BioHD: An Efficient Genome Sequence Search Platform Using HyperDimensional Memorization

Zhuowen Zou[1], Hanning Chen[1], Prathyush Poduval[2], Yeseong Kim[3], Mahdi Imani[4], Elaheh Sadredini[5], Rosario Cammarota[6] and Mohsen Imani[1]*

[1]University of California Irvine, [2]Indian Institute of Science, [3]Daegu Gyeongbuk Institute of Science and Technology, [4]Northeastern University, [5]University of California Riverside, [6]Intel Labs

* Corresponding author: m.imani@uci.com

ABSTRACT

In this paper, we propose BioHD, a novel genomic sequence searching platform based on Hyper-Dimensional Computing (HDC) for hardware-friendly computation. BioHD transforms inherent sequential processes of genome matching to highly-parallelizable computation tasks. We exploit HDC memorization to encode and represent the genome sequences using high-dimensional vectors. Then, it combines the genome sequences to generate an HDC reference library. During the sequence searching, BioHD performs exact or approximate similarity check of an encoded query with the HDC reference library. Our framework simplifies the required sequence matching operations while introducing a statistical model to control the alignment quality. To get actual advantage from BioHD inherent robustness and parallelism, we design a processing in-memory (PIM) architecture with massive parallelism and compatible with the existing crossbar memory. Our PIM architecture supports all essential BioHD operations natively in memory with minimal modification on the array. We evaluate BioHD accuracy and efficiency on a wide range of genomics data, including COVID-19 databases. Our results indicate that PIM provides 102.8× and 116.1× (9.3× and 13.2×) speedup and energy efficiency compared to the state-of-the-art pattern matching algorithm running on GeForce RTX 3060 Ti GPU (state-of-the-art PIM accelerator).

ACM Reference Format:

Zhuowen Zou[1], Hanning Chen[1], Prathyush Poduval[2], Yeseong Kim[3], Mahdi Imani[4], Elaheh Sadredini[5], Rosario Cammarota[6] and Mohsen Imani[1]*. 2022. BioHD: An Efficient Genome Sequence Search Platform Using HyperDimensional Memorization. In *The 49th Annual International Symposium on Computer Architecture (ISCA '22), June 18–22, 2022, New York, NY, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3470496.3527422

1 INTRODUCTION

In December 2019, a series of pneumonia cases of unknown cause were documented. Deep sequencing of lower respiratory tract samples indicated a novel coronavirus [1–3] and the corresponding disease "COVID-19". Within the last few months, the global number of cases and deaths exceeded 413 million and 5.8 million, respectively [4]. The number of available COVID-19 genome assemblies is growing rapidly which in turn creates a need for scalable methods to enable real-time analysis and dissemination [5–7].

Unfortunately, the optimal solution for the sequence alignment or sequence matching scales poorly with the number of sequences. An underlying reason is that data movement costs between the processor and memory still hinder the higher efficiency of application performance, although new processor technology has evolved to serve computationally complex tasks more efficiently. Processing in-memory (PIM) is a promising solution to accelerate applications with a large amount of parallelism [8–19]. Several recent works have explored the advantage of PIM-based architectures to accelerate machine learning algorithms [20, 21]. However, there are several main challenges in using existing PIM architectures to accelerate the sequence matching: (i) the existing PIM architectures are mostly a dot product engine and accelerate vector-matrix multiplication. In contrast, sequence matching involves pairwise distance computation and similarity search, which cannot be supported entirely by existing PIM architectures [22, 23]. (ii) Most PIM architectures are analog-based [14, 20, 21]; thus, they perform computation after mapping data points into analog space. The data converter between analog and digital space takes the majority of chip area and energy [21]. (iii) The existing PIM architectures require separate storage and computing memory units, resulting in a large amount of internal data movements. This not only reduces the computation efficiency but also affects the design scalability. (iv) The emerging devices, i.e., Non-Volatile Memories (NVMs), have various reliability issues such as endurance, durability, and variability [24–26]. This, coupled with the high computation precision required for sequence matching algorithms, limits the applicability of the PIM accelerators.

In this paper, we present a novel strategy that effectively adopts the PIM architecture for the sequence matching problem. At heart, an sequence matching solution requires to perform multiple *search* as a fundamental procedure, i.e., checking the existence of a gene series in a database [27–34]. In this sense, the problem is equivalent to memorization in that we should memorize and recall genomic/proteomic sequences. We accelerate the sequence searching in hardware by redesigning the sequence searching based on a new computing paradigm, Hyper-Dimensional Computing (HDC) [35–38]. HDC is a human memory-inspired method to implement efficient memorization using high-dimensional vectors, called hypervectors. The HDC provides several features that make it well-suited to address the alignment problem: (i) it transforms inherent sequential processes of the sequence searching to highly-parallelizable

ISCA '22, June 18–22, 2022, New York, NY, USA

ACM ISBN 978-1-4503-8610-4/22/06…$15.00
https://doi.org/10.1145/3470496.3527422

computation tasks, where the operations can be natively supported by PIM, (ii) BioHD is memory-centric and highly-parallel, making PIM architecture an ideal platform for hardware acceleration, (iii) it provides strong robustness to noise – a key strength for emerging technologies.

We propose BioHD, a Hyper-Dimensional genome analysis platform. Instead of working with original sequences, BioHD maps the genome sequences into high-dimensional space and performs sequence matching with simple and parallel similarity searches. The main contributions of the paper are summarized as follows:

- To the best of our knowledge, **BioHD is the first end-to-end PIM platform for sequence searching based on Hyper-Dimensional computing.** At algorithm-level, BioHD revisits the sequence searching with brain-like memorization that HDC natively supports. BioHD creates a library for reference sequences by memorizing the patterns in high-dimension.
- **BioHD simplifies the alignment by recalling the exact or approximate existence of a query sequence with the reference library.** Instead of working on the original data, BioHD maps all data points into high-dimensional space, enabling the main sequence searching operations **to process in a hardware-friendly way.** BioHD encoding preserves the similarity of sequences and simplifies the similarity metric to Hamming distance. We also proposed a statistical model that provides a theoretical control of the quality of the alignment.
- **We propose a novel PIM architecture that accelerates BioHD on the existing crossbar memory with essential alignment operations implemented in memory blocks.** This includes supporting search-based distance computing as well as row-parallel arithmetic in memory. The enhanced crossbar array can be used in the pipeline to accelerate BioHD in a massively parallel way.

We evaluate BioHD efficiency on a wide range of practical problems, including the COVID-19 database. Our solution provides, on average, 124.1× and 102.8× and 116.1× (9.3× and 13.2×) speedup and energy efficiency improvement compared to the state-of-the-art alignment algorithm running on GeForce RTX 3060 Ti GPU (state-of-the-art PIM accelerator [39]).

2 BACKGROUND

Genome Sequence search: The process of predicting the possible DNA/RNA sequence that a specific protein has originated from is called back-translation. Aligning the back-translated RNA sequence against the database locates the most similar sequences used to predict the functionality of the unknown protein sequence. Proteins are made up of one or more chains of 20 common amino acids. An unknown protein can be characterized when its sequence shares significant similarity with a protein with known characteristics [28–31, 40–43]. Figure 1a describes the overview of BioHD computational task flow. From a computational standpoint, a protein sequence can be considered a string over a set that includes different amino acid letters, e.g., $S = \langle \texttt{Met}, \texttt{Phe}, \cdots \rangle$. The protein sequence is originally translated from an mRNA, which can be considered a string over the four alphabets, {A,C,G,U}. Each non-overlapping three-letter window of the mRNA, known as a Codon, encodes a specific letter in the amino acid alphabet according to the

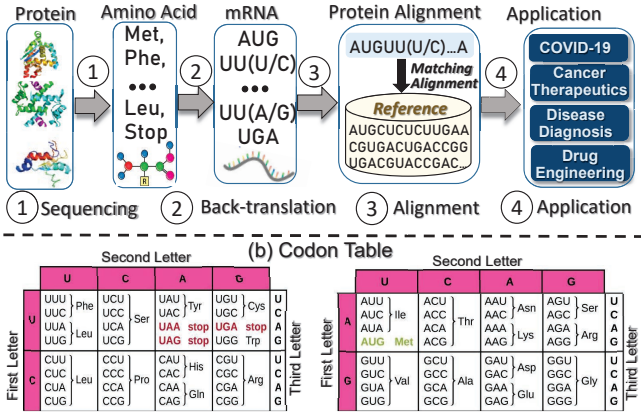

Figure 1: (a) Protein back-translation and alignment flow, (b) RNA translation to amino acids (Codon table).

Codon table (Figure 1b). The protein sequence is the result of replacing each codon of the mRNA sequence from start to end with its corresponding amino acid. The task of "back-translation" uses the Codon table to generate an mRNA sequence representing the most likely non-degenerate coding sequence. Since the back-translation in most cases does not yield a unique result, a consensus sequence derived from all possible Codons is needed. The sequence searching finds the regions with high similarity to the query sequence. The regions with high similarity, called hits, are used in many applications to predict the functionality of the unknown query.

Hyper-dimensional computing (HDC): HDC effectively mimics several important functionalities of the human memory and allows energy-efficient computation based on its massively parallel computation flow [35, 37, 44–46]. HDC is motivated by an observation that the human brain operates on a *robust high-dimensional* representation of data due to the large size of brain circuits [47–49]. HDC mimics the properties based on the idea that we can represent the information with a hypervector and the correlation with the hyperspace's distance. The dimensionality should be large enough to ensure two D-components vectors that are randomly chosen from $\{-1, 1\}$ or {0,1}, are *near-orthogonal*, referring that the similarity in the vector space is almost zero. As a result, HDC applications use hypervectors that have thousands dimensionality, e.g., $D = 10,000$.

