CASA: An Energy-Efficient and High-Speed CAM-based SMEM Seeding Accelerator for Genome Alignment

Yi Huang
School of Integrated Circuits, BNRist, Tsinghua University
Beijing, China
y-huang21@mails.tsinghua.edu.cn

Lingkun Kong
Rice University
Houston, TX, USA
klk@rice.edu

Dibei Chen
School of Integrated Circuits, BNRist, Tsinghua University
Beijing, China
chendb17@mails.tsinghua.edu.cn

Zhiyu Chen
Rice University
Houston, TX, USA
zhiyuchen2017@gmail.com

Xiangyu Kong
School of Integrated Circuits, BNRist, Tsinghua University
Beijing, China
kxy21@mails.tsinghua.edu.cn

Jianfeng Zhu
School of Integrated Circuits, BNRist, Tsinghua University
Beijing, China
jfzhu@tsinghua.edu.cn

Konstantinos Mamouras
Rice University
Houston, TX, USA
mamouras@rice.edu

Shaojun Wei
School of Integrated Circuits, BNRist, Tsinghua University
Beijing, China
wsj@tsinghua.edu.cn

Kaiyuan Yang
Rice University
Houston, TX, USA
kyang@rice.edu

Leibo Liu*
School of Integrated Circuits, BNRist, Tsinghua University
Beijing, China
liulb@tsinghua.edu.cn

ABSTRACT

Genome analysis is a critical tool in medical and bioscience research, clinical diagnostics and treatment, and disease control and prevention. Seed and extension-based alignment is the main approach in the genome analysis pipeline, and BWA-MEM2, a widely acknowledged tool for genome alignment, performs seeding by searching for super maximal exact match (SMEM). The computation of SMEM searching requires high memory bandwidth and energy consumption, which becomes the main performance bottleneck in BWA-MEM2. State-of-the-Art designs like ERT and GenAx have achieved impressive speed-ups of SMEM-based genome alignment. However, they are constrained by frequent DRAM fetches or computationally intensive intersection calculations for all possible k-mers at every read position.

We present a CAM-based SMEM seeding accelerator for genome alignment (CASA), which circumvents the major throughput and power bottlenecks brought by data fetches and frequent position intersections through the co-design of a novel CAM-based computing architecture and a new SMEM search algorithm. CASA mainly consists of a pre-seeding filter table and a SMEM computing unit. The former expands the k-mer size to 19 using limited on-chip memory, which enables the efficient filtration of non-SMEM pivots. The latter applies a new algorithm to filter out disposable SMEMs that are already contained in other SMEMs. We evaluated a 28nm CASA implementation using the human and mouse genome references. CASA achieves 1.2× and 5.47× throughput improvement over ERT and GenAx while only requiring less than 30GB/s DRAM bandwidth and keeping the same alignment results as BWA-MEM2. Moreover, CASA provides 2.57× and 6.69× higher energy efficiency than ERT and GenAx.

*Corresponding author: Leibo Liu (liulb@tsinghua.edu.cn)

MICRO '23, October 28–November 01, 2023, Toronto, ON, Canada

ACM ISBN 979-8-4007-0329-4/23/10…$15.00
https://doi.org/10.1145/3613424.3614313

CCS CONCEPTS

• **Hardware → Biology-related information processing**.

KEYWORDS

CAM, Genome Alignment, SMEM Seeding, Filtering

ACM Reference Format:
Yi Huang, Lingkun Kong, Dibei Chen, Zhiyu Chen, Xiangyu Kong, Jianfeng Zhu, Konstantinos Mamouras, Shaojun Wei, Kaiyuan Yang, and Leibo Liu. 2023. CASA: An Energy-Efficient and High-Speed CAM-based SMEM Seeding Accelerator for Genome Alignment. In *56th Annual IEEE/ACM*

International Symposium on Microarchitecture (MICRO '23), October 28–November 01, 2023, Toronto, ON, Canada. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3613424.3614313

1 INTRODUCTION

Gene sequence analysis is of paramount importance as it provides vital insights into the underlying genetic mechanisms of life. The rapid development of gene sequencers and sequencing algorithms leads to a significant reduction in monetary and time costs [25, 52]. However, the sequencing data derived from the sequencer is massive and typically broken into billions of fragments known as reads, ranging from around 35 bp (base pairs) to 250 bp short reads in SRS (short-read sequencing technologies) [8, 15, 65] and more than 1 Kbp long reads using LRS (long-read sequencing technologies), such as ONT [41, 43, 58] and PacBio [44, 57]. Such massive sequencing data generation, in turn, brings the demand for high-speed and real-time genomics analysis.

In genome analysis [18, 60], one common task is to align reads against a long sequence of DNA known as a reference, which consists of billions of nucleotides or so-called bases (A, G, T, and C, encoded using 2-bit). Such a long reference sequence makes traditional substring lookup algorithms (e.g., Boyer-Moore (BM) [4] and Knuth-Morris-Pratt (KMP) [32]) extremely slow. Meanwhile, genetic mutations, variations, and sequencing errors make the alignment more complicated, known as approximate string matching (ASM) [10, 20, 24, 49, 50, 54, 55, 59, 62, 70]. Consequently, new algorithms have been developed for identifying the position of a read within the reference, including BWA-MEM [38, 66] for short reads alignment and Minimap [39, 40] for long reads. This work focuses on short read processing as the Illumina short read sequencing machine [25] remains the predominant solution in the field today.

Seed and seed extension is the mainstream read alignment method [36–38]. During the seeding phase, the aligner first looks up the potential matching positions of a read in the reference by finding exact matches of small substrings with a size k, called seeds or k-mer. Then in the extension phase, the aligner performs the ASM process on all reference hits found in the seeding phase and outputs the reference hits with the best alignment score [59] or the least errors [50].

A popular heuristic employed in the widely adopted BWA-MEM [12, 37, 38] seeding is based on super-maximal exact matches (SMEM). The SMEMs with a size equal to or greater than a certain number, l, are reported as seeds. Intuitively, larger l values result in fewer candidate positions. Therefore, in practice, a small value of l leads to an excessive number of hits for later processing, while a large l may lead to inaccurate alignments. BWA-MEM2 empirically sets $l = 19$ as a good trade-off between performance and accuracy.

In software implementations of SMEM seeding, the FM-index [35, 37] requires a one-base-at-a-time lookup, leading to frequent, irregular, and unpredictable memory access to DRAM. This results in limited performance and high energy consumption on CPUs and GPUs [6, 7]. The learned index LISA and BWA-MEME [21, 27] offer higher speed [6] by reducing memory access and offering multi-stride search, but their index tables exceed 100 GB in size.

Custom SMEM seeding accelerators have gained significant interest and have shown significant performance and power improvement over software implementations. State-of-the-art (SOTA) accelerators employ various methods to enhance data locality and reduce memory access. GenAx [16] and GenCache [51] explore seed & position tables for better data locality, which reduces off-chip DRAM access and optimizes on-chip SRAM access and data movement. ERT [61] explores algorithmic optimization of index table data structures to use off-chip DRAM as the main memory but with reduced accesses. All prior accelerators opt to look up a k-mer that is smaller than the threshold $l = 19$ in their index table and then extend this k-mer into a SMEM. The k-mer size is limited due to the exponential growth of the index table as the k-mer size increases, leading to the processing of numerous positions that do not result in any SMEMs. This, consequently, causes redundant SMEM computation.

Here, we propose a Content Addressable Memory (CAM) based Seeding Accelerator, CASA, for accelerating SMEM seeding. CASA stores the reference genome in the CAM and enables the detection of SMEMs by matching k-mers on the read to the reference in CAM. CAMs, typically used for string matching, are broadly used in genomics analysis [3, 19, 22, 30, 34, 46, 67, 71, 72]. Prior ReCAM-based seeding accelerators such as GenPIP [46] and SaVI [34] provide impressive speedup against CPU. However, they require large on-chip ReRAM, which is incompatible with the CMOS technology. Meanwhile, prior CAM-based seeding accelerators use a naive strategy that matches k-mers against all the CAM entries which store the reference. Such an approach is energy-expensive.

To reduce the energy consumption and maintain high throughput for SMEM seeding with limited on-chip capacity, we propose a tightly coupled co-design of filter-enabled SMEM seeding algorithm and architectural design to (1) filter out k-mers which the current reference part does not contain, (2) increase k-mer size to 19 for higher k-mer filter rate with a low area overhead while still keeping k less than the minimum SMEM length to maintain accuracy, (3) predict and filter out pivots on the read that will not lead to SMEMs, (4) fully utilize the CAM parallelism with low demand for DRAM bandwidth, and (5) selectively enable CAM entries when performing the SMEM computing to save energy.

In summary, this paper makes the following contributions:

- We propose CASA, an energy-efficient and high-speed CAM-based SMEM seeding accelerator. CASA replaces the index table fetches by performing multi-stride matches in CAMs. Through holistically optimized architecture and algorithm designs, CASA improves the throughput and energy efficiency while only increasing 33.9% area consumption compared to GenAx.
- We propose a cache-like, 3-stage pre-seeding filter to discard non-existent k-mers. This table allows scaling up the k-mer size by linearly increasing memory capacity while SOTA requires exponential memory increase. Thus, CASA supports 19-mer with no memory increase compared to the 12-mer seed & position table in GenAx, but much higher filter rate for better performance and efficiency. In our experiments, for 1 MB references, CASA needs a 45 MB pre-seeding filter table of 19-mer, while the index table for GenAx, GenCache, or ERT will require at least 800 GB memory.

- We propose a filter-enabled SMEM seeding algorithm that leverages the information from our pre-seeding filter. This algorithm efficiently detects pivots on the read that cannot lead to a SMEM. By discarding these pivots during SMEM detection, this algorithm provides a 10× acceleration in throughput for GRCh38.
- CASA is rigorously evaluated in the 28 nm CMOS process on the GRCh38 human genome assembly with 787,265,109 Illumina Platinum Genomes 101 bp single-ended reads and GRChm39 mouse genome assembly with simulated reads. While only using 55 MB on-chip memory, CASA achieves 17.26×, 1.2×, and 5.47× higher seeding throughput than BWA-MEM2, ERT, and GenAx, and CASA provides 2.57× and 6.69× higher energy efficiency than ERT and GenAx.

2 BACKGROUND

2.1 Seeding by Searching for SMEMs

As an important step in read alignment, *Seeding* computes a set of candidate positions (*hits*) in the reference genome where a read may potentially align. BWA-MEM2 [37, 38, 66] is a state-of-the-art read aligner that is broadly used as part of the Broad Institute's best practices genomics pipeline [18]. BWA-MEM2 also uses seeding to find candidate positions for potential alignment. The seeding algorithm in BWA-MEM2 is based on the identification of substrings in the read that have *super-maximal exact matches* (SMEMs) with the reference genome. A *maximal exact match* (MEM) is a substring in the read that exactly matches a substring in the reference genome such that it cannot be extended by adding new bases in either direction. A SMEM is a MEM that is not fully contained in any other MEM.

