# Sparsity Controllable Hyperdimensional Computing for Genome Sequence Matching Acceleration

Hanning Chen[1], Yeseong Kim[2], Elaheh Sadredini[3], Saransh Gupta[4], Hugo Latapie[5] and Mohsen Imani[1]
[1]UC Irvine, [2]DGIST, [3]UC Riverside, [4]IBM Research, and [5]Cisco Systems
{hanningc, m.imani}@uci.edu, yeseongkim@dgist.ac.kr, elaheh@cs.ucr.edu, saransh@ibm.com, hlatapie@cisco.com

2023 IFIP/IEEE 31st International Conference on Very Large Scale Integration (VLSI-SoC) | 979-8-3503-2599-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/VLSI-SoC57769.2023.10321874

***Abstract*—In this paper, we propose a Hyper-Dimensional genome analysis platform. Instead of working with original sequences, our method maps the genome sequences into high-dimensional space and performs sequence matching with simple and parallel similarity searches. At the algorithm level, we revisit the sequence searching with brain-like memorization that Hyper-Dimensional computing natively supports. Instead of working on the original data, we map all data points into high-dimensional space, enabling the main sequence searching operations to process in a hardware-friendly way. We accordingly design a density-aware FPGA implementation. Our solution searches the similarity of an encoded query and large-scale genome library through different chunks. We exploit the holographic representation of patterns to stop search operations on libraries with a lower chance of a match. This translates our computation from dense to highly sparse just after a few chuck-based searches. Our evaluation shows that our accelerator can provide 46× speedup and 188× energy efficiency improvement compared to a state-of-the-art GPU implementation. Results show that our accelerator achieves up to 3440.6 GCUPS using a single Xilinx Alveo U280 board.**

## I. Introduction

Sequence pattern matching is one of the key algorithms in identifying and analyzing genomic data. Unfortunately, the optimal solution for the 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.

A sequence matching solution requires performing multiple *sequence searching* as a fundamental procedure, i.e., checking the existence of a gene series in a database [1]. At heart, the problem is equivalent to memorization in that we should memorize and recall genomic/proteomic sequences. Traditionally, sequence searching application is based on Needleman–Wunsch Algorithm(NWA) [2] or Smith–Waterman Algorithm(SWA) [3]. The vast majority of sequence aligners based on those two algorithm relied on Dynamic Programming (DP) which naturally have non-linear execution time and memory. Currently most of the genome sequence matching acceleration on GPU or FPGA are based on SWA or NWA [4], [5], [6]. To overcome DP long execution time, we need to give up the traditional DP-matrix style searching method. Therefore in this paper, we accelerate the sequence searching in hardware by redesigning the sequence searching based on a new computing paradigm, Hyper-Dimensional Computing (HDC) [7], [8], [9], [10]. 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 sequence matching problem: (i) it transforms inherent sequential processes of the sequence searching to highly-parallelizable computation tasks, where the operations can be supported by FPGA [11], (ii) HDC is memory-centric and highly-parallel, making FPGA architecture an ideal platform for hardware acceleration [12], (iii) it provides strong robustness to noise – a key strength for enabling approximation [13].

979-8-3503-2599-7/23/$31.00 ©2023 IEEE

In this paper, we propose HyMATCH, a Hyper-Dimensional genome analysis platform. Instead of working with original sequences, HyMATCH maps the genome sequences into high-dimensional space and performs sequence matching with simple and parallel similarity searches. At the algorithm level, HyMATCH revisits the sequence searching with brain-like memorization that HDC natively supports. Instead of working on the original data, HyMATCH maps all data points into high-dimensional space, enabling the main sequence searching operations to process in a hardware-friendly way. Our FPGA accelerator parallelizes both the encode and search process and efficiently track different libraries' searching states. Combining our statistical simulation, our kernel transfers the traditional sequential searching process into of order parallel searching process. Besides large innovation inside the kernel, we also integrate the HBM inside-out accelerator to improve the data level searching parallelism [14]. HyMATCH searches the similarity of an encoded query and large-scale genome library through different chunks. We exploit the holographic representation of patterns to stop search operations on libraries with a lower chance of matching. This translates our computation from dense to highly sparse just after a few chuck-based searches.