Let us assume $\vec{A}$ and $\vec{B}$ are two random generated hypervectors in high-dimensional space. HDC encoding works based on a well-defined set of operations: *(i) Bundling* (+) is an addition or majority operation of multiple hypervectors into a single reference hypervector ($\vec{R} = \vec{A} + \vec{B}$). The result of the bundling preserves similarity to its component hypervectors i.e., $\delta(\vec{R}, \vec{A}) >> 0$, where δ is a cosine similarity. The bundling operation is well suited for representing sets. *(ii) Binding* (*) associates multiple orthogonal hypervectors (e.g., $\vec{A}$, $\vec{B}$) into a single hypervector ($\vec{R} = \vec{A} * \vec{B}$). Binding is done by the component-wise multiplication (XOR in the binary representation) between two hypervectors. Binding is used for variable-value association and, more generally, for the mapping. *(iii) Permutation* (ρ) defines as a single rotational shift. The permutation operation generates a hypervector, which is unrelated to the

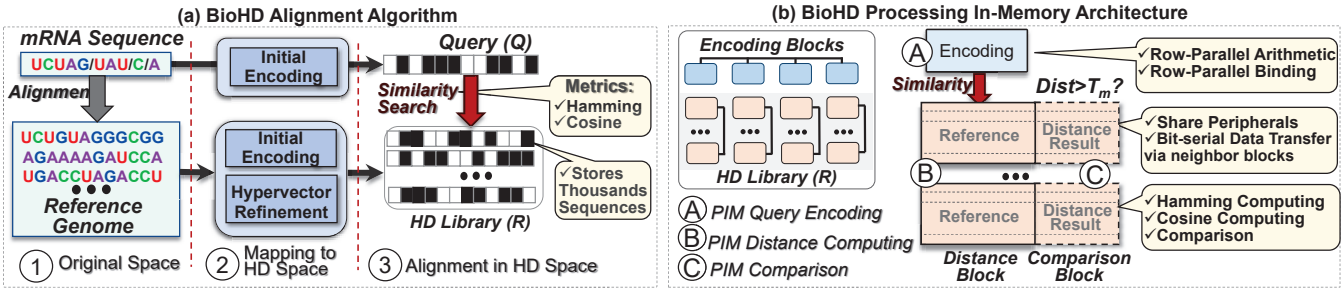

Figure 2: Overview of (a) BioHD alignment in high-dimensional space, (b) the proposed PIM accelerator.

given hypervector ($\delta(\vec{A}, \rho\vec{A}) \simeq 0$). It is commonly used for storing a sequence of tokens in a single hypervector.

3 BIOHD: GENOME SEQUENCE SEARCH

Figure 2 shows an overview of BioHD sequence search in the high-dimensional space. The first step of BioHD is to map the genome sequence into a high-dimensional space. BioHD assigns a hypervector corresponding to each base alphabet in $\Sigma = \{A, C, G, T\}$ for DNA and $\Sigma = \{A, C, G, U\}$ for RNA. The encoding module depends on the data type and the genomics task. In terms of protein data, BioHD assigns a hypervector representing each RNA bases and then combines them to create a hypervector representing each amino acid (as explained in Section 2, and Figure 1). The amino acids' hypervectors are combined by mapping each protein sequence into a high-dimensional space (explained in Section 3.1). BioHD aggregates all encoded protein sequences to generate a reference genome, called *HDC Library*. An HDC library consists of several reference hypervectors, where each hypervector memorizes thousands of genome sequences in high-dimensional space. Similar to the human memorization that requires practice, BioHD iteratively checks the correctness of memorized information in each library hypervector to find the most refined hypervectors. During the sequence searching, BioHD uses the same encoding to map a query sequence into a hypervector. We perform a similarity computation between a query and each reference hypervector. By searching for an exact or approximate match, BioHD identifies a query's closeness with thousands of memorized patterns stored in each HDC library hypervector.

We design a novel processing in-memory (PIM) architecture that accelerates BioHD with minor modification on existing crossbar memory (Section 4). Our approach is general and can work with any bipolar non-volatile memory (NVM) devices. Each memory block in PIM architecture supports the essential alignment operations, including row-parallel distance computing and vector-based binding operation (Section 5). These PIM functionalities are implemented over digital data stored in memory using search-based and row-parallel arithmetic computation. For scalability, BioHD architecture enables row-parallel bit-serial data transfer between neighbor memory blocks. Figure 2b shows an example of PIM functionality. In the first step, BioHD encodes the coming genome sequence using the encoding block. Next, BioHD checks the similarity of the encoded query with all reference hypervectors stored in memory in a massively parallel way. In each block, we support the similarity measurement using row-parallel Hamming distance and dot-product distance metrics. Finally, the search result will be written to the comparison block, and this block will identify reference hypervectors that have been closely matched.

3.1 BioHD Genome Sequence Encoding

The entire BioHD computation happens after mapping genomics data (DNA, RNA, or protein) into high-dimensional space. To encode a sequence to a hypervector, BioHD assigns a hypervector corresponding to each base alphabet in $\Sigma = \{A, C, G, T\}$ for DNA and $\Sigma = \{A, C, G, U\}$ for RNA. We call hypervectors as *base hypervectors*, and denote them with $\Sigma_{HV} = \{\vec{A}, \vec{C}, \vec{G}, \vec{T}\}$ and $\Sigma_{HV} = \{\vec{A}, \vec{C}, \vec{G}, \vec{U}\}$. These bases are randomly generated, thus they are nearly orthogonal. To be more precise, their dot products, an indicator of similarity, follow a certain distribution: $\vec{A} \cdot \vec{C} \sim 2Bin(D, 0.5) - D$, where $Bin(D, 0.5)$ denotes the binomial distribution with parameter $n = D$, the HDC dimension, and $p = 0.5$. (Figure 3❶).

Protein Encoding: BioHD encodes each protein sequence into high-dimensional space by (i) encoding the mRNA sequences into high-dimensional space, (ii) combining multiple mRNA hypervectors to generate a hypervector for each amino acid (Figure 1b), and (iii) combining the amino acids sequences to create a hypervector for each protein sequence.

(i) mRNA Encoding: BioHD encodes each DNA or mRNA sequence by binding its corresponding base hypervectors. Let us consider a short query string, 'ACGU'. BioHD encodes the sequence by binding the base hypervectors into high-dimensional space (Figure 3❷). $\vec{H} = \vec{A} * \rho^1(\vec{C}) * \rho^2(\vec{G}) * \rho^3(\vec{U})$, where ρ^n represents n-bit(s) rotational shift. The independence among the base hypervectors and their permuted forms ensures that the generated hypervectors for different RNA sequences are nearly orthogonal. Given $\vec{H_1}$ and $\vec{H_2}$ generated from different sequences, $\vec{H_1} \cdot \vec{H_2} \sim 2Bin(D, 0.5) - D$.

(ii) Amino Acid Encoding: As the Codon table shows, each amino acid may correspond to multiple mRNA sequences. This makes the alignment problem more complicated as we need to consider all possibilities that a protein sequence may be back-translated to. The amino acid hypervector needs to memorize the information of multiple mRNA hypervectors. We explain amino acid encoding in an example. Based on Codon table, Phe amino acid corresponds to UUU and UUC sequences. Our encoding represents this amino acid as:

$$\vec{H}_{Phe} = (\vec{U} * \rho^1\vec{U} * \rho^2\vec{U}) \frown (\vec{U} * \rho^1\vec{U} * \rho^2\vec{C})$$

where the term '$\frown$' indicates the probabilistic merging. This approach randomly samples half of the dimensions from the first and second hypervectors and generates a new hypervector with half

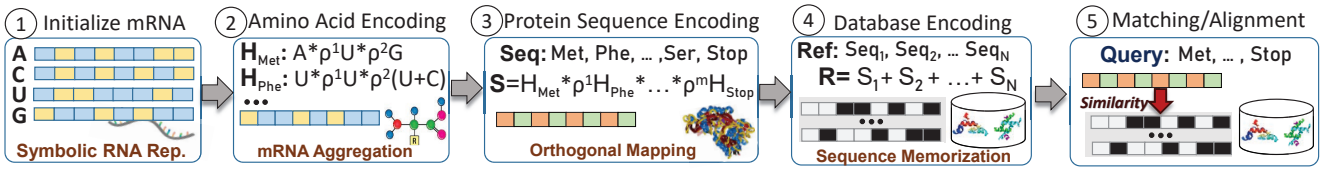

Figure 3: BioHD protein alignment steps.

similarity to each generated bases. This encoding preserves the similarity while representing each amino acid as a single binary hypervector (Figure 3❷). This compact representation significantly reduces the noise and the complexity of the BioHD alignment.

(iii) Protein Encoding: Protein is a sequence of amino acids. BioHD exploits the generated amino acid hypervectors to encode different protein sequences. For the sequence searching, the encoding remembers each amino acid and its corresponding position in a sequence. For example, protein sequence of $\mathrm{S} = \langle \mathtt{Met}, \mathtt{Phe}, \mathtt{Ser}, \mathtt{Gly}, \mathtt{Stop} \rangle$ can be decoded as:

$$\vec{\mathrm{S}} = \vec{\mathrm{H}}_{Met} * \rho\vec{\mathrm{H}}_{Phe} * \rho^2\vec{\mathrm{H}}_{Ser} * \rho^3\vec{\mathrm{H}}_{Gly} * \rho^4\vec{\mathrm{H}}_{Stop}$$

(iv) Insertion/Deletion: BioHD encoding can also count for a possible insertion and deletion in a sequence. For the sequence searching, our encoding module needs to keep the similarity of the sequences in the high-dimensional space. Instead of encoding the entire sequence at once, BioHD split the sequence into small partitions. Then, BioHD exploits the binding to map the small partition into high-dimensional space. BioHD splits sequence into small non-overlapped partitions with the length of n. For a query sequence with length of q, BioHD splits sequence to q/n number of partitions. Each chunk encodes into high-dimensional space using the same encoding method used for exact pattern matching. For example, for i^{th} partition, the encoding performs as: $S_i = H_1 * \rho H_2 * \cdots * \rho^{n-1} H_n$. Since the partitions have small sizes, they have a high possibility that all genome digits to be the same in each partition (4^n distinct patterns for partitions with the size of n). Partitions are bundled into substrings of the length equal to the query. Thus, each substring has q/n partitions. BioHD aggregates the information of different partitions by preserving their position using a set of position hypervectors: $\vec{Q} = \vec{\mathrm{P}}_1 * \vec{\mathrm{S}}_1 + \vec{\mathrm{P}}_2 * S_2 + \cdots + \vec{\mathrm{P}}_{\frac{q}{n}} * \vec{\mathrm{S}}_{\frac{q}{n}}$, where $\vec{\mathrm{P}}$s are a set of correlated hypervectors, where $\vec{\mathrm{P}}_1$ and $\vec{\mathrm{P}}_{n/q}$ are nearly orthogonal while $\vec{\mathrm{P}}_k$ and $\vec{\mathrm{P}}_{k+1}$ have high similarity.