In general, there are two ways to search for SMEMs. The first way is to extend exact matches in the read in both directions, which we call *bi-directional search*, and the second way is to always extend exact matches to the right, which we call *uni-directional search*.

Bi-directional search is used in BWA-MEM2 [38, 66] and [61]. As shown in Figure 1(a), it starts from one position in the read (*pivot*) and first searches for exact matches on the right of the pivot until a mismatch (i.e., *forward search*). The mismatch position in the read will become the next pivot. During such a forward search, all the positions in the read where there is a change in the number of hits will be recorded as *left extension points* (LEPs). Starting from each recorded LEP, the bi-directional search will look for the longest exact matches on the left of the pivot (i.e., *backward search*). After the backward search, the bi-directional search will detect all MEMs in a read. SMEMs, therefore, can be identified by discarding MEMs fully contained in other longer MEMs.

Uni-directional search is used in GenAx [16] and GenCache [51]. As shown in Figure 1(b), for each base (**pivot**) in the read, this approach searches for a right maximal exact match (RMEM) with a minimum length (E.g., 19 in BWA-MEM). After all RMEMs are detected, SMEMs can be identified by discarding RMEMs contained in other RMEMs.

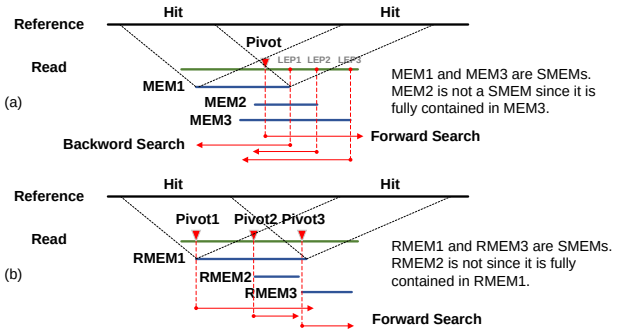

Figure 1: (a) Bi-directional and (b) uni-directional search for SMEM seeding

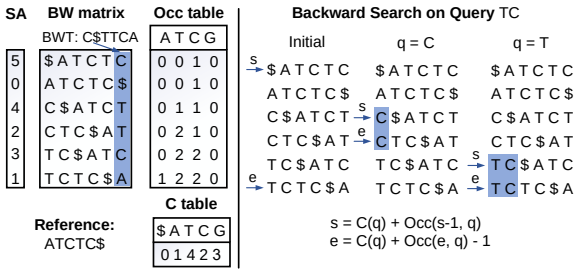

Figure 2: Example of FM-index-based seeding

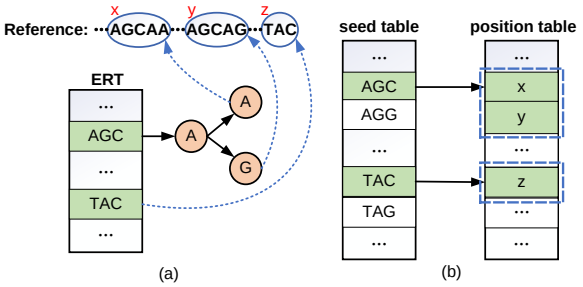

Figure 3: (a) ERT-index (b) Seed & Position tables

2.2 Data Structures for SMEM Seeding and Hardware Limitation

To identify SMEMs and their corresponding hits on the reference, prior works use different data structures to support the lookup of a string on the reference genome. Table 1 summarizes the pros and cons of each data structure.

FM-index. The FM-index structure is widely used in aligners and metagenomics classification tools such as Bowtie, BWA-MEM, and Centrifuge [28, 36–38]. As shown in Figure 2, the FM-index consists of (1) a *suffix array* (SA) that contains the locations of lexicographically sorted suffixes of the reference genome ($ is inserted as the smallest character), (2) a *Burrows-Wheeler Transform* (BWT) that is the last column of the cyclically sorted suffix array of the reference, (3) a *count table* (C) that contains the number of characters lexicographically smaller than a given character, (4) an *occurrence table* (Occ) that stores the number of occurrences of a given character prior to a certain index in the suffix array. One can calculate the position interval (*s* and *e*) of a specific query using C and Occ as shown in the right side of Fig 2. The FM-index allows the lookup of a

string of length n in a reference genome using approximately $O(n)$ memory operations. The FM-index inherently involves sequential dependent memory accesses, which hampers its performance in terms of both bandwidth utilization and throughput.

ERT-index. The *Enumerated-Radix-Trees* (ERT) index is proposed in [61], and it is adopted in one of the latest versions of BWA-MEM2 [5], whose FPGA implementation outperforms the original 72-thread CPU-version BWA-MEM2 by 3.5×. As shown in Figure 3(a), the ERT-index consists of an index table whose indexes are k-mers and values are pointers to the root of a radix tree. In the lookup of SMEMs, the index table is accessed by a k-mer on the read, and the root of a radix tree is fetched if the k-mer leads to a hit on the reference. To find SMEMs, one can perform a forward search by expanding the k-mer, followed by a backward search, where k-mers can be expanded by comparing the subsequent letters after the k-mer on the read to the letters inside the tree nodes. The ERT-index has better space locality and provides speedup compared with FM-index since it uses k-mers as indexes and supports quick forward search by expanding k-mers with radix trees. The memory footprint of the ERT-index is $O(4^k + n)$, where k is the size of k-mer, and n is the length of the reference. When k is large, a substantial amount of memory is required.

The ERT accelerator [61] uses the bi-directional approach and the ERT index to identify SMEMs and has improved the data efficiency significantly compared to FM-index. However, to support a 15-mer index table, it demands high DRAM capacity and, therefore, has high power consumption. For example, the size of the dedicated memory required by the ERT-index is 62.1 GB [64]. We have evaluated the 64 GB DDR4 power consumption of the ERT accelerator for seeding computation using DRAMpower [9]. We observed the power consumption of DDR4 is higher than 15 W, which occupies about 47% of the power usage in ASIC-ERT. Moreover, it still has some random accesses left caused by tree root fetches and k-mer searches. Thus, the DRAM bandwidth utilization remains a throughput bottleneck in ERT. We have evaluated the bandwidth utilization of the ERT accelerator using Ramulator [31] with 1 M reads randomly selected from ERR194147_1.fastq[14] that contain about 80 % exact matches on GRCh38 [64]. We observed that only about 50 % DDR4 bandwidth on average is utilized, which is consistent with the result reported in [61].

Seed & Position Tables. Prior works like Darwin [63] and GenAx [16] provide a data structure that contains both seed and position tables for SMEM searching. As shown in Figure 3(b), the seed table is indexed by k-mers, and each of its entries points to a set of hits recorded in the position table. Such a data structure enables the efficient lookup of the hits of k-mers in the aforementioned unidirectional approach, which accelerates the computation of RMEMs. To illustrate, to compute RMEM for a pivot x on the read, we can determine the hits (H_1) for the first k-mer starting from the pivot by the lookup of seed & position tables. Then, we can stride by k and find the hits (H_2) for a k-mer starting at $x + k$ position in the read. By intersecting H_1 and H_2, we find the hits ($H_1 \cap H_2$) for the 2k-mer after the pivot. We can continue this process until the intersection returns an empty set. Then, we reduce the stride progressively from $k/2$, $k/4$, $k/8$, ..., 1, which effectively is a binary search, to compute the RMEM with non-zero hits. This data structure with seed & position tables provides a better locality compared to FM-index. However, it also has a large memory footprint $O(4^k + n)$, where k is the size of k-mer, and n is the length of the reference.

Table 1: Pros and Cons of Data Structures Used for Seeding

FM-index	Pros: low memory cost. Cons: low throughput and low bandwidth utilization.
ERT-index	Pros: high throughput. Cons: high memory cost with large k-mer.
Seed & Position Tables	Pros: high throughput, easy-to-implement algorithm. Cons: high memory cost with large k-mer.

GenAx [16] chooses an on-chip implemented seed & position table and performs a unidirectional RMEM search, which features 128 seeding lanes to query 128 k-mers simultaneously. Although the on-chip seed table can reduce the DRAM usage and thus lower the DRAM power, the large number of pivots to start the unidirectional RMEM search incur massive k-mer fetches and k-mer intersections, becoming the major performance bottleneck in GenAx. We observed that there exist ∼4000 position intersections and ≥200 index fetches per read per segment. These k-mer fetches and intersections in one pivot show low parallelism since the binary search of RMEM requires the hardware controller to know the next k-mer to search. Moreover, SRAM conflicts restrict the number of seeding lanes, further limiting GenAx's performance. To make things worse, the same batch of reads should conduct such an expensive process repeatedly 512 times in the human genome due to the limited on-chip memory.

GenCache [51] refines the design of GenAx, introducing a fast seeding mechanism to handle reads with low errors, effectively bypassing SMEM seeding for these reads. However, GenCache employs the same SMEM computing algorithm as GenAx, which requires considerable k-mer fetches and intersections per read, continuing to bottleneck the overall pipeline. Additionally, GenCache opts for a multi-bank cache to store the index table instead of on-chip storage, triggering extensive DRAM fetches and significantly diminishing the overall SMEM seeding performance.

2.3 Binary CAM (BCAM)

Figure 4(a) shows a conceptual view of a BCAM array. Once the search data match the stored data in the memory, a logic '1' will be generated at the corresponding row. As shown by the 10-transistor NOR-type BCAM cell [53] in Figure 4(b), two pull-down paths from the *match line* (ML) to ground formed by M1 to M4 implement a logical XNOR between the input search data on the differential *search line* (SL1/SL2) and the *differential storage data* (D1/D2) in the 6-transistor SRAM cell (bitlines, wordlines and access transistors are omitted for simplicity). A mismatch between D1/D2 and SL1/SL2 will pull down the pre-charged ML through one of the NMOS pairs, while a match will break both pull-down paths. When multiple cells are connected to a single ML to form a CAM word, all pull-down paths are effectively performing logical NOR operations. The ML remains pre-charged voltage (i.e. logic '1') only when all cells are in the matched state.

