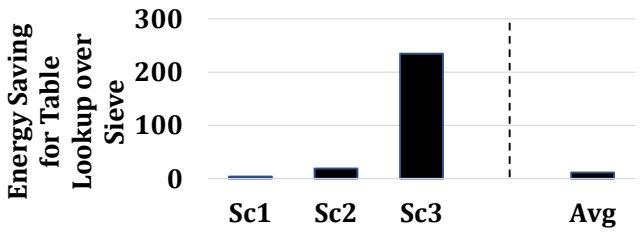

Figure 11: Energy saving of KrakenOnMem over Sieve for *Table Lookup*.

We make two observations. First, KrakenOnMem achieves a better energy consumption for all three scenarios, with an average of 11.34× energy saving over Sieve. One cycle matching and label retrieval, pipelined design, available parallelism, and analog computing are the reasons behind this significant improvement in energy consumption of *Table Lookup* in KrakenOnMem over Sieve. Second, Sieve offers a very data-dependent energy consumption that changes to an almost 62× depending on where the first mismatch among keys occurs. This shows that while ETM in Sieve might be compelling enough in average cases, it cannot be reliably used for a fixed/calculated energy consumption. KrakenOnMem solves this issue.

Our power evaluations show that KrakenOnMem can profile with a rate of $4.75\frac{Mbp}{j}$. This is while Merelli, et al. [14, 55] show that original Kraken2 on a 8-core XeonD processor can achieve only a maximum of $0.22\frac{Mbp}{j}$. Note that we have not directly compared the energy consumption of KrakenOnMem with Kraken2 for two reasons. First, our expertise in memory accelerators allows us to measure the expected profiling rate for KrakenOnMem, but it would not be a fair comparison with a software-only solution. Second, only a few works [14, 55] provide energy numbers, and we utilized them for our rough energy comparisons with KrakenOnMem. These results showcase that KrakenOnMem consumes much lower energy for the same datasets than Kraken2.

Area. Although TL-PIM consists of memory arrays and controller logic, we only consider the area of its controller as additional area overhead. This is because sequencing machines are heterogeneous systems that already use various memory technologies (e.g., DRAM and SSD) and computational units (CPU, GPU, and FPGA) since their benefits justify their cost. Therefore, having memristors installed in those machines as well is not a far-fetched idea if we can harvest their power efficiently. The area overhead for the required controller in TL-PIM is 0.009 mm^2. This is a very modest overhead, only 0.002% of Skylake-SP, a modern Intel Processor at 14 nm [29]. Although any processor can be chosen for this comparison, depending on the final product, we pick the Skylake-SP processor for the area baseline only to be similar to works on PIM-based accelerators and non-volatile memories [8, 62]. Note that Sieve Type III, which we used in our performance and energy evaluations, incurs 10.9% area overhead for the required logic of (1) k-mer matching and (2) row-address latches over an 8-bank DRAM chip in 22nm technology mode. We use methods presented in [73] to scale the area consumption down to 28nm technology mode and find an overhead

of higher than 6%. On the other hand, KrakenOnMem only incurs 0.0007% extra die area for its controller (as LCA- and Key-arrays exist regardless) over the same memory size using our STT-MRAM devices.

Memory utilization. Fig. 12 and Fig. 13 depict the memory utilization for Key-Arrays and LCA-Arrays, respectively, considering different array configurations, i.e., varying size and #SAs.

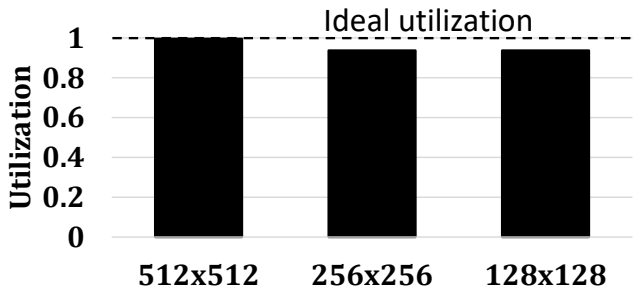

Figure 12: The memory utilization of Key-Arrays.