3.2 Reference Generation

The query sequence is typically short, while the reference protein is a long-length sequence. The reference can be encoded by mapping each protein sequence (Figure 3❹). This encoding is repeated over the reference genome using a sliding window scheme. A window moves through a protein sequence, and BioHD encodes a protein sequence in each window. Next, BioHD accumulates multiple sequence hypervectors into a single reference hypervector: $\vec{\mathrm{R}} = \sum_{j=1}^{P} \vec{\mathrm{S}}_j$. We check the existence of a query, $\vec{Q}$, in the reference hypervector by performing the following similarity measurement:

$$\delta(\vec{\mathrm{R}}, \vec{\mathrm{Q}}) = \underbrace{\delta(\vec{\mathrm{S}}_k, \vec{Q})}_{\textbf{Signal}} + \underbrace{\sum_{i=1, i\neq k}^{P} \delta(\vec{\mathrm{S}}_i, \vec{Q})}_{\textbf{Noise}} \quad (1)$$

If $\vec{\mathrm{S}}_k = \vec{Q}$ for some k, the output of the function will be 1, For reference patterns that do not match with the query, the similarity is nearly zero, $\delta(\vec{\mathrm{S}}_i, \vec{Q}) \simeq 0$. The level of noise is increasing the number of non-matched sequences. This limits the number of sequence hypervectors that can be stored in each reference hypervector. BioHD needs to ensure that the level of noise is small enough that makes the signal identification non-distinguishable. To address the issue, BioHD exploits multiple hypervectors representing the reference sequence. Each hypervector stores the information of a pre-defined number of patterns (evaluated in Section 6.2).

3.3 Protein Pattern Matching

BioHD performs the sequence searching by checking the similarity of an encoded protein sequence with the HDC library. Depending on the encoding, BioHD searches for an exact or approximate match of a query with each reference hypervector. The search checks if $\vec{Q}$ exists in $\vec{\mathrm{R}}$ using a T-membership function:

$$\begin{cases} 1 & \vec{\mathrm{S}} \cdot \vec{Q} \geq T_m \times D \\ 0 & Otherwise \end{cases} \quad (2)$$

where an output of 1 indicates membership, and 0 otherwise. We exploit different T_m for an exact or approximate match. In case of the exact matching, the theoretical recall rate (correctly identify match) can be evaluated precisely:

$$Pr(M_{S,T}(Q) = 1 | Q \in S) \geq \sum_{i=a}^{(P-1)D} \frac{1}{2^{(P-1)D}} \binom{(P-1)D}{i} \quad (3)$$

where $a = (P + T_m - 2k) \times D/2$. The last step achieves equality when the query appears exactly once in the reference. This indicates that we can improve the matching accuracy by (i) using a larger threshold value T_m, (ii) increasing dimensionality D, or (iii) retraining the model (increasing k). Note that increasing the T_m results in larger true negative cases. Also, the increase in dimensionality comes at the cost of lower computation efficiency (evaluated in Section 6.2).

Adaptive Library Refinement: Similar to the human brain, it is difficult for BioHD to remember the information of all accumulated data into each reference hypervector using single-pass memorization. The pattern of the most common sequence dominates the reference hypervector and results in vanishing the pattern of the less frequent sequences. To better memorize the information, BioHD looks at the same object multiple times to boost the accuracy by discarding the mispredicted queries from the corresponding model hypervector and adding them to the right one. If a query ($\vec{\mathrm{S}}$) corresponding to R^c reference mispredicts with R^i reference hypervector, we update the library:

$$\vec{\mathrm{R}}^c = \vec{\mathrm{R}}^c + (1 - \vec{\mathrm{R}}^c \cdot \vec{\mathrm{S}}/D) \times \vec{\mathrm{S}} \quad \& \quad \vec{\mathrm{R}}^i = \vec{\mathrm{R}}^i - (1 - \vec{\mathrm{R}}^i \cdot \mathrm{S}/D) \times \vec{\mathrm{S}}$$

where the term $1 - \vec{\mathrm{R}} \cdot \vec{\mathrm{S}}/D$ ensures that both reference hypervectors update depending on how far a query is mispredicted.

4 HARDWARE ACCELERATION

BioHD can be accelerated on existing parallel architectures. For example, BioHD provides high computation efficiency when running on GPU platforms (explained in Section 6.4). However, in existing platforms, data movement between memory and computing units (moving the reference library to a processing core) takes most of the alignment performance and energy. We propose a novel processing in-memory (PIM) architecture to accelerate BioHD computation. Besides data movement, BioHD has several features that make it well-suited for PIM: (i) the operations of the proposed algorithms can be natively supported in the memory, e.g., Hamming distance computing, (ii) BioHD is memory-centric and highly parallel at heart, and (iii) it provides strong robustness to noise. In this section, we explain how PIM can support all required BioHD operations by making minor modifications to existing crossbar memory.

Figure 4 shows the summary of the supported operations: (i) search-based Hamming computing to measure the distance between a query and reference hypervectors, (ii) row-parallel arithmetic operations to support high-precision dot-product distance computation as well as to check the existence of a query in a reference hypervectors, and (iii) a row-parallel binding operation to perform encoding operation.

4.1 Search-based Distance Computing

Exact Search: Content addressable memories (CAMs) support exact search operation [50]. As Figure 4 shows, in conventional CAM, each cell represents using two memory elements, where the elements take complementary values. For example, in a crossbar memory, a CAM cell represents using two NVM devices (0T-2R). The memory elements stores complementary values. A CAM array consists of a match-line (ML) shared among all cells in a row and a bitline that controls all cells in a column. Before search operation, all MLs are pre-charged to high voltage. During the search, the bitlines load the search query among all CAM rows in parallel. If the value of the CAM cells matches the search query, they do not discharge the ML. Otherwise, mismatch cells add discharging currents. CAMs assign a sense amplifier to the tail of each ML to detect a row with all cells matched.

Hamming Distance Computing: Several recent works modified the CAM sense amplifier to detect the search for the nearest Hamming distance search [37, 51, 52]. Sense amplifiers in different CAM rows are competing to find an ML that has the minimum discharging current. Although the nearest search is important functionality, in many real-world problems, we must measure the actual distance of the query with CAM rows. Similarly, BioHD also requires computing the distance of an encoded query with the HDC library. In CAM, the ML discharging current is related to the number of mismatched cells. The more mismatches, the higher current passes through the ML. However, this current does not linearly scale with the number of mismatches (Figure 4a). This non-linear behavior makes Hamming computing challenges.

To address this issue, we propose a technique that enables ML discharging current to better scale with the number of mismatches. We observe that the dropping ML voltage is the main reason that the ML current does not scale with the number of mismatches. Ideally, each CAM cell needs to have a discharging current equivalent to:

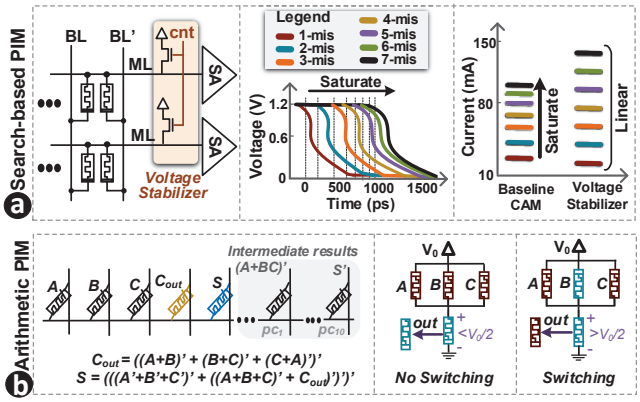

Figure 4: PIM Search-based & arithmetic computation.

$I_{miss} = V_{ML}/R_{ON}$, where V_{ML} is the ML voltage and R_{ON} is the resistance value of CAM cell during mismatch. Due to the delay of the sense amplifier in sampling the ML current and quick drop in ML voltage, the ML I_{miss} does not have a linear value over all mismatches. We exploit a *voltage stabilizer* that connects the ML to a constant supply voltage to ensure V_{ML} voltage does not drop lower than V_{dd}. Figure 4a shows the enhanced CAM array. This ensures that every mismatch will have the same impact on the ML discharging current. Figure 4a also shows that the ML current that was initially saturating scales linearly up when using the voltage stabilizer. BioHD exploits a low-precision analog-to-digital converter (ADC) for Hamming computing. When searching over a very large CAM, the ML discharging current can still be close, and thus the sense amplifier may not detect their difference. To address this challenge, we perform sequential Hamming computing over small windows sizes.

4.2 Row-Parallel PIM-based Arithmetic

BioHD supports arithmetic operations directly on digital data stored in memory without reading them out of sense amplifiers [22, 53–58]. Our design exploits the NVM switching characteristic to implement NOR gates in digital memory [22, 54]. BioHD selects three or more columns of the memory as input NOR operands by connecting them to ground voltage (shown in Figure 4b). This NOR computation performs in row-parallel on all the activated memory rows by the row-driver. Since NOR is a universal logic gate, it can be used to implement other logic operations like addition [59] and multiplication [22].

4.3 Row-parallel Binding

BioHD encoding requires to support the binding operation. The binding over multiple binary vectors is an XOR operation. In fact, the binding computes if there are even or odd numbers of 1-bits among the selected inputs. The XOR operation can be computed in a digital way by performing a series of row-parallel NOR operations. However, this requires several sequential cycles. To support fast encoding, BioHD exploits the same ADC block used for Hamming distance computing to perform binding in an analog way. During computing mode, each memory cell adds a current to an ML. The cells storing '1' bit are adding I_{on} current, while cells are storing '0' bit add a small I_{off} current to the bitline, where $I_{on} >> I_{off}$. This current is accumulated in an analog way using the ADC block.

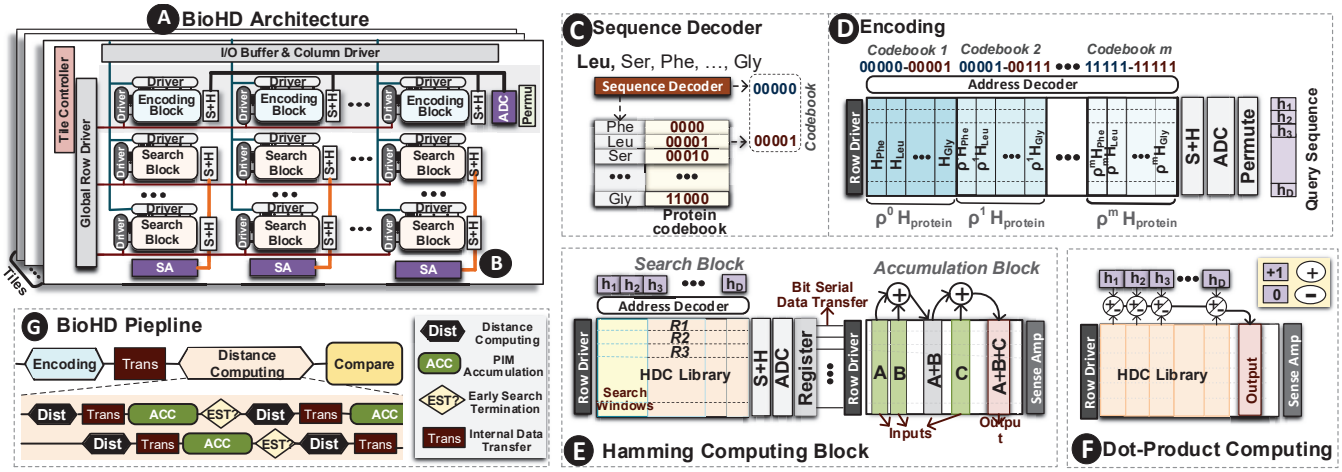

Figure 5: Overview of PIM-based BioHD architecture along with encoding and distance computing blocks.