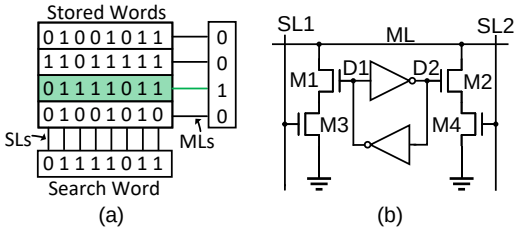

Figure 4: (a) Conceptual diagram of BCAM (b) NOR-type BCAM cell

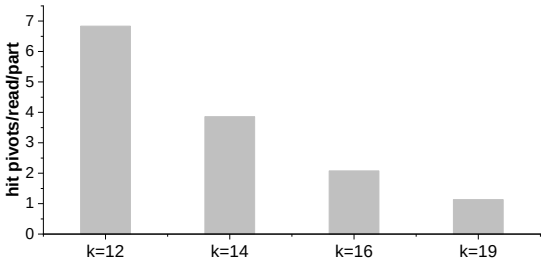

Figure 5: The number of k-mers that lead to hits on the reference genome scaling with the size of the k-mer

3 DESIGN PRINCIPLES OF CASA

SOTA seeding algorithms require SMEM lengths to exceed a certain number (e.g., 19 in BWA-MEM2 [37]), and only pivots that are starting positions of k-mers, which cause hits on the reference, can potentially be starting positions of SMEMs. We observe that as the k-mer size increases, the number of k-mers on the read that have a hit on the reference genome declines sharply. As depicted in Figure 5, increasing k from 12 to 19 results in a 6.04-fold decrease in the number of k-mers that leads to a hit on a reference genome partition. The design of CASA is inspired by this observation and is tailored to efficiently fetch large k-mers from the index table. This enables CASA to filter out many pivots, preventing unnecessary SMEM computation and accelerating seeding computation.

Besides, prior works like ERT perform the matching of k-mers by storing the reference into index tables, which requires a memory footprint of size $O(4^k + n)$ and pays a lot for increasing k-mer size, leading to high DRAM power.

Given that CAM naturally supports large string matching with only $O(n)$ space as shown in GenPIP [46], CAM is a superior alternative to RAM-based index tables [27, 61]. CAM can serve as the SMEM computing unit by storing the reference and comparing consecutive k-stride bases directly within itself. This approach mitigates the substantial capacity and bandwidth demands of DRAM while preserving rapid SMEM search. We call the CAM used to store the reference in CASA as the *SMEM computing CAM*.

The straightforward method to apply CAM to store the reference and perform SMEM search is using overlapped sub-strings sliding by one base without any index table (see Figure 6(b)). However, this naive approach faces three major limitations: (1) It requires performing intensive SMEM computation for each pivot on the read, which easily becomes the energy and throughput bottleneck. For example, in Figure 6(b), the naive algorithm will perform the SMEM searching on all 10 pivots on the read even though only one pivot will be detected as the start index of a SMEM (i.e., index 3 on the read). (2) It duplicates reference segments and hence increases the on-chip memory usage by k times. (3) Matching k-stride bases against all the CAM entries is power-hungry. To enable efficient SMEM computation, CASA not only builds a CAM-based index table supporting large k-mers but also introduces a novel filtering mechanism that can efficiently predict and discard pivots that will not lead to a SMEM. In addition, CASA stores the reference in CAM as non-overlapped sub-strings to reduce memory usage and enable multi-stride SMEM search. Finally, CASA proposes an efficient grouping strategy that partitions CAM entries into groups. CASA will only activate the group such that a k-mer is found inside its CAM entries to reduce the energy cost of seeding.

Filtering. CASA implements a novel filter-enabled SMEM search algorithm that can detect pivots that cannot lead to a SMEM and directly skip the computation of SMEMs for those pivots. This algorithm, utilizing a CAM-based pre-seeding filter that can determine whether a k-mer exists in the reference, is able to filter out 99.9% of the pivots on the read, which allows CASA to massive reduce the energy and time cost of computing SMEMs. We will present the design of our pre-seeding filter in § 4.1, and we will explain the details of this algorithm in § 4.2.

Non-overlapped Storage. CASA stores the reference in the CAM in a non-overlapped manner, which is inspired by the equivalence between searching a query string on k multi-stride automata and searching k query strings on a single automata. For example, Figure 7(a) shows a single-stride automaton that is used to match query `CAAT` to reference `TCAAT`, where out edges of the start state indicate each base on the reference may be the start of the query. Figure 7(b) shows 2-stride automata used to match `CAAT` with `TCAAT`, where each state contains a 2-mer for matching a 2-base subsequence inside the query. To ensure accuracy, we need to construct two automata, on both of which the query `CAAT` will be executed. The automaton on the top of Figure 7(b) reports no match, yet the automaton on the bottom reports a match. Figure 7(c) shows the matching between `CAAT` and `TCAAT` based on searching two queries on a single 2-stride automaton, which is equivalent to the computation as shown in Figure 7(b). Instead of using two automata, we will match two queries `CAAT` and `XCAAT` to `TCAAT`, where `X` is a padded base that can match any base. In Figure 7(c), `CAAT` cannot be matched by the automaton, yet `XCAAT` leads to a match. In the same spirit, when CASA searches a k-mer u in CAM, instead of storing the overlapped reference in CAM (like using k multi-stride automata), CASA stores the reference in CAM in a non-overlapped manner and creates k queries u, `X`u, `XX`u, ..., and vu to match the reference, where v is `X` repeated for $k-1$ times. This mechanism will reduce the throughput of CASA since more queries need to be processed, yet it will lower the memory usage of CASA since the reference is stored only once. As not all padding patterns match the reference for a k-mer, CASA stores how many bases to pad for a k-mer (so-called *start positions*) to reduce the padded queries. Meanwhile, our filtering algorithm discards a lot of pivots that have no SMEM. Therefore, the reduction of throughput is mitigated, and the energy efficiency of CASA can be massively enhanced.

CAM Grouping. Naive CAM implementations typically match k-stride in the whole CAM arrays, which causes excessive energy overhead. CASA proposes a group-level SMEM computing CAM management strategy to reduce such overhead. CASA pre-computes

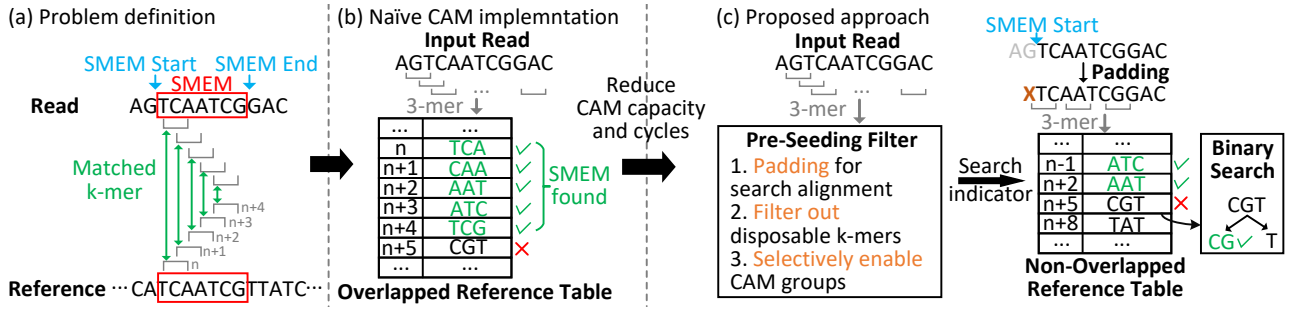

Figure 6: Design of CASA (k-mer size is 3 in the example)

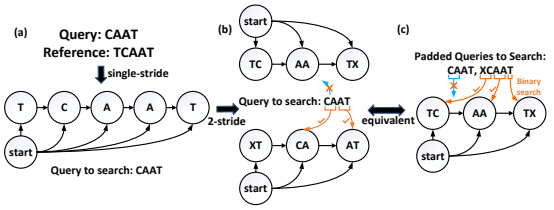

Figure 7: Construction of the non-overlapped reference

the *group indicator* offline, which is a bit vector used to indicate which CAM group contains the k-mer so that irrelevant groups can stay idle. Moreover, CASA uses *search indicator* to represent a tuple that combines the start position and the group indicator of a k-mer.

Working Example. Figure 6(c) presents an example of the whole SMEM computation in CASA. In the beginning, all read positions are viewed as pivots, and we access the pre-seeding filter table to obtain search indicators for 3-mers of all pivots. CASA then discards the unexisting 3-mers, which have no start position like AGT. The remaining is sent to the SMEM computing CAMs and the first matching 3-mer is TCA in SMEM computing phase. In this example, TCA starts at the second base inside the CAM entry, from which we know the query is padded with one X. CASA then performs a consecutive 3-stride search until a mismatch is met. Moreover, CASA will perform a binary search for the first mismatched 3-mer to acquire the exact end position of the SMEM. Finally, CASA compares the newly computed SMEM with the previous SMEMs to check whether to discard it. If the answer is no, the location of hits and the SMEM will be sent to the result buffer. Then CASA updates the candidate pivots using the new SMEM by performing the proposed filtering algorithm. Such a process will be iterated for the remaining pivots.

4 CASA DESIGN

Overall, implementing a SMEM seeding accelerator presents two major challenges. The first is the massive number of read positions initiating the SMEM computation, as mentioned in §2.2, which dominates GenAx's running time. The second is mitigating the energy demands of DRAM, which occupies nearly half of the total power consumption in ERT.

CASA reduces DRAM usage by directly storing the reference into CAMs instead of a DRAM-based index table and further prohibits the CAM from the naive power-hungry search strategy by selectively disabling match lines and readout circuits. Moreover, CASA introduces a pre-seeding filter table and a filter algorithm to avoid the enormous SMEM computing process.

4.1 Architectural Improvement

Energy-efficient SMEM Computing CAMs. The SMEM computing CAMs, storing the reference part by multi-stride, accomplish the search process of the unidirectional RMEM search algorithm. They receive the pivots and search indicators from the pre-filter table. Then they identify which pivots to start a search and how many bases should be padded into the sub-strings of these pivots. After padding, they find matches stride by stride until they meet mismatches. At the end of the stride search, a binary search will be performed on the first mismatched entries to obtain the end of SMEM. A secondary pivot filtering algorithm is performed using the search indicators and the new SMEM. And a new SMEM computing starts at the next pivot.

However, always enabling all SMEM computing CAM arrays to perform searches consumes a large amount of dynamic power. Thus, we propose power gating techniques among different CAM arrays and within one CAM array. First, we only enable specific batches of CAM arrays where the k-mer of one pivot exists while keeping others on standby. Second, the entries within each CAM array are selectively enabled based on the automata matching results in the last cycle when performing a multi-stride search. For instance, when CASA begins to search the SMEM of one pivot in the reference, in the first cycle, we will enable all the entries to search a stride of sub-string in the CAM groups that contain that pivot's k-mer. Then, for the next stride, only the CAM entries coming after the entries which have been determined to contain the currently searched stride are enabled. In implementing the coarse enabling, we cluster computing CAM arrays into groups and use a one-hot bit vector (termed group indicator) to indicate which group the k-mer belongs to. The previous cycle's match line results, stored in DFF, facilitate the disabling in the CAM array level with a small overhead.