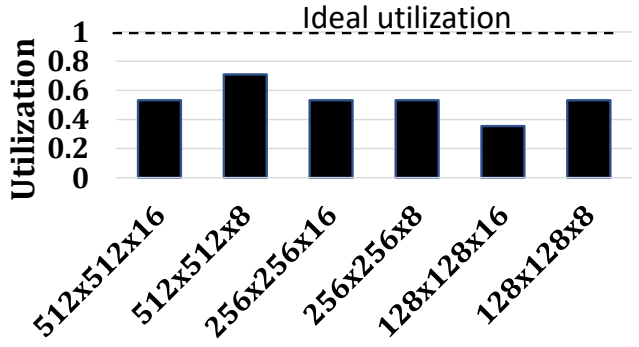

Figure 13: The memory utilization of LCA-Array.

We make three observations. First, TL-PIM utilizes Key-Arrays close to the ideal case. For three standard evaluated array sizes of 512×512, 256×256, and 128×128, TL-PIM achieves a utilization higher than 93%. Second, utilization of LCA-Arrays is lower than that in Key-Arrays. However, they are still on average higher than 50%. This is expected in any accelerator, including TL-PIM, that does not change the memory structure or data representation yet aims for the highest achievable performance. Third, utilization of Key-Arrays only depends on the array size, while LCA-Arrays' utilization is also affected by #columns that share one SA. The reason behind this is that the new parameters affect the number of LCA-Arrays per one Key-Array, as discussed in Section 4.5.

7 DISCUSSIONS AND FUTURE WORKS

Higher Memory Utilization. KrakenOnMem sacrifices memory utilization for performance, especially in LCA-Arrays. To ensure that KrakenOnMem can perform the *Table Lookup* operation in 1 cycle for the worst-case scenario, it has to leave some cells of the LCA-Arrays unused. However, it is possible to reach a higher utilization without a considerable performance loss. Currently, KrakenOnMem opts for this solution to provide the maximum performance one can expect. Different data layouts for LCA-Arrays can be investigated as future works for applying the same idea to other profilers or applications.

Optimization Possibilities for Rank-Level Profiling. KrakenOnMem can potentially improve profiling performance at different taxonomic ranks as well. An example of such optimizations can be the placement of keys related to a particular species on the columns of a Key-Array that do not share their SAs. This way, one can investigate the match of the query key to that species in 1 cycle. However, we leave the investigation of such optimizations for future work.

More Application Support. KrakenOnMem focuses on taxonomic profiling due to its importance and promise of being adopted with the newest technologies rather than the importance of general *Table Lookup* itself. However, TL-PIM from KrakenOnMem can also be used in any other application bottlenecked by similar large (hash) tables. We leave the exploration of such applications and benefits TL-PIM and a similar HW/SW co-design to KrakenOnMem can provide to these applications for future work. The effectiveness, however, depends on how much of the issue this operation is in the original problem.

Support for TL-PIM Data Mapping. Currently, KrakenOnMem does not have API support for the required data mapping of flexible datasets and tables. We avoid virtual memory translation by mapping the KrakenOnMem's memory space directly to the CPU host. Therefore, currently, KrakenOnMem loads the required hash table to the TL-PIM memory arrays based on prior knowledge about the (hash) table in Kraken2. We leave building an API to support all data mapping required for an efficient *Table Lookup* for future work. Note that loading the hash table is a one-time (rare) task before starting the query operation. Since these hash tables rarely change and we usually reuse genomics databases, the cost of our approach is still acceptable for the intended long usage period.

Building Reference Table. Currently, KrakenOnMem does not build the required (hash) table or the taxonomy tree. Therefore, the current design always requires the (hash) table construction to be done first on another platform, which can be the primary host (CPU) used also in KrakenOnMem. However, this task is a one-time job and does not diminish the benefits of KrakenOnMem.