5 BIOHD PIM ARCHITECTURE

Figure 5Ⓐ shows an overview of the PIM architecture consisting of 128 tiles. Each tile consists of 128 crossbar memory blocks. Due to the existing challenges of crossbar memory [60, 61], each memory block is assumed to have a size of $1k \times 1k$. BioHD consists of two types of memory blocks: encoding and distance computing. Both blocks are the same conventional crossbar array; they are organized in each tile to enable fast and parallel sequence searching. Both memory blocks support parallel computation. Thus, we translate all BioHD computation to vector-based operations. Our computation is performed in a row-parallel way by activating multiple memory columns and computing the result on the sense amplifier located next to each memory row. As Figure 5Ⓑ shows, the block sense amplifiers are low-precision ADCs that are shared among several memory blocks. Unlike the existing analog-based PIM, crossbar memory takes the majority of our architecture, and ADCs only take a tiny area. Each memory block also supports row-parallel bit-serial data transfer, where a single bit can be written to the next memory block over all selected rows. The memory controller is responsible for assigning the task to each memory block in a pipeline manner to provide high parallelism. Here, we describe the encoding and distance computing blocks and how they work in a pipeline.

5.1 Encoding Acceleration

PIM-based implementation of BioHD encoding requires (i) reading the pre-stored acid hypervectors from memory blocks, (ii) copying it to a new block, and performing permutation depending on the acid position in a sequence, and (iii) finally binding all permuted hypervectors. These steps are computationally costly and incur a lot of internal data movement and write operations, which are significantly slower in PIM architecture.

Codebooks: Figure 5Ⓒ shows the first step of the encoding that assigns a codebook to each amino acid. Since there are 20 acids in the Codon table (Figure 1b), these objects can be addressed using 5-bit codebooks. In hardware, the codebooks are assigned using a lookup table (CAM) with the exact search functionality. We also store m-bits value along with each codebook specifying the amino acid position. The value of m depends on the length of the protein sequence. For example, for a sequence with 256 proteins, we require 8-bits for addressing each protein. For example, codebook `00100` `00010` indicates that 5^{th} protein in the sequence is Leu which corresponds to 2^{nd} acid in the Codon table.

Encoding Memorization: Instead of naive encoding implementation, we exploit computational reuse to avoid redundant encoding computation. Let us assume a memory block with $1K \times 1k$ size. Our approach pre-stores all amino acid hypervectors in the first 32-columns of the memory block. If BioHD dimensionality is larger than 1k, we exploit multiple neighbor (vertical) blocks to store the entire hypervector. For example, for hypervectors with $D = 4k$, four vertical blocks store BioHD dimensionality. Next, we permute all protein hypervectors (ρ: single permutation) and store them on the second 32-columns of memory. Similarly, we perform more permutations on the acid hypervectors and store them on the next set of columns. For a block with 1k-columns, each block stores 32 permutations of the acid hypervectors. As Figure 5Ⓓ shows, each 32-columns stores all possible hypervectors that a protein in a specific position can take. These hypervectors can be directly addressed using our codebooks. For example, codebook `00110` `00111` addresses to 130^{th} column of memory storing $\rho^5 H_{Ser}$. This address directly indicates that the 5^{th} amino acid in the sequence is 7^{th} acid in the Codon table.

Hypervector Binding: By accessing n codebooks in parallel, we can activate n columns of the memory block at the same time. Note that in every 32-columns chunk, only a single memory column will be activated, depending on the acid codebook. These pre-stored and permuted hypervectors eliminate the costly and sequential memory read/write and rotational shift. Finally, the encoding block binds all selected memory columns using the ADC block shown in Figure 5Ⓓ. Depending on the protein length (which often does not fit in a single memory block), our PIM computes the final binding result during multiple sequential cycles. As Figure 5Ⓑ shows, each memory block is enhanced with a permutation block. After reading n elements in a sequence, BioHD shifts the `binded` result depending on the position of the windows in the sequence. To support a protein sequence with a length of q, we compute the final binding result

in q/n iterations. The first window does not get any permutation on the binding results, while i^{th} windows is permuted by $i \times n$ positions.

5.2 Distance Computing

Distance computing is the most frequent BioHD operation. The goal of BioHD is to compute the distance of a query with large-scale reference hypervectors stored in memory.

Similarity Metric & Precision: The capability of memory to support search operations depends on the distance metric. The distance metric by itself depends on the representation of the values in the HDC library. For a library with binary representation, BioHD uses Hamming distance as a similarity metric. Although BioHD supports Hamming computing in a fast and efficient way, each binary reference hypervector can only store limited information. This means that the HDC library will consist of several binary reference hypervectors. On the other hand, BioHD can represent the library hypervector with higher precision (e.g.,32-bit values). This increases the hypervector capacity to store more information in each library hypervector. However, in this representation, we require to use costly cosine distance as a similarity metric. Here, we propose a hardware solution to support both Hamming distance and dot product distance computing in a parallel way. For binary Hamming distance, we exploit CAM blocks to support ultra-fast search operation using shared and low-precision ADC blocks, while the dot product operation happens entirely in digital space without any ADC blocks.

Hamming computing: Figure 5Ⓔ shows a row-parallel Hamming computing between a query and all reference hypervectors stored in the memory. BioHD performs sequential Hamming computing on a small search window. In this work, we use 32-bits windows to ensure 5-bit ADC precision. To limit the cost of ADC blocks, we share ADCs among multiple distance computing blocks. We use *sample & hold* (S+H) circuit to record the ML discharging current of each block and use time multiplexing to share ADC among 128 memory blocks. The result of distance computing will be written in the next memory block, called *comparison memory*, in a bit-serial row-parallel way. Since the write speed is slower than the search, we use registers to save the ADC values and write them sequentially in the next memory block. The comparison memory will accumulate the partial distance values using digital-based PIM operations explained in Section 4.2. Since the search is performed over D dimensional vectors, we require $D/32$ vector-based addition to compute the final distance result. Note that the addition is performed in the pipeline with the search operation (explained in section 5.5).

Dot Product Distance Computing: Using a high-precision HDC library, we can compute the dot product similarity of a query with multiple reference patterns stored in memory. Dot product operation is an expensive operation, especially due to the complexity of the multiplication operation. However, BioHD encoding generates a query that is always a binary hypervector. This simplifies the dot product operation to accumulate multiple vectors in memory. Figure 5Ⓕ shows row-parallel dot product operation between a query and stored reference hypervectors. As discussed in Section 4.2, BioHD arithmetic operations can add or subtract digital data located in different memory columns. BioHD uses query data as a decider to add or subtract different dimensions of the reference hypervectors. Each reference dimension (memory column) corresponding to a positive query value (+1) will be added, while dimensions corresponding to zero query value will be subtracted from the final dot product result. Our PIM solution enables addition/subtraction independent of the number of memory rows. Since the search happens on long-size vectors, the vector accumulation may not fit in a single memory block. Therefore, we exploit multiple memory blocks to compute the dot product operation partially. The partial results can be aggregated using bit-serial row-parallel data transfer between neighbor blocks. Note that, unlike Hamming computing, dot product similarity does not require costly ADC blocks for distance computing. This increases BioHD area efficiency.

5.3 Thresholding

After distance computing, BioHD needs to select a reference hypervector with close distance with the query data. The closeness defines compared to a fixed pre-defined threshold (T_m). Depending on the exact or approximate match, we may use different threshold values. Like distance computing, this operation requires massive parallelism as we need the same comparison over all reference hypervectors. Using Hamming or dot product distance metric, the result of distance computing will be stored in the *comparison block*. The comparison block is exactly one of the distance computing blocks. The only difference is that this block pre-stores the copy of the T_m threshold in every row (single column) of the block. To check for a match, we subtract T_m column from all distance vectors stored in the same memory block. Any subtracted value with a positive sign bit indicates a match.

5.4 Early Search Termination

During alignment, a query sequence will most likely match with one or a few sequences in the HDC library. In fact, this is unlikely to find several references with high score matching. Conventionally, we need to compute the distance when the sequential distance computation covers the entire D dimensions of hypervectors. In order to eliminate the significant distance computing cost, we propose the idea of Early Search Termination (EST). In this approach, the comparison block identifies the reference whose distance to query has already passed an acceptable threshold ($1 - T_m$). Next, BioHD sends a signal to the row driver of distance computing and stops the search operation on the selected rows. This selective activation lowers BioHD computation cost (Section 6.4).

5.5 BioHD Pipeline

BioHD architecture works in a pipeline (shown in Figure 5Ⓖ). At first, the encoding block maps data into high-dimensional space. Instead of assigning large PIM resources to generate all D dimensions of a query hypervector, BioHD only generates d dimensions that can be used by the distance computing block ($d < D$). Next, BioHD computes the distance of the query hypervector with all pre-stored reference hypervectors (over windows of d dimensions). During search, BioHD pipeline ensures maximum utilization of the memory blocks. As soon as a search is finished, the search for the second window starts in the same memory block. All partial distances are transferred to the comparison block. The comparison block accumulates the partial distance and checks if it is pass a

Table 1: BioHD PIM Parameters.

Component	Param	Spec	Area	Power
Crossbar array	size	1Mb ($1k \times 1k$)	$3136\mu m^2$	6.14mW
Sense Amp	number	1k	$49.2\mu m^2$	0.09mW
Memory Block	number	1	$3185.2\mu m^2$	6.23mW
Hamming Computing (HAM)				
Tile Memory	number	128 blocks	$0.40mm^2$	0.78W
ADC	resolution number	5-bits, 1.75GS/s 128	$0.16mm^2$	0.14W
S+H	number	$128 \times 1k$	$857.6\mu m^2$	14.97mW
Interconnect	size	1k/row	$0.01mm^2$	31.04mW
Controller	number	1	$146.5\mu m^2$	118.9mW
HAM-tile	size	128Gb	$0.52mm^2$	1.07W
Dot Product Computing (DOT)				
Tile Memory	number	128 blocks	$0.40mm^2$	0.78mW
Interconnect	size	1k/row	$0.01mm^2$	31.04mW
Controller	number	1	$146.5\mu m^2$	118.9mW
DOT-tile	size	128Gb	$0.41mm^2$	0.93W
HAM Chip	number size	128 Tiles 2GB	**73.52mm^2**	**137.81W**
DOT Chip	number size	128 Tiles 2GB	**53.04mm^2**	**119.79W**

desired threshold (EST check). This sequential process continues until covering all D dimensions.