Pre-Seeding Filter. CASA provides a pre-seeding filter to generate search indicators for k-mers and filter out pivots that have no SMEM. We notice that when we split the reference into pieces, the hit rate of the k-mer decreases dramatically, and the larger the k-mer size is, the fewer hits we will find. For example, when GRCh38 is fragmented into 768 parts, the first part only contains 0.003% of all possible 19-mers while it contains more than 80% of all possible 10-mers.

Previous studies like GenAx [16] and ERT [61] build an index table to locate k-mers on the reference, whose memory footprint

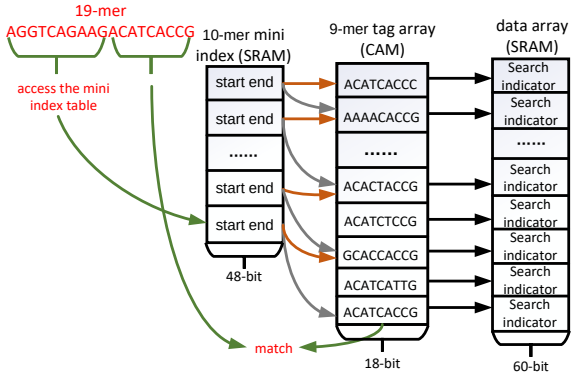

Figure 8: Architecture of proposed pre-seeding filter

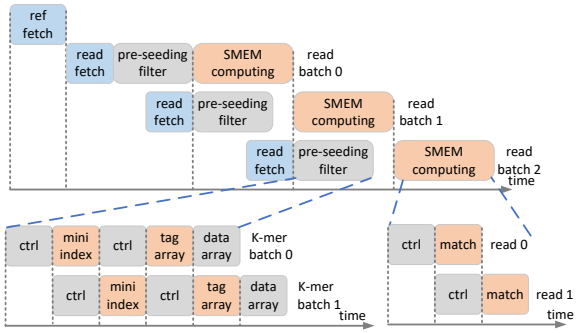

Figure 9: Timing diagram of pipelines in CASA.

grows exponentially by k, i.e., the size of the k-mer. However, directly using a k-mer CAM as GenPIP [46] did faces high power brought by enabling the whole CAM entries. Inspired by the cache architecture, we expand the tag and data arrays to a pre-seeding filter to filter out pivots using the non-existing k-mers and ensure no k-mer misses or false positives. It only stores existing k-mers and the corresponding search indicators so that the memory usage scales linearly with k. Meanwhile, we add a small SRAM array in front of the CAM tag array and use the start and end pointers from the SRAM to power-gating the CAM array, which reduces the power of the CAM array.

Figure 8 shows the structure of our pre-seeding filter, which consists of three main components: (1) a mini index table for m-mers with 4^m entries, where m is smaller than k, (2) a tag array implemented by CAM which stores the rest of (k-m)-mers with n entries, where n is the number of bases in the partition of the reference sequence (e.g., 4M for a 1MB partition), and (3) a data array that stores the search indicators with the same number of entries as the tag array.

CASA builds the mini index table and the tag table offline for each reference partition using the following steps: (1) To begin with, for each k-mer that starts at index x in the partition, we will compute its start positions in the CAM as $x \mod 40$ and the group indicator as $x \mod 20$, where 40 is the stride length, and 20 is the number of CAM groups. (2) We will then sort k-mers in lexicographical order and split each k-mer into an m-mer and a (k-m)-mer. In our design, we choose $m = 10$ for $k = 19$ to roughly split the k-mer half by half. As a certain number of k-mers will share the same m-mer, each m-mer, which has been sorted in lexicographical order, may correspond to a range of (k-m)-mer. In this case, we also store the start and the end of such range inside the mini index table. (3) Finally, for each (k-m)-mer, we will encode its search indicator into a one-hot bit vector and assign it to each entry of the tag table.

As all k-mers contained in the reference partition are enumerated, the proposed pre-seeding filter table avoids k-mer false positives or misses, unlike the bloom filter in GenCache. Worth mentioning, we observe that matching the (k-m)-mer in the whole tag array is still power-hungry. To reduce this dynamic power, the start and end pointers fetched from the mini-index table are decoded in a range decoder to power-gating corresponding entries in the tag array.

The whole process of determining whether a 19-mer exists in the reference is shown in Figure 8. We will first split the 19-mer into a 10-mer and a 9-mer. We will use the 10-mer as the address to access the mini-index table and fetch the start/end pointers to 9-mers stored in the tag table that share the same 10-mer prefix. If the 9-mer split from the 19-mer matches one of the entries between the pointers, the 19-mer exists in the reference part, and the connected data array row will be activated to fetch the 19-mer's search indicator.

In summary, the proposed pre-seeding filter has the space complexity of $O(4^m + n)$, in which n is the reference part size, m is the mini m-mer size which is much smaller than k. Therefore, the proposed pre-seeding filter eliminates the exponential dependence on k and replaces it with a fixed smaller m. As mentioned before, we empirically choose k to be 19 and m to be 10. The total on-chip memory usage of the pre-seeding filter is only 45 MB to support 19-mer for 1 MB reference. It is smaller than 66 MB in GenAx for 1.5 MB reference (scaled to 1 MB reference, this will be 60 MB), which only supports 12-mer. Besides, the k size increase brings significant throughput enhancement. According to our evaluation in Figure 5, we find that increasing the k-mer size from 12 to 19 reduces the pivots per partition by 6.04×, leading to a cycle decrease in the 19-mer case.

Overall Operating Pipeline. Our design consists of three stages and executes them in the pipeline in read batch level: read fetch, pre-seeding filter, and SMEM computing, as shown in Figure 9. In the pre-seeding phase, three reads (together with the reverse strands) are sent to the pre-seeding filter each time, and the reads that have k-mers existing in the current reference part are then sent to the 512-entry FIFO together with its pivots' search indicators. In the SMEM computing phase, the computing controller fetches a read and its search indicators from the FIFO and begins SMEM computing. CASA implements a multi-banking pre-seeding filter like GenAx, where multiple reads and searches can occur with high parallelism. Therefore, the pre-seeding phase is typically faster than the SMEM computing phase. Nevertheless, there exist cases where the latency of the SMEM computation cannot hide the latency of the pre-seeding filtering. Such cases typically occur when we process the first read batch in the SMEM computation where no pivot has k-mer existed in the pre-seeding filter. As FIFO allows read and write in parallel, CASA can overlap the pre-seeding filter phase and the SMEM computing phase for the same read batch to better conceal the latency of the pre-seeding filter phase in such cases.

Therefore, the throughput of CASA is mainly determined by the SMEM computing stage.

Since the proposed pre-seeding filter consists of three hardware components and the pre-seeding controller takes time to schedule multi-bank accesses and selectively enable tag array, we design a sub-pipeline within the pre-seeding phase in read level, which has five stages as illustrated in the bottom of Figure 9. The first and third stage schedule the multi-bank accesses to the mini index table and tag array. The other three stages are used to access the mini index table, tag array, and data array, respectively. Moreover, we realize another sub-pipeline in the SMEM computing phase which consists of two stages (Figure 9), the computing controller phase and the CAM matching phase. Because the computing controller decides how many bases we should match in the binary search based on the last cycle's match results, we design this to process two reads in parallel and break down the critical path.

4.2 Algorithm for filtering out pivots

We use a pre-seeding filter table to store k-mers that exist in the reference genome. In the uni-directional search of RMEMs, we can discard a lot of pivots on the read if the k-mer that starts on the pivot is not stored in the pre-seeding filter table. We will compute RMEMs for the rest of the pivots. Some RMEMs are completely contained in other longer RMEMs, which we call *disposable RMEMs* since they will be further discarded in the identification of SMEMs. In practice, if we know which pivot on the read may lead to a disposable RMEM, which we call *disposable pivot*, before computing the RMEM, we can filter out this pivot before seeding and thereby improve the performance.

Let us consider a SMEM r that starts at pivot x and ends at index y on the read (i.e., the length of r is $y-x+1$). In this case, the closest k-mer on the right side of r should start at index $y-k+2$ and end at $y+1$, which we call the *closest right k-mer* (CRkM) of r. In example 1 of Figure 10, for a SMEM `CATTGTCA` on the read (i.e., $x=3, y=10$ since we consider the index starts at 1), the closest right 4-mer (CR4M) of this SMEM is `TCAT`. We can determine whether a pivot z indexed on the right of x (i.e., $z > x$) is disposable by performing `Analysis1` that checks whether r is non-extendable and `Analysis2` that checks whether the k-mer starts at z is unaligned with the CRkM of r.

We say r is non-extendable if we cannot find a hit for the CRkM of r on the reference. If this is the case, no string on the read that covers the CRkM of r can be found on the reference genome, which indicates no pivot between the start index of r (i.e., x) and the start index of the CRkM (i.e., $y-k+2$) can lead to a RMEM that is not fully covered by r. Therefore, pivots $x+1, x+2, ..., y-k+2$ are disposable, and $y-k+1$ will be set as the next pivot for later SMEM searching. In example 1 shown in Figure 10, if the CR4M `TCAT` of the SMEM cannot be fetched from the pre-seeding filter table (i.e., it has no hit on the reference), we can skip pivots indexed from 3 to 8 (i.e., `CATTGT`) and set the next pivot as 9 (i.e., `C`).