8 RELATED WORKS

To our knowledge KrakenOnMem is the first work to present an HW/SW co-designed framework for taxonomy profiling of genomics samples exploiting the PIM paradigm. KrakenOnMem accelerates the bottleneck of a SOTA profiler, *Table Lookup*, using TL-PIM, a PIM-enabled accelerator. We have compared KrakenOnMem extensively to previous SOTA taxonomic profilers, both heuristic-based and alignment-based and, in Section 6. This section briefly discusses previous works on taxonomy profilers and (PIM-enabled) genomics accelerators.

Metagenomic Profilers. SOTA taxonomic profilers take one or a combination of three following directions to improve the accuracy and/or execution time of profiling: (1) Reference database's size

reduction with pre-alignment filters [2, 90] or heuristics for taxonomic classification [34, 40, 46, 70, 85]. (2) Post species-level classification presence and abundance estimation heuristics [40, 41, 51]. (3) Hardware acceleration for an algorithm or function that can later be used for metagenomic studies (e.g., alignment step in an alignment-based profiler) [4, 12, 16, 21, 25, 47, 48, 65, 77, 81].

KrakenOnMem is orthogonal to the works in the first two groups. KrakenOnMem is loosely an example of works discussed in the last group. However, previous works focused on alignment and not *Table Lookup* due to the importance of alignment in other genomics pipelines. They also do not scale well with the current and rapidly growing datasets metagenomics deals with. Therefore, we believe KrakenOnMem is more suitable for future taxonomic profilers, as it is compared to existing designs.

(PIM-based) Genomics Accelerators. Many recent works explored different architectures for genomics-related kernels or full application. GenASM [8] proposes a framework for approximate string matching (ASM) and employs it in multiple genome-related analyses. DARWIN [76] is a co-processor hardware accelerator for sequence alignment. DARWIN uses a filtering algorithm (D-SOFT) and a new algorithm to perform the long sequence alignment using constant memory. RADAR [79] exploits 3D ReRAM and accelerates multiple computational units of BLASTN by eliminating unnecessary data movement and performing seed-and-extend algorithm and mapping more efficiently. AligneR [93] and RASSA [31] are two other two PIM-enabled accelerators for short and long-read alignment, respectively. Helix [49] and BRAWL [50] accelerate basecalling on a NVM-based PIM-enabled architecture. Laguna et al. [38] propose an in-memory architecture using TCAMs for seed-and-vote read mapping. MEDAL [26] proposes a PIM-enabled accelerator for seeding on DIMM between DRAM modules. MEDAL utilizes the memory bandwidth very efficiently and has fine-grained memory accessibility. NEST [27] is a DIMM-based PIM-enabled architecture for k-mer counting.

KrakenOnMem differs from all these works as it uses hardware acceleration for *Table Lookup*, a critical function in all SOTA taxonomic profilers. KrakenOnMem is, to the best of our knowledge, the only end-to-end hardware-accelerated framework for taxonomy profiling. Moreover, TL-PIM in KrakenOnMem also prevents unnecessary data movement using a PIM-enabled design, which was not the main focus of many previous works.

9 CONCLUSION

This paper introduces KrakenOnMem, the first HW/SW co-designed framework for taxonomic profiling via in-memory hardware acceleration using emerging memory technologies. KrakenOnMem accelerates the bottleneck of Kraken2, a SOTA taxonomic profiler, by a memristor-based PIM-enabled hardware called TL-PIM. TL-PIM enables *Table Lookup* operation while it simultaneously aims for (1) being data-independent, (2) maximizing the achievable performance for the worst-case scenarios, (3) having linear scalability, (4) maintaining its design optimization advantages for future designs, and (5) incurring minimal hardware and area overhead. The evaluation results show that TL-PIM alleviates the existing bottleneck to the extent that the end-to-end performance and energy consumption of Kraken2 significantly surpasses that of original Kraken2. We expect that ideas presented for TL-PIM and the HW/SW co-designed of Kraken2 enable the designs of future accelerators in other genomic applications. We also expect that the overall improvement in end-to-end taxonomic profiling introduced by KrakenOnMem further helps the upcoming metagenomic studies and opens new doors for improving our lives.