5.6 Other Genomics Data

BioHD is a general alignment platform that can be used to accelerate other genomic data rather than protein. For DNA and RNA data, BioHD exploits the same framework to support high-dimensional alignment. The only difference is that for DNA and RNA alignment, we do not require an intermediate step to generate amino acid hypervectors. Similar to protein, the encoding depends on the desired alignment task. For an exact match, BioHD binds the entire DNA sequence into a single hypervector. To count for mismatches, BioHD encoding is performed over smaller windows sizes.

We exploit the same hardware architecture to accelerate BioHD computation. For encoding, PIM pre-stores the hypervector corresponding to each DNA or RNA in the encoding block. Each chunk stores all possible characters that each DNA or RNA digits can take. This limits the size of each chunk to four hypervectors ($\{\vec{\mathbf{A}}, \vec{\mathbf{C}}, \vec{\mathbf{G}}, \vec{\mathbf{T}}\}$ for DNA data). The chunks are repeated by pre-storing different permutations of the hypervectors. For example of DNA data, the i^{th} chunk stores $\{\rho^i\vec{\mathbf{A}}, \rho^i\vec{\mathbf{C}}, \rho^i\vec{\mathbf{G}}, \rho^i\vec{\mathbf{T}}\}$ hypervectors. Since the size of the query and chunk hypervectors are often limited to 200 digits [62, 63], a single memory block (>800 bitlines) can store all permuted base hypervectors. It eliminates the necessity of using permutation block and data aggregation to compute the encoding results. Regardless of data type or query length, BioHD maps both query and reference library to hypervectors. Thus, it can exploit the same hardware for distance computing.

Table 2: Evaluated Genome Sequence Datasets

	Details	Data Type	Size	Q.Length
E.Coli	Escherichia coli [70]	DNA	4.6MB	200
Human	Human chromosome 14 [70]	DNA	107MB	200
COVID-19	SRR11092057, SRR11513776 [69]	DNA	4.9GB	200
NBCI-Gene	NCBI Gene data [71]	Protein	20 GB	150
1KGene	1000Genomes data [72]	Protein	15GB	180
RefSeq	Reference Sequence [73]	protein	40GB	140

6 EVALUATION

6.1 Experimental Setup

The proposed BioHD framework has been implemented with two co-designed modules: software framework and hardware accelerator. Our software framework supports HDC library generation, updates the model, and tests BioHD quality of the alignment. Thanks to BioHD vector-based operations, we fully integrated BioHD with TensorFlow [64], and designed a cycle-accurate simulator to emulate BioHD functionality during different alignment tasks. For the hardware design, we use HSPICE for circuit-level simulations to measure the energy consumption and performance of all the BioHD operations in 28nm technology. We used System Verilog and Synopsys *Design Compiler* [65] to implement and synthesize the BioHD controller. For parasitics, we used the same simulation setup considered by work in [59]. The interconnects are modeled in both circuit and architecture levels. The robustness of all circuits have been verified by considering 10% process variations on the size and threshold voltage of transistors. Our PIM works with any bipolar resistive technologies, which are the most commonly NVMS. To have the highest similarity to commercially available 3D Xpoint, we adopt the memristor device with a VTEAM model [66]. The memristor's model parameters are chosen to produce a switching delay of 1ns, a voltage pulse of 1V and 2V for RESET and SET operations to fit practical devices [54, 55].

Table 1 shows the detailed configurations of BioHD consisting of 128 tiles. Each tile has 128 crossbar blocks. BioHD has two configurations: Hamming computing that uses shared ADC blocks for distance computing, and Dot Product computing (DOT), where the distance is computed using row-parallel PIM arithmetic. In DOT-tile, the crossbar memory takes the majority of the area and power consumption, while in HAM-tile, ADCs are taking 28% and 15% of total area and power consumption. Each HAM-tile (DOT-tile) takes $0.57mm^2$ ($0.41mm^2$) area and consumes 1.07W (0.93W) power. The total HAM-chip (DOT-chip) area and average power consumption are $73.52mm^2$ and 137.81W ($53.04mm^2$ and 119.79W), respectively. All our evaluations are performed when BioHD provides the same area in both configurations. Note that our HAM chip can be configured to perform both Hamming distance and dot product similarity, while DOT Chip is an optimized version that just supports dot product similarity.

We compare BioHD efficiency with two state-of-the-art alignment tools running on GPU: (i) NVBIO [67]: GPU-accelerated C++ framework for high-throughput sequence analysis, and (ii) GPU-BLAST [68]: a GPU-accelerated nucleotide alignment tool based on NCBI-BLAST. We test BioHD efficiency on popular genomic data. Table 2 summarizes the evaluated sequence datasets. This includes DNA datasets such as E.coli (MG1655) and recent COVID-19 reference genome [69], to large-scaled protein databases.

6.2 Quality of BioHD Sequence Searching

In BioHD, each reference hypervector has a limited capacity to store information of other encoded sequences. This capacity depends on the dimensionality and the precision of the hypervector elements. Figure 6a shows the theoretical capacity of a reference hypervector with 32-bit and 1-bit precision to memorize the information of P bundled patterns (Equation 3). The results indicate that BioHD memorizes more patterns using higher dimensionality and precision. Figure 6b compares the experimental and theoretical capacity of a reference hypervector (32-bit precision) after the HDC library refinement, explained in Section 3.3. BioHD experimental capacity is saturating by increasing the number of bundled patterns due to the non-orthogonality of the hypervectors and the weakness of the refinement method. A 32-bit precision (binary) follows the theoretical capacity using $D = 40k$ ($D = 10k$) dimensionality. Further increasing dimensionality will have less impact on increasing the hypervector capacity.

Figure 6c shows the distribution of the signal and noise when we bundle a different number of patterns into a single reference hypervector. Our evaluation shows that increasing the number of patterns increases the amount of noise. By selecting a suitable threshold value, we can identify the signal from noise. The threshold value decides for an exact or approximate match. Figure 6d is a Receiver Operating Characteristic (ROC) curve that shows the impact of the threshold value, storing a different number of patterns (P). The ROC curve shows the quality of the alignment based on false positive and true positive using different threshold values. As the graphs show, storing a large number of hypervectors into a reference ($P = 10^9$, which is more than experimental capacity) results in a BioHD quality loss (true positive less than 100%). This happens as the Gaussian noise (in Figure 6c) overlaps with the main signal. Regardless of the number of patterns, a large threshold increases the false positive rate, while a smaller threshold increases the rate of a true positive. Depending on the number of vectors stored in each reference, we select a point in the top-left region of the curve that provides maximum accuracy. In the rest of the section, we limit the capacity of each reference hypervector to its experimental level to ensure 100% true positive rates.

6.3 BioHD on PIM Architecture

Unlike the traditional alignment algorithms, BioHD has a memory-centric architecture. This makes our proposed PIM architecture a promising solution for computing local data stored in storage-class memory. Figure 7b compares BioHD efficiency running on our PIM architecture. All results are normalized to GPU running BioHD-DOT. Regardless of configuration, our PIM solution provides significantly higher efficiency compared to GPU architecture by: (i) addressing the data movement issue by enabling in-place computation, (ii) enabling massive parallelism to accelerate BioHD computation, and (iii) accelerating the essential BioHD operations including distance computing and thresholding. For example, PIM-HAM (PIM-DOT) provides, on average, 103.0× (60.5×) faster and 89.3× (61.7×) higher energy efficiency compared to BioHD-DOT running on GPU.

BioHD Configurations: To show the impact of each optimization, we report PIM efficiency on different configurations: (i) *Naive*

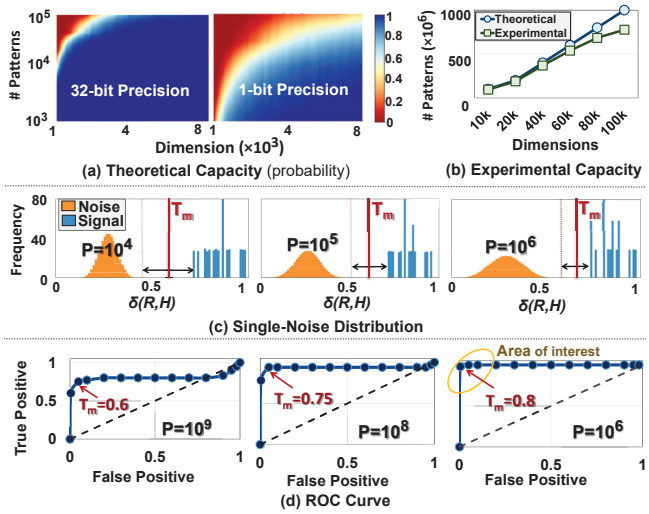

Figure 6: (a-b) Theoretical and experimental capacity of a hypervector. (c) The signal-noise distribution, (d) ROC curve indicating false positive and true positive rates.

Encoding that implements an optimized version of the encoding without exploiting the computational reuse (Section 5.1). (ii) *Naive Thresholding* that implements the thresholding by performing the tree-based comparison, rather than using the proposed thresholding (Section 5.3). (iii) *PIM-DOT* and *PIM-HAM* that use the dot product and Hamming computing circuit for distance measurement, (ii) *PIM-DOT-EST* and *PIM-HAM-EST* that enhance the distance measurement with Early Search Termination (Section 5.4).

Dot Product vs. Hamming Distance As explained in Section 4, BioHD has an option of using binary or full precision HDC library. Although the binary library can exploit low-cost Hamming distance, it requires a large number of reference hypervectors. The library is more compact but requires a more costly distance metric in high precision. Figure 7a-b compares the average BioHD efficiency using Hamming and dot product distance metrics. On GPU (Figure 7a), GPU-DOT achieves significantly higher efficiency than GPU-HAM as the performance is bounded by the number of resources. In contrast, our PIM architecture (Figure 7b) provides massive parallelism by computing over the local data. Our evaluation shows that PIM-HAM provides 1.7× faster and 1.4× higher energy efficiency compared to PIM-DOT.