If r is extendable, we can apply the second analysis to further determine whether a pivot z between the head of r and the head of the CRkM of r is disposable. In this analysis, we will check whether the k-mer that starts at pivot z is unaligned with the CRkM of r. We consider this k-mer has a set of hits $\{(a_1, a_1+k-1), (a_2, a_2+k-1), ...\}$, where a_i corresponds to the starting index of the hit on the reference genome. We also consider the CRkM of r has a set of hits $\{(b_1, b_1+k-1), (b_2, b_2+k-1), ...\}$, where b_i is the starting index of the hit. We say the k-mer that starts at z is unaligned with the CRkM of r iff the distance between the heads of their hits is always not the same as their heads on the read, i.e., $|b_j - a_i| \neq (y-k+2)-z$ for all i, j. If they are not aligned, there is no way we can extend a RMEM from z that is not fully covered by r, i.e., pivot z is disposable. However, computing the starting indexes of hits (i.e., a_i and b_i) for a k-mer is expensive, since in the seed&position table, one needs to compare all the positions of two k-mers. Our CAM-based architecture allows us to efficiently estimate whether these two k-mers are unaligned by only performing a shifted-AND on search indicators and one padded search. Suppose the size of each CAM entry is s, a necessary but not sufficient condition for $|b_j-a_i| \neq (y-k+2)-z$ is $|b_j-a_i| \bmod s \neq ((y-k+2)-z) \bmod s$. Therefore, by checking the starting indexes of hits in the CAM

Algorithm 1 Filter-enabled SMEM computing algorithm

Input: A set of pivots $\{x_0, x_1, ..., x_n\}$ on a read
Output: SMEM set $S = \{s_0, s_1, ..., s_m\}$ and their hit set $H = \{h_0, h_1, ..., h_m\}$ in the reference

```
1: procedure CRkM_Check(last_smem, k)
2:     // Check whether the closest right k-mer of last_smem exists. Return 1 if it exists, otherwise 0.
3: end procedure
4: procedure Align_Check(last_smem, pivot, k)
5:     // Check whether the k-mer on pivot is aligned with the CRkM of last_smem in CAM. Return 1 if aligned, otherwise 0.
6: end procedure
7: procedure RMEM_Search(pivot)
8:     // Start a uni-direction RMEM search at pivot. Return a RMEM and its hits on the reference if found, otherwise null.
9: end procedure
10: procedure OVERLAP_Check(last_smem, rmem)
11:     // Check whether last_smem fully contains rmem. Return 1 if it is the case, otherwise 0.
12: end procedure
13: Initialize last_smem as null, k as k0, and ;
14: for each pivot on read do
15:     if last_smem = null then
16:         s, h = RMEM_SEARCH(pivot);
17:         last_smem = s;
18:         Add s and h into S and H;
19:     else if CRkM_CHECK(last_smem, k) = 0 && pivot < last_smem.tail − k + 3 then
20:         // Discard this pivot
21:         Continue;
22:     else if ALIGN_CHECK(last_smem, pivot, k) = 0 then
23:         // Discard this pivot
24:         Continue;
25:     else
26:         s, h = RMEM_SEARCH(pivot);
27:         if OVERLAP_CHECK(last_smem, s) = 0 then
28:             last_smem = s;
29:             Add s and h into S and H;
30:         end if
31:     end if
32: end for
```

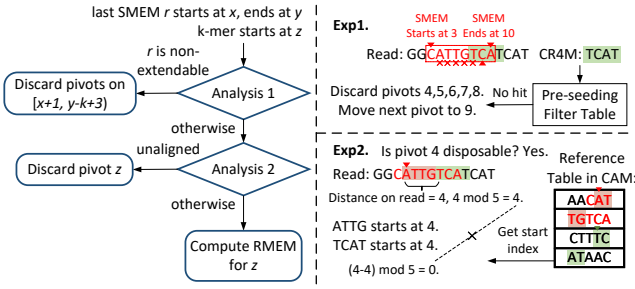

Figure 10: Overview of the pre-seeding filter algorithm.

entry (i.e., $b_j \mod s$ and $a_i \mod s$), we can determine whether two k-mers are unaligned. In example 2 of Figure 10, we assume that the CR4M TCAT is not found on the pre-seeding filter table. In this case, the second analysis is triggered to determine whether pivots 4, 5, 6, 7, and 8 are disposable. In our example, we consider the second analysis on pivot 4 (i.e., $z = 4$) to check whether it is aligned with the CR4M TCAT in CAM. Suppose each CAM entry is of size 5 and the start index in each entry is 1, we will first fetch the starting index of TCAT in the CAM entry which returns 4. We will then check the starting index of 4-mer starts at z (i.e., ATTG) in the CAM entry, which returns 4. As ATTG (resp., TCAT) has only one hit in CAM, we know the distance between the heads of their hits (denoted by d_h) satisfies $d_h \bmod 5 = 0$, where 5 is the size of each CAM entry. The distance between ATTG and TCAT on the read is 4 (i.e., $8 - 4$). If we use d_r to denote this distance, we have $d_r \bmod 5 = 4$. As $d_r \bmod 5 \neq d_h \bmod 5$, we know ATTG is unaligned with TCAT. Therefore, we can discard pivot 4.

Notice here we may overapproximate some unaligned k-mers as aligned. This overapproximation will not affect the accuracy of our SMEM computation because, at the end of operations, we will always discard the RMEM that is fully contained in a previous SMEM to avoid false positives.

Algorithm 1 shows the pseudo-code of our SMEM seeding algorithm with the pre-seeding filter, and the left side of Figure 10 shows the flowchart of the program for processing a single pivot. For the first pivot on the read, our algorithm will perform the RMEM search (i.e., RMEM_SEARCH(pivot)) to find the first SMEM. Then, the algorithm will check whether the SMEM is non-extendable and try to discard pivots. If the SMEM is extendable, we will check whether the k-mer on the pivot is aligned with the CRkM of the last SMEM, and we will discard the pivot if it is unaligned. Finally, as shown from line 27 to 29 in Algorithm 1, we will always discard RMEMs fully contained in the last SMEM.

4.3 Pre-Processing of Exact Match Reads

GenStore [45] and GenCache [51] have shown that, when the error rate is low, filtering out reads that exactly match the reference and only computing SMEMs for the remaining reads can speed up the seeding process. This technique aligns with CASA's focus on SMEM computing acceleration, a design orthogonal to GenStore and GenCache. We adopted a two-step approach to compute SMEMs, integrating CASA with pre-processing of exact matches. First, we identify exact match reads for all reference parts like GenCache. Next, we perform the SMEM finding operation for the rest

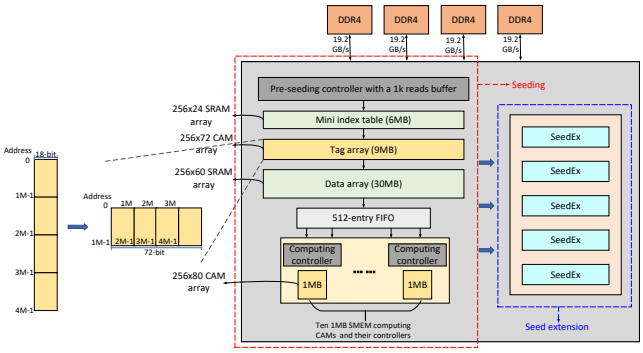

Figure 11: Overview of CASA abstraction.

of the reads. To search for the exact match reads, CASA first gathers the search indicators of several non-overlapping m-mers (m is the size of a mini index entry). Then, CASA attempts to align all these m-mers using the same method as §4.2. If these m-mers can be aligned, CASA will match the whole read in the reference. The exact match process for one read will be aborted whenever finding unaligned m-mers or mismatches.

5 SYSTEM IMPLEMENTATION

Figure 11 outlines the design of CASA. When reads are scheduled for access to the pre-seeding filter table, which consists of the mini index, tag array, and data array, some are discarded because they have no hit in the current reference. Each of the 10 SMEM computing CAMs then fetches a remaining read, selecting the pivots to initiate the uni-directional RMEM computation for the read using the proposed filter algorithm, or starting the exact match search. Each computing CAM features a result buffer to store computational results for subsequent operations. CASA then forwards the results to 5 SeedEx machines [17] for seed extension, with SeedEx being a state-of-the-art seed extension accelerator. Each SeedEx machine contains 12 BSW cores and 4 edit machines, equipping us to catch up with the seeding throughput and maintain alignments identical to BWA-MEM2. With the alignment accelerator, CASA first processes exact match reads and then proceeds to SMEM computation for the remaining reads. During both stages, the read batches are imported into CASA via two DDR4 channels, delivering an average bandwidth of 25 GB/s bandwidth.

Rather than naively using an 18-bit word CAM array to store 9-mers, which would inflate peripheral area, CASA stores four 9-mers, each striding by 1 M addresses, in one CAM entry (Figure 11). This strategy requires a 72-bit word CAM array, but it reduces the area of the tag array by 2.62× due to the shared sense amplifiers among the four 9-mers, at the expense of search energy. As the total number of 9-mers is 256 K, much smaller than the 1 M address gap in CAM entries, such data layout optimization will not bring additional cycles in the tag array lookup.

6 EVALUATION METHOD

We evaluated CASA using the latest version of human genome assembly (GRCh38) and mouse genome assembly (GRCm39) from the UCSC genome browser [64]. We replaced all the N bases in the reference genome and reads with one of the standard nucleotides. For GRCh38, we used the input reads from the ERR194147 dataset [14].

Table 2: Baseline System Configuration

CPU	Intel(R) Core(TM) i7-6800K CPU @3.40GHz with 6 cores	Intel(R) Xeon(R) E5-2699 v3 processor @2.3GHz with 18 cores 2 sockets
L1D cache	192KB Data	32KB
L2 cache	1.5MB	256KB
L3 cache	15MB	45MB/socket
Memory	96GB DRAM	64GB/socket

Table 3: Circuit models in 28nm

Components	delay(ps)	area(μm^2)	energy(pJ)	leakage(μA)	Size
6T SRAM	424	2535	2.33	6.29	256 × 24
6T SRAM	444	5563	4.89	14.18	256 × 60
6T SRAM	548	22046	20.92	38.198	256 × 256
10T BCAM	495	18056	17.60	18.69	256 × 72

This dataset consists of 787,265,109 reads, and each read is of size 101 bp. For GRCm39, we used DWGSIM [13] to generate 10 million simulated reads as input, where each read is of size 101 bp. We have compared our SMEMs among all the reference parts with BWA-MEM2 and GenAx and find out CASA produces identical SMEMs to GenAx and 100% SMEMs of BWA-MEM2 are contained without considering N-bases on GRCm39 and GRCh38 leading to the same alignment as BWA-MEM2.

We compared the performance of CASA with ASIC ERT, BWA-MEM2, and GenAx. The software-based BWA-MEM2 is performed on a CPU with 12 and 32 threads. Table 2 lists the CPU configuration. We estimated the performance of ASIC-ERT (16 seeding machines with 4 MB k-mer reuse cache) by modifying the software ERT to get the memory trace. We re-implemented the seeding in GenAx (68 MB SRAM and 128 seeding lanes) and estimated its performance by counting the seed & position table accesses and CAM lookups and assuming that GenAx can reach the 128 seeding lanes parallelism and a 60 TB/s on-chip peak bandwidth. To evaluate CASA's seeding performance, we developed a cycle-level C++ simulator, where the cycle count of the simulator is verified on a small reference partition (100 Kbp) and a mini read batch (1 K reads of 101 bp) using the RTL implementation of CASA. We evaluated the DRAM power and performance using DRAMpower with Micron DDR4 specifications [47, 48] and Ramulator [31]. The data of DRAM controller PHY [11, 33, 56] is derived from [33] by scaling the DQs and C/As.

We synthesized the pre-seeding and computing controller using Synopsys Design Compiler 2018 at the TSMC 28 nm technology node to obtain the area, delay, and power in RVT. The frequency of controllers reaches 2 GHz and is limited by scheduling k-mer accesses in the pre-seeding phase and checking the next pivot in the read in the SMEM computing phase. Table 3 lists the memory parameters of SRAM and CAM arrays in TSMC 28 nm CMOS, including access time, area, dynamic energy, and leakage. We used TSMC memory compiler to evaluate the SRAM performance. As CAM is not available in the compiler, we customized CAM arrays using a silicon-verified CAM design [68] that is in the same technology node and of similar memory capacity. Our implementation achieves comparable energy and speed performance as the results reported in [68] with SPICE simulation. We applied the same memory parameters to GenAx and ERT for fair comparisons.