## II. Hyperdimensional Genome Matching

Figure 1 shows an overview of HyMATCH sequence matching in high-dimensional space. HyMATCH supports exact and approximate sequence matching. HyMATCH maps each genome sequence into an orthogonal high-dimensional space, regardless of their similarity in the original space(Figure 1a).

After generating the encoded sequences, HyMATCH memorizes multiple sequences into a single reference hypervector, $\vec{\mathcal{R}} = \vec{\mathcal{V}}_1 + \vec{\mathcal{V}}_2 + \vec{\mathcal{V}}_3$, where $\vec{\mathcal{V}} \in \{-1, +1\}^D$ or $\vec{\mathcal{V}} \in \{e^{i\theta} : \theta \sim p(\omega)\}^D$, here D is the dimension of hypervector. As Figure 1b shows, each reference hypervector can store the information of thousands of sequence patterns in a compressed way. The number of patterns that can be stored in each reference depends on the orthogonality of the encoded patterns and the dimensionality of the reference hypervector. The orthogonality determines by the encoding methods, and dimensionality directly increases the computation cost. HyMATCH aggregates all encoded protein sequences to generate a reference genome, called *HDC Library*. HDC library consists of several reference hypervectors, where each hypervector memorizes thousands of genome sequences in high-dimensional space.

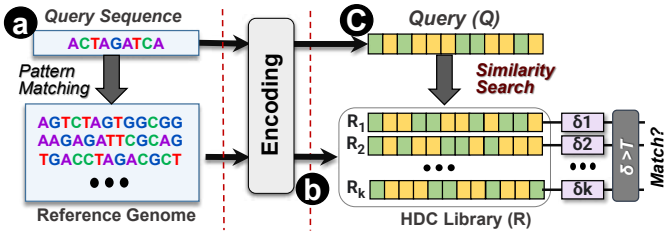


Fig. 1. HyMATCH: pattern matching in high-dimensional space.

During matching, HyMATCH uses the same encoding to map query sequence into a hypervector. A similarity computation between a query and each reference hypervector in the HDC library computes the pattern matching (Figure 1c). A reference hypervector with a similarity larger than $T$ distance thresholds determines an alignment match. We determine the distance threshold depending on dimensionality and the number of sequence patterns stored in each reference hypervectors. Overloading a reference hypervector over its capacity increases false positive or true negative matches.

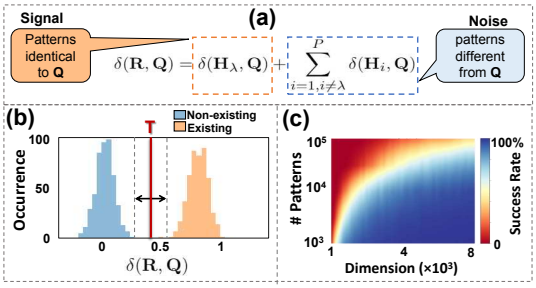


Fig. 2. (a) The theoretical capacity of a hypervector in memorization of encoded genome patterns, (b) distribution of existing and non-existing patterns in reference library during similarity search, (c) Capacity of a hypervector as function of dimensionality.

## III. Hardware Acceleration

HDC based genome sequence alignment application could be generally divided into two steps: (i) encoding raw data into hypervector genome library (ii) hypervector pairs matching. Previous work using HDC to accelerate focused more on accelerating the HDC vector encoding process but put little strength on the hypervector matching process itself [15], [16]. In this section, we are going to focus on the latter part. We will provide a new genome matching algorithm, on the one hand, different parallel libraries ($libs$) matching process, on the other hand, conduct early drop to avoid unnecessary match computing. We also notice that with the matching process going on, the sparsity of each library will increase significantly. To fully utilize FPGA computing resources, a density-aware mechanism dynamically adjusts each library.

### A. Basic Concepts

First, we clarify some basic concepts related to our accelerator in this section. Due to the large size of the hyperdimensional vector, it sometimes reaches 4k bytes. we cut a **D bytes** HDC vector into **H chunks**. Suppose each chunks' size is **M bytes**, here we have $D = H * M$. Suppose there are total **N** similarity search computing units (ALU). Due to the nature of HDC vector's information distribution over the whole vector, we compute the similarity search ($SS$) over maximum N different libraries, which means at the beginning stage of the matching process, at each memory request accelerator kernel will request total $N*M$ bytes from on-chip or off-chip storage.