Early Search Termination: Figure 7b compares BioHD efficiency using early search termination (EST). In this configuration, BioHD stops costly distance computing on the majority of the memory rows during the alignment process. The number of terminated rows depends on the threshold margin (T_m). Using $T_m = 0.9$ for DNA matching, the EST technique reduces the number of required comparisons by 63% on average. Our evaluation shows that the EST does not affect BioHD performance while improving energy efficiency by 1.3× and 1.4× in PIM-HAM and PIM-DOT, respectively. In summary, PIM-HAM is 102.8× faster and 116.1× higher energy efficient than GPU running the baseline BioHD. Although we do not explore, however, EST can further enhance BioHD performance using sparsity-aware framework. Memory blocks that are early terminated can start querying the next data in the pipeline. This

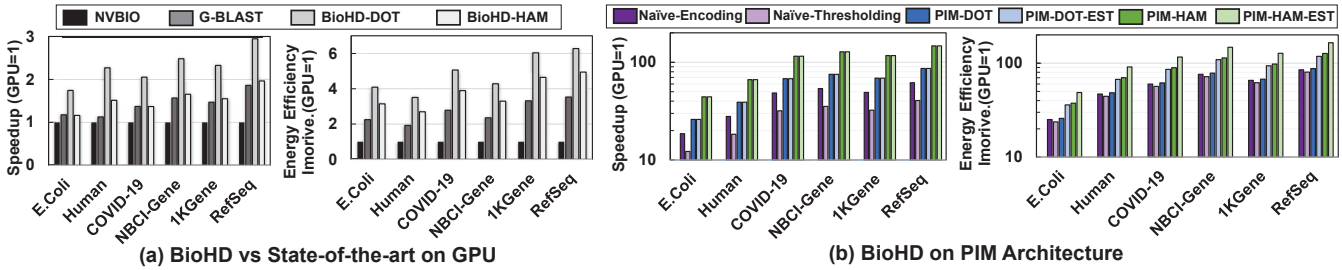

Figure 7: BioHD efficiency on GPU and PIM in different configurations. Results are normalized to GeForce RTX 3060 Ti GPU.

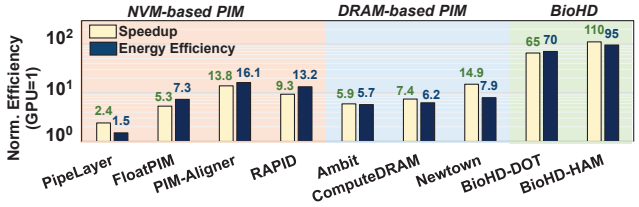

Figure 8: BioHD-PIM comparison with state-of-the-art.

approach increases the BioHD resource-utilization and provides speedup.

BioHD Encoding & Thresholding: Figure 7b shows the impact of optimized encoding and thresholding on PIM efficiency. Our evaluation shows that our encoding provides 1.4× speedup compared to the Naive-Encoding implementation. The proposed encoding has a minor impact on energy efficiency as the encoding takes a small portion of total energy. In BioHD, thresholding performs over large-scaled reference data. Our parallel thresholding results in 1.9× speedup and 1.1× energy efficiency than the naive approach.

6.4 BioHD on GPU & PIM

Figure 7a compares BioHD computation efficiency with the state-of-the-art alignment algorithms running on GeForce RTX 3060 Ti GPU. All results are normalized to the energy and execution of GPU running NVBIO running nvBowtie [74]. The G-BLAST provides higher performance and efficiency compared to NVBIO by performing a heuristic alignment since G-BLAST has (1) higher potential for GPU acceleration compared to bowtie, and (2) it enables heuristic and approximation, which can cause quality loss. All our BioHD evaluations are performed in a setting that guarantees 100% quality of the alignment (100% true positive rate). In BioHD, the library generation has a one-time cost, and generation overhead will be amortized over several alignment queries.

Our evaluation shows that BioHD provides higher efficiency compared to the baseline running on the GPU platform. This efficiency comes from the following reasons: (i) BioHD memorization reduces the number of required computations to perform an alignment task. In BioHD, each reference hypervector represents thousands of sequences. Therefore, a single similarity comparison between a query and a reference hypervector is equivalent to thousands of pattern matching in the original space. (ii) Alignment problem is not computationally expensive, while it requires bringing a large amount of data to on-chip memory and processor to perform alignment comparison. BioHD reduces this data movement by creating a compressed reference library that can be directly used for massively parallel computation. Our evaluation shows that BioHD using Hamming (HAM) and dot product (DOT) distance provide, on average, 1.6× and 2.3× faster and 3.7× and 4.9× higher energy efficiency compared to NVBIO. Note that BioHD provides higher efficiency for protein alignment rather than DNA. In protein alignment, BioHD represents each acid with a single hypervector, reducing the required operations.

6.5 BioHD-PIM vs State-of-the-art PIM

Figure 8 compares BioHD efficiency with other PIM accelerators. All PIMs have the same area as BioHD in the 1-chip configuration. The efficiency values are reported compared to GPU. We compute the efficiency of PIM accelerator using our cycle-accuracy simulator. The results are validated with the performance and efficiency reported on each original paper. PipeLayer [13] and FloatPIM [22] are neural network accelerators, but their operations can be used to accelerate the sequence searching algorithm. Our evaluation shows that BioHD provides significant efficiency improvement compared to PIM architectures. This efficiency comes from: (i) BioHD capability in revisiting alignment using HDC with hardware-friendly operations, (ii) BioHD PIM architecture supporting highly parallel essential operations, and (iii) data flow in BioHD that eliminates internal data movement. In contrast, PipeLayer and FloatPIM require a large amount of internal data movement and costly operations with lower parallelism. PIM-Aligner [75] and RAPID [39] are also recent PIM accelerators for alignment based on magnetic and resistive devices. Both accelerators have limited programmability and lack of their architectural support results in a large amount of internal data movement.

Figure 8 also compares BioHD efficiency with DRAM-based accelerators: Ambit [76], ComputeDRAM [77], and Newton [78]. DRAM-based PIMs are suitable to accelerate existing alignment algorithms that rely on extensively parallel bitwise and arithmetic computation. In contrast, these accelerators do not support associative search, which is the key functionality of BioHD computation. This makes DRAM-based solution ineffective for BioHD acceleration. In addition, DRAM-based PIM operations are often destructive, thus can result in losing the information of input operands. Our evaluation shows that, in the same area, BioHD provides 7.3× and 12.0× (14.8× and 15.3×) faster and higher energy efficiency compared to Newtown (ComputeDRAM). This efficiency mainly comes from BioHD algorithm-hardware co-design, which makes matching nearly ideal for search-based operation.

6.6 BioHD Scalability & Robustness

Technology Scaling: Figure 9 compares BioHD area and computation efficiency in different technology nodes. The results are

	DOT	HAM
Reference	32-bits	1-bit
Query	1-bit	1-bit
Tech. 45nm	1.2%	19.2%
Tech. 28nm	2.1%	28..3%
Tech. 16nm	3.9%	42.5%
Tech. 7nm	5.8%	61.7%

ADC Area Overhead

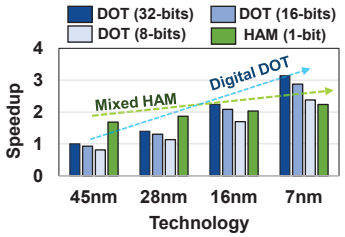

Figure 9: Technology scaling and BioHD efficiency.

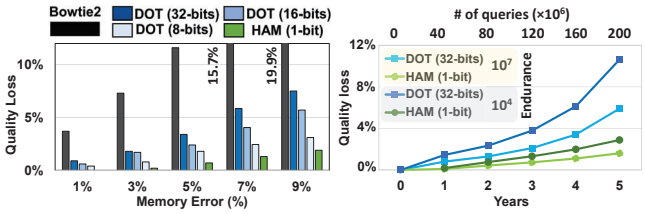

Figure 10: BioHD quality vs. noise and endurance.

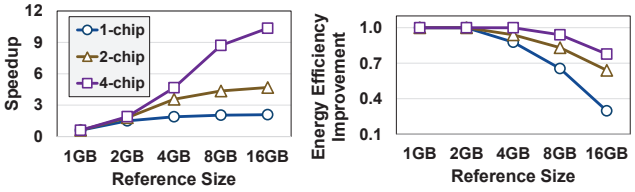

Figure 11: Scalability to the reference size and # of chips.

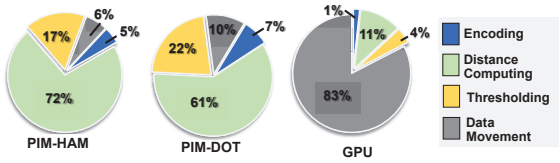

Figure 12: BioHD-PIM performance breakdown.

computed based on the detailed area-efficiency analysis of each technology. Our evaluation shows that although BioHD-HAM has higher efficiency in 45nm and 28nm technology node, BioHD-DOT is expected to provide higher efficiency in 16nm technology and beyond. This is because the ADCs take larger chip areas, as mixed-signal circuits do not scale as fast as digital technology. This results in low performance/area efficiency of BioHD-HAM in scaled technology. We also expect NVM scaling to further improve BioHD-DOT efficiency as it relies on minimal analog circuits. As NVMs are moving towards multi-bit technologies [79, 80], BioHD-DOT is expected to provide higher density and efficiency.

Robustness: Figure 10a compares BioHD robustness to noise in the memory devices. The results are reported for the Bowtie2 Alignment, and BioHD using DOT and HAM distance similarity. As we expected, regardless of the distance metric, BioHD provides significantly higher robustness to memory noise than Bowtie2 alignment. On the other hand, BioHD-HAM has higher robustness to noise compared to BioHD-DOT. In binary representation, an error only flips a reference dimension from 0 to 1 or 1 to 0. This results in minor changes in the entire hypervector pattern. In contrast, an error in BioHD-DOT can happen in the most significant bits, which can significantly impact the absolute value and robustness. Our results indicate that 9% failure in memory cells results in 1.9% and 7.5% loss on BioHD-HAM and BioHD-DOT quality of the alignment.

Figure 10b explores the impact of limited NVM endurance on BioHD quality of the alignment. We assume an endurance model with 10^7 and 10^4 [24]. Our evaluation shows that after a few years of using our PIM-based platform, similar to the human brain, BioHD starts forgetting information stored in reference hypervector. As Figure 10b shows, the aging is faster for technologies with lower endurance. To address this issue, we perform wear-leveling to distribute writes on different memory blocks uniformly. The overhead of wear-leveling is minor as BioHD has a predictable write pattern and wear-leveling only happens in long-time periods. Wear-leveling aims to move the computation such that (1) each block does not use the same memory columns as reserved columns for PIM operation. (2) it distributes the tasks among the different memory blocks. BioHD uses two counters for each memory block to implement wear-leveling. The counters keep track of the number of writes and the memory columns used for intermediate PIM computation. Our technique shifts the computation across the columns and blocks. Wear-leveling reduces the average writes in each memory cell by 18×. It creates 0.4% area overhead while having no impact on BioHD performance. Our evaluation shows that BioHD-DOT has higher sensitivity to the endurance. This is because BioHD-DOT distance computation requires PIM arithmetic operation that involves several device switching. In contrast, BioHD-HAM computes distance using a sense amplifier with minimal write operation in memory.