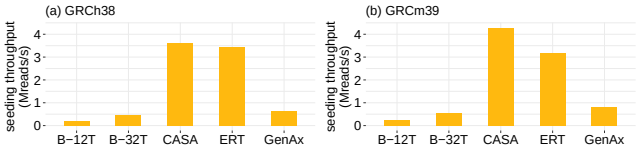

Figure 12: Seeding throughput for reference genome (a) GRCh38 and (b) GRCm39.

7 RESULTS

7.1 SMEM Seeding Performance

We have evaluated the throughput of seeding for BWA-MEM2, GenAx, ASIC-ERT, and CASA in the read alignment for reference genomes GRCm39 and GRCh38. Figure 12 shows the comparison of the seeding throughput between these tools, where the throughput is measured as million reads per second. We observe that CASA outperforms BWA-MEM2 with 12 threads (B-12T), with 32 threads (B-32T), GenAx, and ERT by 17.26×, 7.53×, 5.47×, and 1.2× on average for GRCm39 and GRCh38.

Notice CASA outperforms GenAx by more than 5 times even though it uses only 10 computing CAMs while GenAx uses 128 seed lanes. This is because (1) CASA features a pre-seeding filter table supporting larger k-mer, (2) CASA's filter-enabled SMEM computing algorithm provides a high filter rate leading to ~ 30× speedup per read, and (3) the pre-processing of exact match reads prevents ~ 80% of reads from the expensive SMEM searching computation, which provides 2.77× speedup. The seeding throughput of CASA is slightly higher than ERT. However, ERT uses on average 68 GB/s DRAM bandwidth and a 64GB dedicated DRAM for indexing. In contrast, CASA only requires only 25 GB/s DRAM bandwidth for fetching reads, which allows it to be more energy-efficient compared to ERT.

7.2 Energy Efficiency and Area Breakdown

We have evaluated the power by measuring the number of per cycle activated SRAM and CAM arrays, and the number of DRAM accesses in our simulator. Table 4 lists the result. Figure 13 shows the comparison of the power consumption and the energy efficiency between GenAx, ASIC-ERT, and CASA. The on-chip power contains the power of computing units, controllers and memory. We observe that CASA reduces the power consumption compared to GenAx and ASIC-ERT by 22 % and 91 % respectively. ERT consumes the highest power since it requires large DRAM capacity for its index table, and all of its DRAM channels are working at a high bandwidth due to the random access of tree roots; While CASA and GenAx only require a bandwidth that is less than 30 GB/s mainly for loading reads. CASA uses an efficient strategy to group CAM entries and only enables groups of CAM entries that contain the k-mer to be matched, together with a within groups technique to only enable the entries related to the automata transitions while performing multi-stride search, which allows it to consume only 4.2 % of the power compared to the naive implementation that enables all CAM entries. Moreover, CASA has 6.69× and 2.57× higher energy efficiency than GenAx and ERT since it provides the highest throughput with the lowest power consumption.

Table 4: Power and Area breakdown

Components	delay(ps)	area(mm^2)	power(W)
Pre-seeding controller	490	13.764	4.102
Computing controllers (total)	480	4.049	0.354
Pre-seeding filter table (45MB)	N/A	188.411	7.166
Computing CAMs (10MB)	N/A	90.329	6.949
DDR4 (total)	N/A	N/A	3.604
DRAM controller PHY	N/A	N/A	1.798

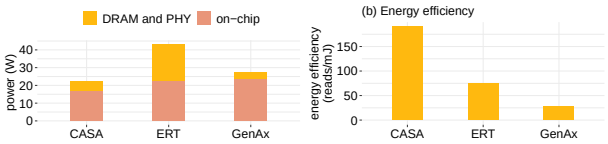

Figure 13: Comparison of (a) power consumption (W) and (b) energy efficiency (reads/mJ) between CASA, ERT, and GenAx.

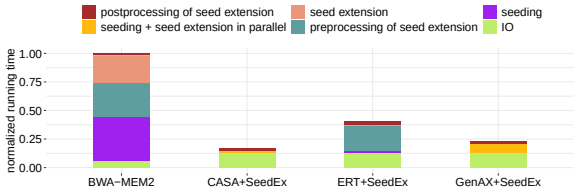

Figure 14: Comparison of the overall throughput between BWA-MEM2, ERT+SeedEx, GenAx+SeedEx, and CASA+SeedEx.

Table 4 shows the area breakdown of CASA. The main fraction of the area is devoted to the on-chip memory including CAMs and SRAMs, especially CAMs. Our 10T BCAM is large in area because 10T BCAM has nearly double the size of cells compared to 6T SRAM, and it contains extra peripherals for matching. In total, CASA takes a 296.553 mm^2 area at 28 nm, adding 33.9% compared to GenAx's 220.544 mm^2.

7.3 End-to-end comparison

We have evaluated the performance of CASA in terms of the end-to-end pipeline of genome analysis, which includes I/O (input reading, encoding and decoding of SAM files, etc.), seeding, the preprocessing of seed extension (suffix array lookup, chaining, etc.), seed extension, and the postprocessing of seed extension. We sent the seeds generated by CASA, GenAx, and ERT to 5 SeedEx lanes to catch up with the speed of the seed generation. Figure 14 shows the comparison of throughput among `CASA+SeedEx`, `GenAx+SeedEx`, `ERT+SeedEx`, and 12-thread BWA-MEM2, where `CASA+SeedEx` is 2.4×, 1.4×, and 6× faster than `ERT+SeedEx`, `GenAx+SeedEx`, and BWA-MEM2 respectively. The speed-up of CASA over GenAx mainly comes from the enhancement of seeding throughput as shown in Figure 12. Both `CASA+SeedEx` and `GenAx+SeedEx` can perform the computation of seeding and seed extension in parallel due to the on-chip reference, which allows them to directly fetch the hit of seeds on the reference. Moreover, for `CASA+SeedEx` and `GenAx+SeedEx`, the running time of preprocessing of seed extension is negligible since SMEMs generated by CASA and GenAx can be directly used in SeedEx for seed extension. In contrast, ERT does not have an on-chip reference and needs the CPU to perform the extra process on seeds and reference, such as chaining and packaging reads, reference, and metadata. Therefore, though Figure 12 does not show massive throughput speedup of seeding provided by CASA against ERT, `CASA+SeedEx`, in the end-to-end comparison, outperforms `ERT+SeedEx` by 2.4 times since it has a low overhead of preparing seeds for seed extension and allows to parallelize the execution of seeding and seed extension.

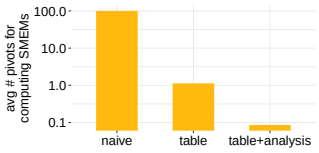

Figure 15: Comparison of the average number of pivots for SMEM computation per read for one reference partition (`naive`: the naive implementation of the SMEM searching algorithm; `table`: using only the pre-seeding filter table; `table+analysis`: both the pre-seeding filter table and the proposed heuristic analysis).

7.4 Impacts of Pre-Seeding Filtering and Inexact Matching Comparison

The naive implementation of the uni-directional SMEM search requires the computation of SMEMs for each pivot on the read, which is inefficient. Therefore, we have implemented a pre-seeding filter that performs analysis on each pivot and discards the pivot if the analysis determines the pivot cannot be the start index of a SMEM. The implementation of our pre-seeding filter consists of a pre-seeding filter table. We utilized this table to check the existence of the k-mer that starts at a pivot on the reference genome. If such a k-mer does not exist, we can directly discard the pivot. Moreover, as described in Section 4.2(c), we have provided a novel analysis to further algorithmically determine whether a pivot will lead to a MEM that is fully contained in a SMEM. With this analysis (i.e., optimization), we can further reduce the number of pivots that will trigger the SMEM searching computation.

Figure 15 shows the average number of pivots per read that triggers the SMEM computation for a partition of the reference genome (lower is better for seeding). We observe that our algorithms can greatly reduce the number of pivots that lead to SMEM computation, where, compared to `naive`, our `table` algorithm filters out 98.9 % of the pivots, and `table+analysis` filters out 99.9 %.

Figure 16 shows the comparison of the throughput performance of the inexact matching among CASA, ERT, and GenAx, where the throughput of CASA is 3.86x of GenAx and 0.72x of ERT. The speedup is not significant because the CAM introduces additional padding operations for each pivot and reduces the number of compute units to save energy.

8 RELATED WORKS

Seeding in Read Alignment. The seeding step of read alignment is a major bottleneck contributing around 40% to the overall execution time of BWA-MEM when aligning whole human genome reads. This is because BWA-MEM uses a compressed index structure called the FM-Index [37, 38], which causes random accesses to memory

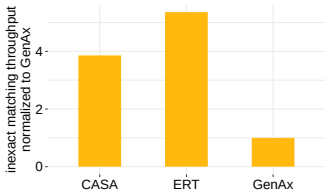

Figure 16: Comparison of the inexact matching throughput among CASA, ERT, and GenAx, where the throughput is normalized to GenAx.

and frequent cache misses on the CPU [6, 7]. Some prior works have explored performing multi-stride string matching [7, 26] and reordering memory accesses [69] to enhance the cache utilization of the FM-index. BWA-MEME [27] uses learned indexes to accelerate the computation of SMEMs, but leads to 120GB memory usage. Other works take an orthogonal way by using a hash table for seeding, which is usually coupled with filtration to accelerate read alignment [1, 2, 29]. However, their optimizations are less effective in FM-index aligners. GPUs provide high available memory bandwidth and data parallelism, which have also been leveraged to accelerate FM-index search [6, 42]. However, GPU-based solutions still face bandwidth underutilization as shown in [61]. Sieve[67] and MEDAL[23] explore the in/near DRAM way to perform seeding without heavy peripheral and provide impressive performance gain against CPU. However, they cannot support multi-stride search. Also, the access time of DRAM is typically more than 100 times higher than CAM.

Acceleration on SMEM Seeding. A recent accelerator is proposed for SMEM seeding using the Enumerated-Radix-Trees-index (i.e., ERT-index) [61]. The ERT-index provides better space locality and higher throughput compared with FM-index. As reported in [61], the FPGA implementation of the ERT accelerator improves bandwidth efficiency of BWA-MEM by 4.5 × and seeding throughput by 3.3 ×. Several other accelerators for SMEM seeding, such as GenAx [16] and GenCache [51], utilize a data structure with both seed and position tables. These accelerators improve the performance of seeding by leveraging on-chip memory bandwidth and data parallelism. GenPiP [46] and SaVI [34] utilize ReCAM to accelerate genome analysis and provide significant speedup compared to CPU. The limitation of their design is the compatibility with CMOS technology since they require large on-chip ReRAM. CASA performs the SMEM searches using CAM. CASA is evaluated in the 28 nm CMOS process, and it substantially improves the throughput and power efficiency of seeding compared to GenAx and ERT.