ACKNOWLEDGMENTS

We thank the anonymous reviewers of ICS 2022 for feedback. We thank the CE group members at the QCE department in TU Delft for feedback and the stimulating intellectual environment.

REFERENCES

[1] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute Caches," in *HPCA*, 2017.
[2] M. Alser, T. Shahroodi, J. Gomez-Luna, C. Alkan, and O. Mutlu, "SneakySnake: A Fast and Accurate Universal Genome Pre-Alignment Filter for CPUs, GPUs, and FPGAs," *Bioinformatics*, 2020.
[3] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-m. W. Hwu, J. P. Strachan, K. Roy *et al.*, "PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *ASPLOS*, 2019.
[4] S. S. Banerjee, M. El-Hadedy, J. B. Lim, Z. T. Kalbarczyk, D. Chen, S. S. Lumetta, and R. K. Iyer, "Asap: Accelerated short-read alignment on programmable hardware," *IEEE Transactions on Computers*, 2018.
[5] Barba, M, Czosnek, H and Hadidi, A, "Cost in US Dollars per Raw Megabase of DNA Sequence," https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data.
[6] A. Brady and S. Salzberg, "PhymmBL expanded: confidence scores, custom databases, parallelization and more," *Nature methods*, 2011.
[7] F. P. Breitwieser, J. Lu, and S. L. Salzberg, "A Review of Methods and Databases for Metagenomic Classification and Assembly," *Briefings in bioinformatics*, 2019.
[8] D. S. Cali, G. S. Kalsi, Z. Bingöl, C. Firtina, L. Subramanian, J. S. Kim, R. Ausavarungnirun, M. Alser, J. Gomez-Luna, A. Boroumand *et al.*, "GenASM: A high-performance, low-power approximate string matching acceleration framework for genome sequence analysis," in *MICRO*, 2020.
[9] E. Chen, D. Apalkov, Z. Diao, A. Driskill-Smith, D. Druist, D. Lottis, V. Nikitin, X. Tang, S. Watts, S. Wang *et al.*, "Advances and future prospects of spin-transfer torque random access memory," *IEEE Transactions on Magnetics*, 2010.
[10] L. Cheng, Y. Li, K.-S. Yin, S.-Y. Hu, Y.-T. Su, M.-M. Jin, Z.-R. Wang, T.-C. Chang, and X.-S. Miao, "Functional demonstration of a memristive arithmetic logic unit (memalu) for in-memory computing," *Advanced Functional Materials*, 2019.
[11] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ISCA*, 2016.
[12] J. Daily, "Parasail: SIMD C Library for Global, Semi-global, and Local Pairwise Sequence Alignments," *BMC bioinformatics*, 2016.
[13] J. Dröge, I. Gregor, and A. C. McHardy, "Taxator-tk: precise taxonomic assignment of metagenomes by fast approximation of evolutionary neighborhoods," *Bioinformatics*, 2015.
[14] D. D'Agostino, L. Morganti, E. Corni, D. Cesini, and I. Merelli, "Combining edge and cloud computing for low-power, cost-effective metagenomics analysis," *Future Generation Computer Systems*, 2019.
[15] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *ISCA*, 2018.
[16] X. Fei, Z. Dan, L. Lina, M. Xin, and Z. Chunlei, "Fpgasw: Accelerating large-scale smith–waterman sequence alignment application with backtracking on fpga linear systolic array," *Interdisciplinary Sciences: Computational Life Sciences*, 2018.
[17] J. D. Ferreira, G. Falcao, J. Gómez-Luna, M. Alser, L. Orosa, M. Sadrosadati, J. S. Kim, G. F. Oliveira, T. Shahroodi, A. Nori *et al.*, "pLUTo: In-DRAM Lookup Tables to Enable Massively Parallel General-Purpose Computation," *arXiv preprint*, 2021.
[18] B. Fjukstad and L. A. Bongo, "A review of scalable bioinformatics pipelines," *Data Science and Engineering*, 2017.
[19] D. Fujiki, S. Mahlke, and R. Das, "Duality cache for data parallel acceleration," in *ISCA*, 2019.
[20] G. V. RESEARCH, "Metagenomics market size, share and trends analysis report by product (sequencing and data analytics), by technology (sequencing, function), by application (environmental), and segment forecasts, 2018 - 2025." 2017.
[21] E. Georganas, A. Buluç, J. Chapman, L. Oliker, D. Rokhsar, and K. Yelick, "merAligner: A fully parallel sequence aligner," in *IPDPS*, 2015.
[22] I. Gregor, J. Dröge, M. Schirmer, C. Quince, and A. C. McHardy, "PhyloPythiaS+: a self-training method for the rapid reconstruction of low-ranking taxonomic