6.7 BioHD & Reference Genome

Figure 11 shows BioHD PIM efficiency over several synthetic reference DNA when the data size increases from 1GB to 16GB. The results are normalized to PIM performance using 1GB reference size. Our evaluation shows that PIM performance improves linearly with the reference size. For reference genome larger than PIM capacity, BioHD needs to pay an extra data movement cost to write reference genome on PIM. The performance saturation and energy efficiency degradation in Figure 11 are the result of this sequential data transfer. For example, PIM with 32GB (2GB) reference size provides 2.5× (1.8×) speedup compared to PIM with 1GB reference size. Figure 11 also shows BioHD performance scalability with the number of chips. At the algorithm level, BioHD has very low data dependency. The distance computing and thresholding, two major BioHD operations, can perform independently for reference sequences. This results in a linear performance improvement of PIM with the number of chips.

6.8 BioHD Efficiency Breakdown

Figure 12 shows the breakdown of BioHD performance on GPU and the proposed PIM platforms. In GPU, data movement takes the majority of BioHD execution time (83%). In contrast, PIM-based solutions significantly reduce the amount of data movement. In PIM-HAM and PIM-DOT, the distance computing takes the biggest portion of execution time. After that, the thresholding step is a dominant factor. PIM-DOT requires more internal data movement for aggregating of partial dot products during the distance computation. PIM-HAM has more compact data aggregation using ADC blocks.

7 RELATED WORK

Hyperdimensional Computing: The field of hyperdimensional computing is introduced by P. Kanerva, a computational neuroscientist [35]. Since then, prior research have applied the idea into diverse cognitive tasks [44, 45, 81–88]. Although the mainstream of the research is on learning tasks, HDC is the model of human memory with significant promises for memorization. For example, work in [44] showed the HDC application to enable memorization in robotics. Work in [38, 89] exploit HDC for DNA pattern matching on CMOS technology. However, this approach only supports exact matching with limited applications to DNA data. In contrast, BioHD is a general framework for genome sequence search, highly advanced with the PIM architecture.

Processing in-memory: The capability of non-volatile memories (NVMs) to act as both storage and processing units has encouraged research in Processing In-Memory (PIM) [17, 20, 22]. NVM-based PIMs have been used to accelerate a wide range of big data applications such as supervised learning [13, 15–17, 20, 21, 90], graph processing[8, 9, 91]. There are also multiple PIM accelerator for alignments, e.g., PIM-Aligner [75] and RAPID [39], Genasm [34], EXMA [92], Darwin-WGA [33], FastHASH[32], and GateKeeper[93]. However, these architectures incur large amounts of internal data movement as genomic algorithms are not memory friendly. In addition, all existing PIM-based accelerators are significantly sensitive to even small amounts of noise in hardware (e.g., 1%). However, in practice, memory devices have various reliability issues such as endurance, durability, and variability. Our solution is the first approach that introduces the idea of holographic high-dimensional representation for computation, thus enabling highly robust and efficient computation. Moreover, BioHD simplifies genome matching to hardware-friendly operations which are ideal for PIM technology.

8 CONCLUSION

We propose a novel Hyperdimensional sequence search platform for hardware-friendly computation. BioHD revisits the alignment problem with human memorization, where the reference genomes can be stored and recalled exactly or approximately during alignment. We also design a processing in-memory architecture with massive parallelism and compatible with the existing crossbar memory. Our results indicate that BioHD provides 102.8× speedup and 116.1× energy efficiency compared to GPU-based alignments.

ACKNOWLEDGEMENTS

This work was supported in part by National Science Foundation (NSF) #2127780, Semiconductor Research Corporation (SRC) Task #2988.001, Department of the Navy, Office of Naval Research, grant #N00014-21-1-2225, Air Force Office of Scientific Research, and a generous gift from Cisco.

REFERENCES

[1] Q. Li, X. Guan, P. Wu, X. Wang, L. Zhou, Y. Tong, R. Ren, K. S. Leung, E. H. Lau, J. Y. Wong, *et al.*, "Early transmission dynamics in wuhan, china, of novel coronavirus–infected pneumonia," *New England Journal of Medicine*, 2020.

[2] C. Sohrabi, Z. Alsafi, N. O'Neill, M. Khan, A. Kerwan, A. Al-Jabir, C. Iosifidis, and R. Agha, "World health organization declares global emergency: A review of the 2019 novel coronavirus (covid-19)," *International Journal of Surgery*, 2020.

[3] W.-j. Guan, Z.-y. Ni, Y. Hu, W.-h. Liang, C.-q. Ou, J.-x. He, L. Liu, H. Shan, C.-l. Lei, D. S. Hui, *et al.*, "Clinical characteristics of coronavirus disease 2019 in china," *New England journal of medicine*, vol. 382, no. 18, pp. 1708–1720, 2020.

[4] Y.-C. Liu, R.-L. Kuo, and S.-R. Shih, "Covid-19: The first documented coronavirus pandemic in history," *Biomedical journal*, vol. 43, no. 4, pp. 328–333, 2020.

[5] "Nih genome assembly for coronavirus." https://www.ncbi.nlm.nih.gov/genome/browse#!/viruses/31360/
https://www.ncbi.nlm.nih.gov/datasets/coronavirus/genomes/.

[6] Y.-Z. Zhang and E. C. Holmes, "A genomic perspective on the origin and emergence of sars-cov-2," *Cell*, 2020.

[7] A. Rambaut, E. C. Holmes, V. Hill, A. OToole, J. McCrone, C. Ruis, L. du Plessis, and O. Pybus, "A dynamic nomenclature proposal for sars-cov-2 to assist genomic epidemiology," *bioRxiv*, 2020.

[8] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3, pp. 105–117, 2016.

[9] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," in *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*, pp. 336–348, IEEE, 2015.

[10] M. Gokhale, B. Holmes, and K. Iobst, "Processing in memory: The terasys massively parallel pim array," *Computer*, vol. 28, no. 4, pp. 23–31, 1995.

[11] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, "Top-pim: throughput-oriented programmable processing in memory," in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, pp. 85–98, ACM, 2014.

[12] J. Draper, J. Chame, M. Hall, C. Steele, T. Barrett, J. LaCoss, J. Granacki, J. Shin, C. Chen, C. W. Kang, *et al.*, "The architecture of the diva processing-in-memory chip," in *Proceedings of the 16th international conference on Supercomputing*, pp. 14–25, ACM, 2002.

[13] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," HPCA, 2017.

[14] D. Fujiki, S. Mahlke, and R. Das, "In-memory data parallel processor," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1–14, ACM, 2018.

[15] S. Angizi, Z. He, and D. Fan, "Parapim: a parallel processing-in-memory accelerator for binary-weight deep neural networks," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pp. 127–132, ACM, 2019.

[16] S. Angizi, Z. He, and D. Fan, "Pima-logic: a novel processing-in-memory architecture for highly flexible and energy-efficient logic computation," in *Proceedings of the 55th Annual Design Automation Conference*, p. 162, ACM, 2018.

[17] Y. Zha, E. Nowak, and J. Li, "Liquid silicon: A nonvolatile fully programmable processing-in-memory processor with monolithically integrated reram," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 4, pp. 908–919, 2020.

[18] E. Sadredini, R. Rahimi, M. Lenjani, M. Stan, and K. Skadron, "Impala: Algorithm/architecture co-design for in-memory multi-stride pattern matching," in *2020 IEEE international symposium on high performance computer architecture (HPCA)*, pp. 86–98, IEEE, 2020.

[19] E. Sadredini, R. Rahimi, V. Verma, M. Stan, and K. Skadron, "eap: A scalable and efficient in-memory accelerator for automata processing," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 87–99, 2019.

[20] 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," in *Proceedings of the 43rd International Symposium on Computer Architecture*, pp. 27–39, IEEE Press, 2016.

[21] 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," in *Proceedings of the 43rd International Symposium on Computer Architecture*, pp. 14–26, IEEE Press, 2016.

[22] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *Proceedings of the 46th International Symposium on Computer Architecture*, pp. 802–815, ACM, 2019.

[23] C. Bo, V. Dang, E. Sadredini, and K. Skadron, "Searching for potential grna off-target sites for crispr/cas9 using automata processing across different platforms," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 737–748, IEEE, 2018.

[24] J. B. Kotra, M. Arjomand, D. Guttman, M. T. Kandemir, and C. R. Das, "Re-nuca: A practical nuca architecture for reram based last-level caches," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 576–585, IEEE, 2016.

[25] H. Saadeldeen, D. Franklin, G. Long, C. Hill, A. Browne, D. Strukov, T. Sherwood, and F. T. Chong, "Memristors for neural branch prediction: a case study in strict latency and write endurance challenges," in *Proceedings of the ACM International Conference on Computing Frontiers*, pp. 1–10, 2013.

[26] K. K. Likharev, "Hybrid cmos/nanoelectronic circuits: Opportunities and challenges," *Journal of Nanoelectronics and Optoelectronics*, vol. 3, no. 3, pp. 203–230, 2008.

[27] T. Madden, "The blast sequence analysis tool," in *The NCBI Handbook [Internet]. 2nd edition*, National Center for Biotechnology Information (US), 2013.

[28] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018.

[29] D. E. Wood, J. Lu, and B. Langmead, "Improved metagenomic analysis with kraken 2," *Genome biology*, vol. 20, no. 1, pp. 1–13, 2019.

[30] J. Adams, "Sequencing human genome: the contributions of francis collins and craig venter," *Nature Education*, vol. 1, no. 1, p. 133, 2008.

[31] J. C. Venter, M. D. Adams, G. G. Sutton, A. R. Kerlavage, H. O. Smith, and M. Hunkapiller, "Shotgun sequencing of the human genome," 1998.

[32] H. Xin, D. Lee, F. Hormozdiari, S. Yedkar, O. Mutlu, and C. Alkan, "Accelerating read mapping with fasthash," in *BMC genomics*, vol. 14, pp. 1–13, Springer, 2013.

[33] Y. Turakhia, S. D. Goenka, G. Bejerano, and W. J. Dally, "Darwin-wga: A co-processor provides increased sensitivity in whole genome alignments with high speedup," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 359–372, IEEE, 2019.

[34] 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 *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 951–966, IEEE, 2020.

[35] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, no. 2, pp. 139–159, 2009.

[36] P. Poduval, A. Zakeri, F. Imani, H. Alimohamadi, and M. Imani, "Graphd: Graph-based hyperdimensional memorization for brain-like cognitive learning," *Frontiers in Neuroscience*, p. 5, 2022.

[37] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pp. 445–456, IEEE, 2017.

[38] Y. Kim, M. Imani, N. Moshiri, and T. Rosing, "Geniehd: efficient dna pattern matching accelerator using hyperdimensional computing," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 115–120, IEEE, 2020.