9 CONCLUSION

In conclusion, we propose CASA, a CAM-based SMEM seeding accelerator designed for efficient SMEM seeding. Existing state-of-the-art SMEM seeding accelerators either demand substantial DRAM bandwidth or suffer from excessive k-mer position fetches and intersections. CASA addresses these challenges by co-designing a hardware architecture and algorithms for SMEM pre-seeding filtering. As the result of the experimental evaluation, our ASIC design achieves a 1.2× and 5.47× throughput speedup compared to the state-of-the-art designs ERT and GenAx, while delivering 6.69× and 2.57× higher energy efficiency than GenAx and ERT in a 28 nm CMOS process. Moreover, the filter-enabled architecture of CASA, which supports large k-mer searches, broadens its applicability to long-read alignment, de novo assembly, and metagenomics classification. This feature effectively reduces the number of candidate positions, thereby enhancing efficiency.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their thoughtful comments. This work is supported in part by the Beijing Superstring Academy of Memory Technology, and in part by the National Natural Science Foundation of China (Grant No61834002, No62204139).

REFERENCES

[1] Mohammed Alser, Hasan Hassan, Akash Kumar, Onur Mutlu, and Can Alkan. 2019. Shouji: a fast and efficient pre-alignment filter for sequence alignment. *Bioinformatics* 35, 21 (2019), 4255–4263. https://doi.org/10.1093/bioinformatics/btz234

[2] Mohammed Alser, Taha Shahroodi, Juan Gómez-Luna, Can Alkan, and Onur Mutlu. 2021. SneakySnake: a fast and accurate universal genome pre-alignment filter for CPUs, GPUs and FPGAs. *Bioinformatics* 36, 22-23 (April 2021), 5282–5290. https://doi.org/10.1093/bioinformatics/btaa1015

[3] Shaahin Angizi, Naima Ahmed Fahmi, Wei Zhang, and Deliang Fan. 2020. PIM-Assembler: A Processing-in-Memory Platform for Genome Assembly. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC18072.2020.9218653

[4] Robert S. Boyer and J. Strother Moore. 1977. A Fast String Searching Algorithm. *Commun. ACM* 20, 10 (oct 1977), 762–772. https://doi.org/10.1145/359842.359859

[5] BWA-MEM2 ERT 2022. https://github.com/bwa-mem2/bwa-mem2/tree/ert.

[6] Alejandro Chacón, Santiago Marco-Sola, Antonio Espinosa, Paolo Ribeca, and Juan Carlos Moure. 2015. Boosting the FM-Index on the GPU: Effective Techniques to Mitigate Random Memory Access. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 12, 5 (2015), 1048–1059. https://doi.org/10.1109/TCBB.2014.2377716

[7] Alejandro Chacón, Juan Carlos Moure, Antonio Espinosa, and Porfidio Hernández. 2013. n-step FM-Index for Faster Pattern Matching. In *ICCS*.

[8] Mark Chaisson, Pavel Pevzner, and Haixu Tang. 2004. Fragment assembly with short reads. *Bioinformatics* 20, 13 (2004), 2067–2074. https://doi.org/doi.org/10.1093/bioinformatics/bth205

[9] Karthik Chandrasekar, Christian Weis, Yonghui Li, Benny Akesson, Norbert Wehn, and Kees Goossens. 2022. DRAMPower: Open-source DRAM Power & Energy Estimation Tool. http://www.drampower.info

[10] Kun-Mao Chao, William R. Pearson, and Webb Miller. 1992. Aligning two sequences within a specified diagonal band. *Bioinformatics* 8, 5 (10 1992), 481–487. https://doi.org/10.1093/bioinformatics/8.5.481

[11] Woojun Choi, Taewoong Kim, Jongjoo Shim, Hyungsoo Kim, Gunhee Han, and Youngcheol Chae. 2017. 23.8 A 1V 7.8mW 15.6Gb/s C-PHY transceiver using tri-level signaling for post-LPDDR4. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, New York, USA, 402–403. https://doi.org/10.1109/ISSCC.2017.7870431

[12] Jason Cong, Licheng Guo, Po-Tsang Huang, Peng Wei, and Tianhe Yu. 2018. SMEM++: A Pipelined and Time-Multiplexed SMEM Seeding Accelerator for DNA Sequencing. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 206–206. https://doi.org/10.1109/FCCM.2018.00040

[13] dwgsim 2022. Whole Genome Simulator for Next-Generation Sequencing. https://github.com/nh13/DWGSIM

[14] Michael A. Eberle, Epameinondas Fritzilas, Peter Krusche, Morten Källberg, Benjamin L Moore, Mitchell A Bekritsky, Zamin Iqbal, Han-Yu Chuang, Sean J. Humphray, Aaron L. Halpern, Semyon Kruglyak, Elliott H. Margulies, Gil McVean, and D. R. Bentley. 2016. A reference dataset of 5.4 million phased human variants validated by genetic inheritance from sequencing a three-generation 17-member pedigree. *bioRxiv* (2016).

[15] Elakkiya Elumalai and Krishna Kant Gupta. 2021. *High-Throughput Sequencing Technologies*. Springer, Singapore, 283–304. https://doi.org/10.1007/978-981-16-3993-7_13

[16] Daichi Fujiki, Arun Subramaniyan, Tianjun Zhang, Yu Zeng, Reetuparna Das, David Blaauw, and Satish Narayanasamy. 2018. GenAx: A Genome Sequencing Accelerator. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, Los Angeles, CA, 69–82. https://doi.org/10.1109/ISCA.2018.00017

[17] Daichi Fujiki, Shunhao Wu, Nathan Ozog, Kush Goliya, David Blaauw, Satish Narayanasamy, and Reetuparna Das. 2020. SeedEx: A Genome Sequencing Accelerator for Optimal Alignments in Subminimal Space. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 937–950. https://doi.org/10.1109/MICRO50266.2020.00080

[18] gatk 2022. GATK4 Genome Processing Pipeline. https://github.com/gatk-workflows/gatk4-genome-processing-pipeline

[19] Saransh Gupta, Mohsen Imani, Behnam Khaleghi, Venkatesh Kumar, and Tajana Rosing. 2019. RAPID: A ReRAM Processing in-Memory Architecture for DNA Sequence Alignment. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. 1–6. https://doi.org/10.1109/ISLPED.2019.8824830

[20] D. S. Hirschberg. 1975. A linear space algorithm for computing maximal common subsequences. *Commun. ACM* 18, 6 (June 1975), 341–343. https://doi.org/10.1145/360825.360861

[21] Darryl Ho, Jialin Ding, Sanchit Misra, Nesime Tatbul, Vikram Nathan, Vasimuddin, and Tim Kraska. 2019. LISA: Towards Learned DNA Sequence Search. *ArXiv* abs/1910.04728 (2019).

[22] Wenqin Huangfu, Shuangchen Li, Xing Hu, and Yuan Xie. 2018. RADAR: A 3D-ReRAM based DNA Alignment Accelerator Architecture. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC.2018.8465882

[23] Wenqin Huangfu, Xueqi Li, Shuangchen Li, Xing Hu, Peng Gu, and Yuan Xie. 2019. MEDAL: Scalable DIMM Based Near Data Processing Accelerator for DNA Seeding Algorithm. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 587–599. https://doi.org/10.1145/3352460.3358329

[24] Tommy Tracy Ii, Mircea Stan, Nathan Brunelle, Jack Wadden, Ke Wang, and Gabriel Robins. 2015. Nondeterministic Finite Automata in Hardware - the Case of the Levenshtein Automaton. *Architectures and Systems for Big Data, in conjunction with ISCA* (2015).

[25] illumina 2022. Illumina @ A Glance. https://www.illumina.com/company/news-center/feature-articles/illumina-at-a-glance.html

[26] Lei Jiang and Farzaneh Zokaee. 2021. EXMA: A Genomics Accelerator for Exact-Matching. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 399–411. https://doi.org/10.1109/HPCA51647.2021.00041

[27] Youngmok Jung and Dongsu Han. 2022. BWA-MEME: BWA-MEM emulated with a machine learning approach. *Bioinformatics* 38, 9 (03 2022), 2404–2413. https://doi.org/10.1093/bioinformatics/btac137

[28] Daehwan Kim, Li Song, Florian P Breitwieser, and Steven L Salzberg. 2016. Centrifuge: rapid and sensitive classification of metagenomic sequences. *Genome research* 26, 12 (December 2016), 1721—1729. https://doi.org/10.1101/gr.210641.116

[29] Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu. 2018. GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies. *BMC Genomics* 19, S2 (May 2018), 89. https://doi.org/10.1186/s12864-018-4460-0

[30] Yeseong Kim, Mohsen Imani, Niema Moshiri, and Tajana Rosing. 2020. GenieHD: Efficient DNA Pattern Matching Accelerator Using Hyperdimensional Computing. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, New York, USA, 115–120. https://doi.org/10.23919/DATE48585.2020.9116397

[31] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2016. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Computer Architecture Letters* 15, 1 (2016), 45–49. https://doi.org/10.1109/LCA.2015.2414456

[32] Donald Ervin Knuth, James H. Morris, and Vaughan R. Pratt. 1977. Fast Pattern Matching in Strings. *SIAM J. Comput.* 6 (1977), 323–350.

[33] Hyeongjun Ko, Mino Kim, Hyunkyu Park, Sangyoon Lee, Jaewook Kim, Suhwan Kim, and Joo-Hyung Chae. 2021. A Controller PHY for Managed DRAM Solution With Damping-Resistor-Aided Pulse-Based Feed-Forward Equalizer. *IEEE Journal of Solid-State Circuits* 56, 8 (2021), 2563–2573. https://doi.org/10.1109/JSSC.2021.3062876

[34] Ann Franchesca Laguna, Hasindu Gamaarachchi, Xunzhao Yin, Michael Niemier, Sri Parameswaran, and X. Sharon Hu. 2020. Seed-and-Vote Based in-Memory Accelerator for DNA Read Mapping. In *Proceedings of the 39th International Conference on Computer-Aided Design (ICCAD '20)*. Association for Computing Machinery, New York, NY, USA, Article 56, 9 pages. https://doi.org/10.1145/3400302.3415651

[35] Ruben Langarita, Adria Armejach, Javier Setoain, Pablo Ibanez-Marin, Jesus Alastruey-Benede, and Miquel Moreto. 2022. Compressed Sparse FM-Index: Fast Sequence Alignment Using Large K-Steps. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 19, 1 (Jan. 2022), 355–368. https://doi.org/10.1109/TCBB.2020.3000253

[36] Ben Langmead and Steven L. Salzberg. 2012. Fast gapped-read alignment with Bowtie 2. *Nature Methods* 9 (2012), 357–359.