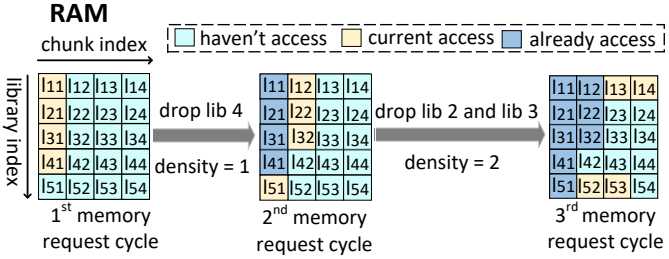


Fig. 3. Library chunks drop and density increase mechanism illustration.

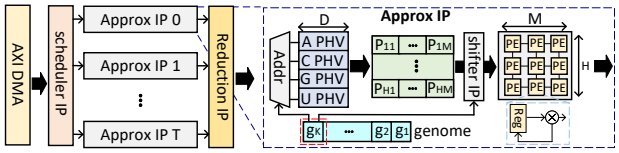


Fig. 4. Encoder architecture design.

Each library is proceeded chunk by chunk. Each genome library and query genome HDC vector similarity search result will be the accumulation of the total H chunks' similarity search result.

### B. High Bandwidth Memory Usage

High bandwidth memory (HBM) is widely used in today's FPGA accelerator design to overcome the traditional DRAM's bottleneck bandwidth. Compared to off-chip DRAM, Xilinx Ultrascale+ FPGA integrates HBM on-chip, which on the one hand, reduces the data transmission time; on the other hand, parallelizes the data reading and writing process. HDC has natural parallelism with extensive volume data needed, ideally suited to HBM's multiple data channel architectures. Specifically for this genome matching task, due to the large volume of encoding genome libraries HDC vectors, sometimes reach 100 MB to 1 GB. Using HBM to parallelize the large-scale data loading from memory to kernel significantly improves the searching speed. Also, since all libraries are independent of each other in high-dimensional space, which means that random access or cross channel access [17] will not happen if each HBM pseudo channel (PC) only supplies one accelerator kernel and if those PC and kernel combinations are logic independents. In such a case, the high throughput benefit of HBM will be maximized.

### C. HDC Early Drop

Due to the nature of the HDC model, we assume after encoding, the information of the original genome library is evenly distributed across the whole large-scale HDC vector. Unlike the previous HDC matching algorithm, which will encode and realize the similarity search across the entire hypervector, we make similarity over all libraries but only a small chunk. This will increase the parallel calculation of different genome libraries and also saves unnecessary matching calculation. Assuming library $l_i$, we cut it into several chunks represented as $V_i$, here $i$ ranges from 1 to $H$. Suppose we store this $l_i$'s chunks' matching result as $r_{l_i}$. In this way we have the whole size $D$ bytes hypervector matching result as: $r_{l_i} = \sum_{j=0}^{H} \delta(q_j, l_j^i)$. Here $\delta$ is a dot product operation used as a similarity search function, $l_j^i$ indicates $j^{th}$ chunk of library $i$, and $q_j$ indicates query the $j^{th}$ chunk of a hypervector. Unlike a normal method, conducting a matching process for the whole hypervector, here we present early stop calculation:

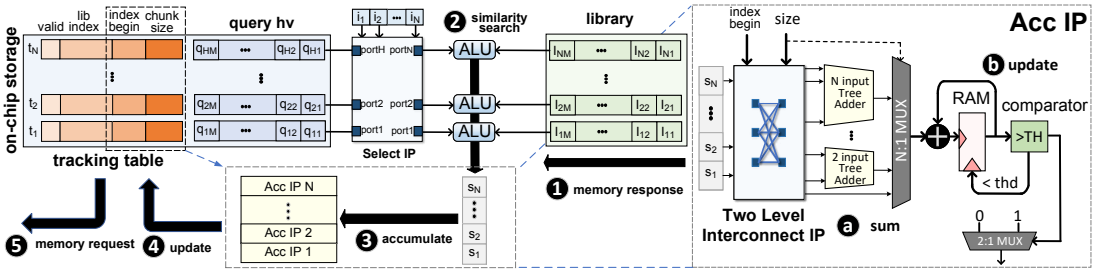


Fig. 5. FPGA accelerator top architecture design.