bins from metagenomes," *PeerJ*, 2016.
[23] S. Hamdioui, L. Xie, H. A. Du Nguyen, M. Taouil, K. Bertels, H. Corporaal, H. Jiao, F. Catthoor, D. Wouters, L. Eike *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *DATE*, 2015.
[24] J. Handelsman, M. R. Rondon, S. F. Brady, J. Clardy, and R. M. Goodman, "Molecular biological access to the chemistry of unknown soil microbes: a new frontier for natural products," *Chemistry & biology*, 1998.
[25] E. J. Houtgast, V.-M. Sima, K. Bertels, and Z. Al-Ars, "An FPGA-based systolic array to accelerate the BWA-MEM genomic mapping algorithm," in *SAMOS*, 2015.
[26] W. Huangfu, X. Li, S. Li, X. Hu, P. Gu, and Y. Xie, "Medal: Scalable dimm based near data processing accelerator for dna seeding algorithm," in *MICRO*, 2019.
[27] W. Huangfu, K. T. Malladi, S. Li, P. Gu, and Y. Xie, "NEST: DIMM based near-data-processing accelerator for K-mer counting," in *ICCAD*, 2020.
[28] D. H. Huson, S. Beier, I. Flade, A. Górska, M. El-Hadidi, S. Mitra, H.-J. Ruscheweyh, and R. Tappu, "MEGAN community edition-interactive exploration and analysis of large-scale microbiome sequencing data," *PLoS computational biology*, 2016.
[29] Ian Curtis., "The Intel Skylake-X Review: Core i9 7900X, i7 7820X and i7 7800X Tested: Die Size Estimates and Arrangements." https://www.anandtech.com/show/11550/the-intel-skylakex-review-core-i9-7900x-i7-7820x-and-i7-7800x-tested/6, 2017.
[30] Intel Corp., "Intel® Performance Counter Monitor." https://www.intel.com/software/pcm, 2017.
[31] R. Kaplan, L. Yavits, and R. Ginosar, "RASSA: resistive prealignment accelerator for approximate DNA long read mapping," *IEEE Micro*, 2018.
[32] G. Karunaratne, M. Le Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, "In-memory hyperdimensional computing," *Nature Electronics*, 2020.
[33] R. Kobus, J. M. Abuín, A. Müller, S. L. Hellmann, J. C. Pichel, T. F. Pena, A. Hildebrandt, T. Hankeln, and B. Schmidt, "A big data approach to metagenomics for all-food-sequencing," *BMC bioinformatics*, 2020.
[34] R. Kobus, C. Hundt, A. Müller, and B. Schmidt, "Accelerating metagenomic read classification on CUDA-enabled GPUs," *BMC bioinformatics*, 2017.
[35] M. Kokot, M. Długosz, and S. Deorowicz, "KMC 3: counting and manipulating k-mer statistics," *Bioinformatics*, 2017.
[36] M. Komalan, S. Sakhare, T. H. Bao, S. Rao, W. Kim, C. Tenllado, J. I. Gómez, G. S. Kar, A. Furnemont, and F. Catthoor, "Cross-layer design and analysis of a low power, high density STT-MRAM for embedded systems," in *ISCAS*, 2017.
[37] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MAGIC—Memristor-aided logic," *TCAS II*, 2014.
[38] A. F. Laguna, H. Gamaarachchi, X. Yin, M. Niemier, S. Parameswaran, and X. S. Hu, "Seed-and-vote based in-memory accelerator for dna read mapping," in *ICCAD*, 2020.
[39] B. Langmead, C. Wilks, V. Antonescu, and R. Charles, "Scaling read aligners to hundreds of threads on general-purpose processors," *Bioinformatics*, 2019.
[40] N. LaPierre, M. Alser, E. Eskin, D. Koslicki, and S. Mangul, "Metalign: Efficient Alignment-Based Metagenomic Profiling via Containment Min Hash," *BioRxiv*, 2020.
[41] N. LaPierre, S. Mangul, M. Alser, I. Mandric, N. C. Wu, D. Koslicki, and E. Eskin, "MiCoP: Microbial Community Profiling Method for Detecting Viral and Fungal Organisms in Metagenomic Samples," *BMC genomics*, 2019.