[37] Heng Li. 2012. Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly. *Bioinformatics* 28, 14 (July 2012), 1838–1844. https://doi.org/10.1093/bioinformatics/bts280

[38] Heng Li. 2013. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM.

[39] Heng Li. 2016. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics* 32, 14 (July 2016), 2103–2110. https://doi.org/10.1093/bioinformatics/btw152

[40] Heng Li. 2018. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* 34, 18 (Sept. 2018), 3094–3100. https://doi.org/10.1093/bioinformatics/bty191

[41] Hengyun Lu, Francesca Giordano, and Zemin Ning. 2016. Oxford Nanopore MinION Sequencing and Genome Assembly. *Genomics, Proteomics & Bioinformatics* 14, 5 (2016), 265–279. https://doi.org/10.1016/j.gpb.2016.05.004

[42] Ruibang Luo, Thomas Wong, Jianqiao Zhu, Chi-Man Liu, Xiaoqian Zhu, Edward Wu, Lap-Kei Lee, Haoxiang Lin, Wenjuan Zhu, David W. Cheung, Hing-Fung Ting, Siu-Ming Yiu, Shaoliang Peng, Chang Yu, Yingrui Li, Ruiqiang Li, and Tak-Wah Lam. 2013. SOAP3-dp: Fast, Accurate and Sensitive GPU-Based Short Read Aligner. *PLOS ONE* 8, 5 (05 2013), 1–11. https://doi.org/10.1371/journal.pone.0065632

[43] Alberto Magi, Roberto Semeraro, Alessandra Mingrino, Betti Giusti, and Romina D'Aurizio. 2017. Nanopore sequencing data analysis: state of the art, applications and challenges. *Briefings in Bioinformatics* 19, 6 (06 2017), 1256–1272. https://doi.org/10.1093/bib/bbx062

[44] Alberto Magi, Roberto Semeraro, Alessandra Mingrino, Betti Giusti, and Romina D'Aurizio. 2017. Nanopore sequencing data analysis: state of the art, applications and challenges. *Briefings in Bioinformatics* 19, 6 (06 2017), 1256–1272. https://doi.org/10.1093/bib/bbx062

[45] Nika Mansouri Ghiasi, Jisung Park, Harun Mustafa, Jeremie Kim, Ataberk Olgun, Arvid Gollwitzer, Damla Senol Cali, Can Firtina, Haiyu Mao, Nour Almadhoun Alserr, Rachata Ausavarungnirun, Nandita Vijaykumar, Mohammed Alser, and Onur Mutlu. 2022. GenStore: A High-Performance in-Storage Processing System for Genome Sequence Analysis. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. ACM, New York, USA, 635–654. https://doi.org/10.1145/3503222.3507702

[46] Haiyu Mao, Mohammed Alser, Mohammad Sadrosadati, Can Firtina, Akanksha Baranwal, Damla Senol Cali, Aditya Manglik, Nour Almadhoun Alserr, and Onur Mutlu. 2022. GenPIP: In-Memory Acceleration of Genome Analysis via Tight Integration of Basecalling and Read Mapping. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 710–726. https://doi.org/10.1109/MICRO56248.2022.00056

[47] Micron. 2021. 4Gb: x8, x16 Automotive DDR4 SDRAM. (2021). Accessed 19 November 2022. https://www.micron.com/products/dram/ddr4-sdram/part-catalog/mt40a256m16ly-062e-aat.

[48] Micron cal 2022. Calculating Memory Power for DDR4 SDRAM. https://media-www.micron.com/-/media/client/global/documents/products/technical-note/dram/tn4007_ddr4_power_calculation.pdf

[49] Eugene W. Myers and Webb Miller. 1988. Optimal alignments in linear space. *Bioinformatics* 4, 1 (1988), 11–17. https://doi.org/10.1093/bioinformatics/4.1.11

[50] Gene Myers. 1999. A Fast Bit-Vector Algorithm for Approximate String Matching Based on Dynamic Programming. *J. ACM* 46, 3 (may 1999), 395–415. https://doi.org/10.1145/316542.316550

[51] Anirban Nag, C. N. Ramachandra, Rajeev Balasubramonian, Ryan Stutsman, Edouard Giacomin, Hari Kambalasubramanyam, and Pierre-Emmanuel Gaillardon. 2019. GenCache: Leveraging In-Cache Operators for Efficient Sequence Alignment. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, Columbus OH USA, 334–346. https://doi.org/10.1145/3352460.3358308

[52] novaseq 2022. NovaSeq 6000 System. https://www.illumina.com/systems/sequencing-platforms/novaseq.html

[53] K. Pagiamtzis and A. Sheikholeslami. 2006. Content-addressable memory (CAM) circuits and architectures: a tutorial and survey. *IEEE Journal of Solid-State Circuits* 41, 3 (2006), 712–727. https://doi.org/10.1109/JSSC.2005.864128

[54] William R. Pearson. 1995. Comparison of methods for searching protein sequence databases. *Protein Science* 4, 6 (1995), 1145–1160. https://doi.org/10.1002/pro.5560040613

[55] William R. Pearson and Webb Miller. 1992. Dynamic programming algorithms for biological sequence comparison. In *Numerical Computer Methods*. Methods in Enzymology, Vol. 210. Academic Press, 575–601. https://doi.org/10.1016/0076-6879(92)10029-D

[56] Rahul M. Rao, Christopher Gonzalez, Eric Fluhr, Abraham Mathews, Andrew Bianchi, Daniel Dreps, David Wolpert, Eric Lai, Gerald Strevig, Glen Wiedemeier, Philipp Salz, and Ryan Kruse. 2022. POWER10™: A 16-Core SMT8 Server Processor With 2TB/s Off-Chip Bandwidth in 7nm Technology. In *2022 IEEE International Solid- State Circuits Conference (ISSCC)*, Vol. 65. IEEE, New York, USA, 48–50. https://doi.org/10.1109/ISSCC42614.2022.9731594

[57] Richard J. Roberts, Mauricio O. Carneiro, and Michael C. Schatz. 2013. The advantages of SMRT sequencing. *Genome Biology* 14 (2013), 405 – 405.

[58] Damla Senol Cali, Jeremie S Kim, Saugata Ghose, Can Alkan, and Onur Mutlu. 2018. Nanopore sequencing technology and tools for genome assembly: computational analysis of the current state, bottlenecks and future directions. *Briefings in Bioinformatics* 20, 4 (04 2018), 1542–1559. https://doi.org/10.1093/bib/bby017

[59] T.F. Smith and M.S. Waterman. 1981. Identification of common molecular subsequences. *Journal of Molecular Biology* 147, 1 (March 1981), 195–197. https://doi.org/10.1016/0022-2836(81)90087-5

[60] Arun Subramaniyan, Yufeng Gu, Timothy Dunn, Somnath Paul, Md Vasimuddin, Sanchit Misra, David Blaauw, Satish Narayanasamy, and Reetuparna Das. 2021. GenomicsBench: A Benchmark Suite for Genomics. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Stony Brook, NY, USA, 1–12. https://doi.org/10.1109/ISPASS51385.2021.00012

[61] Arun Subramaniyan, Jack Wadden, Kush Goliya, Nathan Ozog, Xiao Wu, Satish Narayanasamy, David Blaauw, and Reetuparna Das. 2021. Accelerated Seeding for Genome Sequence Alignment with Enumerated Radix Trees. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, New York, USA, 388–401. https://doi.org/10.1109/ISCA52012.2021.00038

[62] Hélène Touzet. 2016. On the Levenshtein Automaton and the Size of the Neighbourhood of a Word. In *Language and Automata Theory and Applications*. Vol. 9618. Springer International Publishing, Cham, 207–218. https://doi.org/10.1007/978-3-319-30000-9_16 Series Title: Lecture Notes in Computer Science.

[63] Yatish Turakhia, Gill Bejerano, and William J. Dally. 2018. Darwin: A Genomics Co-processor Provides up to 15,000X Acceleration on Long Read Assembly. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, Williamsburg VA USA, 199–213. https://doi.org/10.1145/3173162.3173193

[64] UCB 2022. USCS Genome Browser. https://genome.ucsc.edu/.

[65] Erwin L. van Dijk, Hélène Auger, Yan Jaszczyszyn, and Claude Thermes. 2014. Ten years of next-generation sequencing technology. *Trends in Genetics* 30, 9 (2014), 418–426. https://doi.org/10.1016/j.tig.2014.07.001

[66] Md. Vasimuddin, Sanchit Misra, Heng Li, and Srinivas Aluru. 2019. Efficient Architecture-Aware Acceleration of BWA-MEM for Multicore Systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 314–324. https://doi.org/10.1109/IPDPS.2019.00041

[67] Lingxi Wu, Rasool Sharifi, Marzieh Lenjani, Kevin Skadron, and Ashish Venkat. 2021. Sieve: Scalable In-situ DRAM-based Accelerator Designs for Massively Parallel k-mer Matching. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 251–264. https://doi.org/10.1109/ISCA52012.2021.00028

[68] Cheng-Xin Xue, Wei-Cheng Zhao, Tzu-Hsien Yang, Yi-Ju Chen, Hiroyuki Yamauchi, and Meng-Fan Chang. 2019. A 28-nm 320-kb TCAM macro using split-controlled single-load 14T cell and triple-margin voltage sense amplifier. *IEEE Journal of Solid-State Circuits* 54, 10 (2019), 2743–2753.

[69] Jing Zhang, Heshan Lin, Pavan Balaji, and Wu-Chun Feng. 2013. Optimizing Burrows-Wheeler Transform-Based Sequence Alignment on Multicore Architectures. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. 377–384. https://doi.org/10.1109/CCGrid.2013.67

[70] Zheng Zhang, Scott Schwartz, Lukas Wagner, and Webb Miller. 2000. A greedy algorithm for aligning DNA sequences. *Journal of Computational biology* 7, 1-2 (2000), 203–214. https://doi.org/10.1089/10665270050081478

[71] Farzaneh Zokaee, Mingzhe Zhang, and Lei Jiang. 2019. FindeR: Accelerating FM-Index-Based Exact Pattern Matching in Genomic Sequences through ReRAM Technology. In *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 284–295. https://doi.org/10.1109/PACT.2019.00030

[72] Zhuowen Zou, Hanning Chen, Prathyush Poduval, Yeseong Kim, Mahdi Imani, Elaheh Sadredini, Rosario Cammarota, and Mohsen Imani. 2022. BioHD: an efficient genome sequence search platform using HyperDimensional memorization. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. ACM, New York, USA, 656–669. https://doi.org/10.1145/3470496.3527422