[39] S. Gupta, M. Imani, B. Khaleghi, V. Kumar, and T. Rosing, "Rapid: A reram processing in-memory architecture for dna sequence alignment," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–6, IEEE, 2019.

[40] D. Baulcombe, H. Bäumlein, U. Wobus, J. Pustell, F. Kafatos, A. Bendich, P. Benfey, H. Takatsuji, L. Ren, D. Shah, *et al.*, "Altschul, sf, gish, w., miller, w., myers, ew & lipman, dj (1990)," *J. Mol. Biol*, vol. 215, pp. 403–410, 1998.

[41] M. Koyutürk, Y. Kim, U. Topkara, S. Subramaniam, W. Szpankowski, and A. Grama, "Pairwise alignment of protein interaction networks," *Journal of Computational Biology*, vol. 13, no. 2, pp. 182–199, 2006.

[42] L. Guo, J. Lau, Z. Ruan, P. Wei, and J. Cong, "Hardware acceleration of long read pairwise overlapping in genome sequencing: A race between fpga and gpu," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 127–135, IEEE, 2019.

[43] E. Rucci, C. Garcia, G. Botella, A. De Giusti, M. Naiouf, and M. Prieto-Matias, "Accelerating smith-waterman alignment of long dna sequences with opencl on fpga," in *International Conference on Bioinformatics and Biomedical Engineering*, pp. 500–511, Springer, 2017.

[44] A. Mitrokhin, P. Sutor, C. Fermüller, and Y. Aloimonos, "Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception," *Science Robotics*, vol. 4, no. 30, 2019.

[45] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pp. 64–69, 2016.

[46] Z. Zou, Y. Kim, F. Imani, H. Alimohamadi, R. Cammarota, and M. Imani, "Scalable edge-based hyperdimensional learning system with brain-like neural adaptation," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2021.

[47] B. Babadi and H. Sompolinsky, "Sparseness and expansion in sensory representations," *Neuron*, vol. 83, no. 5, pp. 1213–1226, 2014.

[48] P. Kanerva, J. Kristofersson, and A. Holst, "Random indexing of text samples for latent semantic analysis," in *Proceedings of the 22nd annual conference of the cognitive science society*, vol. 1036, Citeseer, 2000.

[49] Z. Zou, H. Alimohamadi, F. Imani, Y. Kim, and M. Imani, "Spiking hyperdimensional network: Neuromorphic models integrated with memory-inspired framework," *arXiv preprint arXiv:2110.00214*, 2021.

[50] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (cam) circuits and architectures: A tutorial and survey," *IEEE journal of solid-state circuits*, vol. 41, no. 3, pp. 712–727, 2006.

[51] K. Ni, X. Yin, A. F. Laguna, S. Joshi, S. Dünkel, M. Trentzsch, J. Müeller, S. Beyer, M. Niemier, X. S. Hu, *et al.*, "Ferroelectric ternary content-addressable memory for one-shot learning," *Nature Electronics*, vol. 2, no. 11, pp. 521–529, 2019.

[52] M. A. Abedin, Y. Tanaka, A. Ahmadi, T. Koide, and H. J. Mattausch, "Mixed digital–analog associative memory enabling fully-parallel nearest euclidean distance search," *Japanese journal of applied physics*, vol. 46, no. 4S, p. 2231, 2007.

[53] M. Imani, S. Gupta, Y. Kim, M. Zhou, and T. Rosing, "Digitalpim: Digital-based processing in-memory for big data acceleration," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, pp. 429–434, 2019.

[54] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Magic—memristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[55] R. Ben-Hur, R. Ronen, A. Haj-Ali, D. Bhattacharjee, A. Eliahu, N. Peled, and S. Kvatinsky, "Simpler magic: synthesis and mapping of in-memory logic executed in a single row to improve throughput," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[56] A. Siemon, S. Menzel, R. Waser, and E. Linn, "A complementary resistive switch-based crossbar array adder," *IEEE journal on emerging and selected topics in circuits and systems*, vol. 5, no. 1, pp. 64–74, 2015.

[57] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based material implication (IMPLY) logic: design principles and methodologies," *TVLSI*, vol. 22, no. 10, pp. 2054–2066, 2014.

[58] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "Memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.

[59] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic design within memristive memories using memristor-aided logic (magic)," *IEEE Transactions on Nanotechnology*, vol. 15, no. 4, pp. 635–650, 2016.

[60] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pp. 476–488, IEEE, 2015.

[61] A. Nag, R. Balasubramonian, V. Srikumar, R. Walker, A. Shafiee, J. P. Strachan, and N. Muralimanohar, "Newton: Gravitating towards the physical limits of crossbar acceleration," *IEEE Micro*, vol. 38, no. 5, pp. 41–49, 2018.

[62] M. Skovgaard, L. J. Jensen, S. Brunak, D. Ussery, and A. Krogh, "On the total number of genes and their length distribution in complete microbial genomes," *TRENDS in Genetics*, vol. 17, no. 8, pp. 425–428, 2001.

[63] J. Irahe Kasprzykowski, K. Ferreira Fukutani, H. Fábio, A. Maria Prado Barral, and A. Trancoso Lopo de Queiroz, "Hiv-1 nucleotide sequence comprehensive analysis: a computational approach," *Current Bioinformatics*, vol. 12, no. 4, pp. 303–311, 2017.

[64] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[65] D. Compiler, R. User, and M. Guide, "Synopsys," *Inc., see http://www. synopsys. com*, 2000.

[66] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "Vteam: A general model for voltage-controlled memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 8, pp. 786–790, 2015.

[67] "NVBIO: GPU-accelerated C++ framework for High-Throughput Sequence Analysis." https://developer.nvidia.com/nvbio.

[68] P. D. Vouzis and N. V. Sahinidis, "Gpu-blast: using graphics processors to accelerate protein sequence alignment," *Bioinformatics*, vol. 27, no. 2, pp. 182–188, 2011.

[69] "NIH SARS-COV-2 Data." https://www.ncbi.nlm.nih.gov/datasets/docs/command-line-virus/.

[70] "NIH National Library of Medicine." https://www.ncbi.nlm.nih.gov/sars-cov-2/.

[71] "NCBI Gene Database." https://www.ncbi.nlm.nih.gov/gene/.

[72] "1000 Genomes." https://www.internationalgenome.org/data/.

[73] "NCBI Reference Sequence." https://www.ncbi.nlm.nih.gov/refseq/.

[74] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with bowtie 2," *Nature methods*, vol. 9, no. 4, p. 357, 2012.

[75] S. Angizi, J. Sun, W. Zhang, and D. Fan, "Pim-aligner: a processing-in-mram platform for biological sequence alignment," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1265–1270, IEEE, 2020.

[76] 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 *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 273–287, IEEE, 2017.

[77] F. Gao, G. Tziantzioulis, and D. Wentzlaff, "Computedram: In-memory compute using off-the-shelf drams," in *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, pp. 100–113, 2019.

[78] M. He, C. Song, I. Kim, C. Jeong, S. Kim, I. Park, M. Thottethodi, and T. Vijaykumar, "Newton: A dram-maker's accelerator-in-memory (aim) architecture for machine learning," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 372–385, IEEE, 2020.

[79] S. Dutta, H. Ye, W. Chakraborty, Y.-C. Luo, M. San Jose, B. Grisafe, A. Khanna, I. Lightcap, S. Shinde, S. Yu, *et al.*, "Monolithic 3d integration of high endurance multi-bit ferroelectric fet for accelerating compute-in-memory," in *2020 IEEE International Electron Devices Meeting (IEDM)*, pp. 36–4, IEEE, 2020.

[80] C.-X. Xue, J.-M. Hung, H.-Y. Kao, Y.-H. Huang, S.-P. Huang, F.-C. Chang, P. Chen, T.-W. Liu, C.-J. Jhang, C.-I. Su, *et al.*, "A 22nm 4mb 8b-precision reram computing-in-memory macro with 11.91 to 195.7 tops/w for tiny ai edge devices," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64, pp. 245–247, IEEE, 2021.

[81] M. Imani, Y. Kim, S. Riazi, J. Messerly, P. Liu, F. Koushanfar, and T. Rosing, "A framework for collaborative learning in secure high-dimensional space," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 435–446, IEEE, 2019.

[82] L. Ge and K. K. Parhi, "Classification using hyperdimensional computing: A review," *IEEE Circuits and Systems Magazine*, vol. 20, no. 2, pp. 30–47, 2020.

[83] R. Billmeyer and K. K. Parhi, "Biological gender classification from fmri via hyperdimensional computing," in *2021 55th Asilomar Conference on Signals, Systems, and Computers*, pp. 578–582, IEEE, 2021.

[84] A. Hérnandez-Cano, R. Cammarota, and M. Imani, "Prid: Model inversion privacy attacks in hyperdimensional learning systems," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 553–558, IEEE, 2021.

[85] D. Kleyko, M. Davies, E. P. Frady, P. Kanerva, S. J. Kent, B. A. Olshausen, E. Osipov, J. M. Rabaey, D. A. Rachkovskij, A. Rahimi, *et al.*, "Vector symbolic architectures as a computing framework for nanoscale hardware," *arXiv preprint arXiv:2106.05268*, 2021.

[86] E. P. Frady, D. Kleyko, C. J. Kymn, B. A. Olshausen, and F. T. Sommer, "Computing on functions using randomized vector representations," *arXiv preprint arXiv:2109.03429*, 2021.

[87] P. Poduval, Z. Zou, H. Najafi, H. Homayoun, and M. Imani, "Stochd: Stochastic hyperdimensional system for efficient and robust learning from raw data," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 1195–1200, IEEE, 2021.

[88] Y. Halawani, D. Kilani, E. Hassan, H. Tesfai, H. Saleh, and B. Mohammad, "Rram-based cam combined with time-domain circuits for hyperdimensional computing," *Scientific reports*, vol. 11, no. 1, pp. 1–11, 2021.

[89] P. Poduval, Z. Zou, X. Yin, E. Sadredini, and M. Imani, "Cognitive correlative encoding for genome sequence matching in hyperdimensional system," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 781–786, IEEE, 2021.

[90] S. Angizi, Z. He, and D. Fan, "Dima: a depthwise cnn in-memory accelerator," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2018.

[91] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "Graphr: Accelerating graph processing using reram," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 531–543, IEEE, 2018.

[92] L. Jiang and F. Zokaee, "Exma: A genomics accelerator for exact-matching," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 399–411, IEEE, 2021.

[93] M. Alser, H. Hassan, H. Xin, O. Ergin, O. Mutlu, and C. Alkan, "Gatekeeper: a new hardware architecture for accelerating pre-alignment in dna short read mapping," *Bioinformatics*, vol. 33, no. 21, pp. 3355–3363, 2017.