[42] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Phase change memory architecture and the quest for scalability," *Communications of the ACM*, 2010.
[43] J. Leipzig, "A review of bioinformatic pipeline frameworks," *Briefings in bioinformatics*, 2017.
[44] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," in *MICRO*, 2017.
[45] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *DAC*, 2016.
[46] B. Liu, T. Gibbons, M. Ghodsi, T. Treangen, and M. Pop, "Accurate and fast estimation of taxonomic profiles from metagenomic shotgun sequences," *Genome biology*, 2011.
[47] Y. Liu and B. Schmidt, "GSWABE: faster GPU-accelerated sequence alignment with optimal alignment retrieval for short DNA sequences," *Concurrency and Computation: Practice and Experience*, 2015.
[48] Y. Liu, A. Wirawan, and B. Schmidt, "CUDASW++ 3.0: Accelerating Smith-Waterman Protein Database Search by Coupling CPU and GPU SIMD Instructions," *BMC bioinformatics*, 2013.
[49] Q. Lou, S. C. Janga, and L. Jiang, "Helix: Algorithm/Architecture Co-design for Accelerating Nanopore Genome Base-calling," in *PCAT*, 2020.
[50] Q. Lou and L. Jiang, "BRAWL: A Spintronics-Based Portable Basecalling-in-Memory Architecture for Nanopore Genome Sequencing," *IEEE Computer Architecture Letters*, 2018.
[51] J. Lu, F. P. Breitwieser, P. Thielen, and S. L. Salzberg, "Bracken: Estimating Species Abundance in Metagenomics Data," *PeerJ Computer Science*, 2017.
[52] D. Mayhew and V. Krishnan, "PCI Express and Advanced Switching: Evolutionary path to building next generation interconnects," in *11th Symposium on High Performance Interconnects*, 2003.
[53] D. Mayhew and V. Krishnan, "Pci express and advanced switching: Evolutionary path to building next generation interconnects," in *HOTI*, 2003.
[54] A. B. McIntyre, R. Ounit, E. Afshinnekoo, R. J. Prill, E. Hénaff, N. Alexander, S. S. Minot, D. Danko, J. Foox, S. Ahsanuddin *et al.*, "Comprehensive benchmarking and ensemble approaches for metagenomic classifiers," *Genome biology*, 2017.
[55] I. Merelli, L. Morganti, E. Corni, C. Pellegrino, D. Cesini, L. Roverelli, G. Zereik, and D. D'Agostino, "Low-power portable devices for metagenomics analysis: Fog computing makes bioinformatics ready for the Internet of Things," *Future Generation Computer Systems*, 2018.
[56] F. Meyer, A. Fritz, Z.-L. Deng, D. Koslicki, A. Gurevich, G. Robertson, M. Alser, D. Antipov, F. Beghini, D. Bertrand *et al.*, "Critical Assessment of Metagenome Interpretation-the second round of challenges," *bioRxiv*, 2021.
[57] F. Meyer, S. Bagchi, S. Chaterji, W. Gerlach, A. Grama, T. Harrison, T. Paczian, W. L. Trimble, and A. Wilke, "MG-RAST version 4—lessons learned from a decade of low-budget ultra-high-throughput metagenome analysis," *Briefings in bioinformatics*, 2019.
[58] MNEMOSENE partners, "The MNEMOSENE project." http://www.mnemosene.eu/, 2020.
[59] H. A. D. Nguyen, J. Yu, M. A. Lebdeh, M. Taouil, S. Hamdioui, and F. Catthoor, "A classification of memory-centric computing," *JETC*, 2020.
[60] R. Ounit, S. Wanamaker, T. J. Close, and S. Lonardi, "CLARK: fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers," *BMC genomics*, 2015.
[61] PCI-SIG, "PCI-E Specification." https://pcisig.com/specifications.
[62] L. Pentecost, A. Hankin, M. Donato, M. Hempstead, G.-Y. Wei, and D. Brooks, "NVMExplorer: A Framework for Cross-Stack Comparisons of Embedded Non-Volatile Memories," *HPCA*, 2021.
[63] A. E. Pérez-Cobas, L. Gomez-Valero, and C. Buchrieser, "Metagenomic approaches in microbial ecology: an update on whole-genome and marker gene sequencing analyses," *Microbial Genomics*, 2020.
[64] S. Rao, W. Kim, S. van Beek, S. Kundu, M. Perumkunnil, S. Cosemans, F. Yasin, S. Couet, R. Carpenter, B. O'Sullivan *et al.*, "STT-MRAM array performance improvement through optimization of Ion Beam Etch and MTJ for Last-Level Cache application," in *IMW*, 2021.
[65] G. Rizk and D. Lavenier, "GASSST: global alignment short sequence search tool," *Bioinformatics*, 2010.
[66] M. Roberts, W. Hayes, B. R. Hunt, S. M. Mount, and J. A. Yorke, "Reducing storage requirements for biological sequence comparison," *Bioinformatics*, 2004.
[67] M. R. Rondon, P. R. August, A. D. Bettermann, S. F. Brady, T. H. Grossman, M. R. Liles, K. A. Loiacono, B. A. Lynch, I. A. MacNeil, C. Minor *et al.*, "Cloning the soil metagenome: a strategy for accessing the genetic and functional diversity of uncultured microorganisms," *Appl. Environ. Microbiol.*, 2000.
[68] A. Sczyrba, P. Hofmann, P. Belmann, D. Koslicki, S. Janssen, J. Dröge, I. Gregor, S. Majda, J. Fiedler, E. Dahms *et al.*, "Critical assessment of metagenome interpretation—a benchmark of metagenomics software," *Nature methods*, 2017.
[69] N. Segata, L. Waldron, A. Ballarini, V. Narasimhan, O. Jousson, and C. Huttenhower, "Metagenomic microbial community profiling using unique clade-specific marker genes," *Nature methods*, 2012.
[70] N. Segata, L. Waldron, A. Ballarini, V. Narasimhan, O. Jousson, and C. Huttenhower, "Metagenomic microbial community profiling using unique clade-specific marker genes," *Nature methods*, 2012.
[71] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology," in *MICRO*, 2017.
[72] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ISCA*, 2016.
[73] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm," *Integration*, 2017.
[74] Synopsys, Inc., "Synopsys Design Compiler," https://www.synopsys.com/support/training/rtl-synthesis/design-compiler-rtl-synthesis.html.
[75] D. T. Truong, E. A. Franzosa, T. L. Tickle, M. Scholz, G. Weingart, E. Pasolli, A. Tett, C. Huttenhower, and N. Segata, "MetaPhlAn2 for enhanced metagenomic taxonomic profiling," *Nature methods*, 2015.
[76] Y. Turakhia, G. Bejerano, and W. J. Dally, "Darwin: A genomics co-processor provides up to 15,000x acceleration on long read assembly," *ACM SIGPLAN Notices*, 2018.
[77] H. M. Waidyasooriya, M. Hariyama, and M. Kameyama, "FPGA-accelerator for DNA sequence alignment based on an efficient data-dependent memory access scheme," *Highly-Efficient Accelerators and Reconfigurable Technologies*, 2014.
[78] K. Wang, J. Alzate, and P. K. Amiri, "Low-power non-volatile spintronic memory: STT-RAM and beyond," *Journal of Physics D: Applied Physics*, 2013.
[79] Y. Wang, Y. Han, C. Wang, H. Li, and X. Li, "RADAR: A Case for Retention-Aware DRAM Assembly and Repair in Future FGR DRAM Memory," in *DAC*, 2015.
[80] R. Waser, R. Dittmann, G. Staikov, and K. Szot, "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," *Advanced materials*, 2009.

[81] D. Weese, M. Holtgrewe, and K. Reinert, "RazerS 3: faster, fully sensitive read mapping," *Bioinformatics*, 2012.
[82] Wetterstrand KA., "DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP)," https://www.genome.gov/sequencingcostsdata.
[83] D. E. Wood, J. Lu, and B. Langmead, "Improved Metagenomic Analysis with Kraken 2," *Genome biology*, 2019.
[84] D. E. Wood, J. Lu, and B. Langmead, "Kraken 2 Open-Sourced Implementation." https://github.com/DerrickWood/kraken2, 2019.
[85] D. E. Wood and S. L. Salzberg, "Kraken: Ultrafast Metagenomic Sequence Classification Using Exact Alignments," *Genome biology*, 2014.
[86] J. C. Wooley, A. Godzik, and I. Friedberg, "A primer on metagenomics," *PLoS Comput Biol*, 2010.
[87] L. Wu, R. Sharifi, M. Lenjani, K. Skadron, and A. Venkat, "Sieve: Scalable in-situ dram-based accelerator designs for massively parallel k-mer matching," in *ISCA*, 2021.
[88] Q. Xia and J. J. Yang, "Memristive crossbar arrays for brain-inspired computing," *Nature materials*, 2019.
[89] L. Xie, H. A. Du Nguyen, J. Yu, A. Kaichouhi, M. Taouil, M. AlFailakawi, and S. Hamdioui, "Scouting logic: A novel memristor-based logic design for resistive computing," in *ISVLSI*, 2017.
[90] H. Xin, D. Lee, F. Hormozdiari, S. Yedkar, O. Mutlu, and C. Alkan, "Accelerating read mapping with FastHASH," in *BMC genomics*, 2013.
[91] M. Zahedi, M. Mayahinia, M. A. Lebdeh, S. Wong, and S. Hamdioui, "Efficient organization of digital periphery to support integer datatype for memristor-based cim," in *ISVLSI*, 2020.
[92] M. Zahedi, R. van Duijnen, S. Wong, and S. Hamdioui, "Tile Architecture and Hardware Implementation for Computation-in-Memory," in *ISVLSI*, 2021.
[93] F. Zokaee, H. R. Zarandi, and L. Jiang, "Aligner: A process-in-memory architecture for short read alignment in rerams," *IEEE Computer Architecture Letters*, 2018.