### D. Density Aware Consideration

After HDC vector chunks become the basic computation units, with the libraries dropping out, a sparse problem will be raised up and make the on-chip computing resources wasted. Here, we use the process and threads relationship in operating system design as an analogy to illustrate the relationship between genome library and its corresponding HDC vector chunks. In Figure 3, a simple example is provided to illustrate the genome library dropping process and the density aware mechanism. The $l_{ij}$ means the ith genome library's jth chunk. At the beginning stage, every library is stored in memory storage. The libraries will load into the searching kernel for each memory request cycle. Every square represents an HDC vector chunk, and its color represents its access situation. For example, if a square is light blue, this chunk has not been accessed. At the beginning stage of the matching process, the accelerator kernel will read one chunk for each library from memory storage. Let us suppose there are five libraries, and each library has four chunks (shown in Figure 3). The total available computing units on-chip is four. However, after the first similarity searching, the accelerator decided to drop the $4^{th}$ library. To fully use the available computing units on-chip, the $5^{th}$ library will be accessed to replace the $4^{th}$ library. This dropping mechanism only keeps necessary memory access.

Next, we introduce a library density concept ($density$). We use operating system concepts as an analogy, where there are single-threaded and multi-threaded processes. In the genome matching process, since there are many libraries that have not been loaded at the initial stage, we restrict our model as a single library single chunk or said the density of library is 1. However, if we still restrict the density of the library to be 1, many computing units will be wasted. Figure 3 shows that for the 3rd memory request cycle, only the first and fifth libraries are still active. In this situation, if we still restrict the density of those two libraries to 1, half of the on-chip computing units will be wasted. To fully utilize the on-chip computing units, increasing library density becomes critical. Let us assume the active libraries number ($N_{active}$) is less than the available computing units ($N_{comp}$). To avoid resources waste, we will increase the library density or realize density aware mechanism during matching acceleration [18]. The math equation is $density = \lfloor \frac{N_{comp}}{N_{active}} \rfloor$. For example, in Figure 3, at 3rd memory request cycle, $N_{active}$ is 2 and $N_{comp}$ is 4. So the density of libraries 1 and 5 will increase from one to two. Thus, both these libraries will process two chunks in the next cycle.

## IV. Architecture Design

### A. Encoder Microarchitecture

Figure 4 presents the hyperdimensional(HD) encoder architecture design. Both query sequence and reference sequence will first be loaded into the encoder via AXI direct memory access (DMA) as a stream. The scheduler IP will assign each genome data into its corresponding approximate(approx) IP to parallel the encoding process. Suppose the genome sequence's length is **S** and there are total **T** approx IP inside the encoder. In this case, each approx IP will conduct association encoding of **K** genome data where $K = \frac{S}{T}$. Each genome data consists of two parts. The first part is a two-bits index number representing what kind of alphabet this genome is. For example, for DNA $\Sigma = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$, the corresponding index number is: $\Sigma = \{00, 01, 10, 11\}$. This index will be treated as an address to select the corresponding base hypervector. The second part is the position index used to control the shifter IP's shifting step. To efficient process hypervector on FPGA, we reshape each hypervector from 1 $\times$ D into H $\times$ M as we discussed in section III-A. An H $\times$ M, systolic-array style, processing element (PE) array is used to conduct the K genome hypervector's association operation. After every approx IP finishes its operation, a reduction IP is used to conduct bundling operation to these T hypervectors. Finally the encoding genome hypervector will be written into the HBM via AXI Interconnect IP.

### B. Kernel Microarchitecture

The top-level accelerator architecture design is shown in Figure 5. The query hypervector is stored on-chip and divided into $H$ chunks. Each chunks' size is $M$ bytes. The referring library HDC vector chunks (library) are loaded from storage memory (DRAM or HBM) via a depth $N$ FIFO (❶). For each memory response, a totally N chunks will be loaded. The density of each library ranges from 1 to $N$. Since the similarity search of each library's chunk is out of order, borrowed the concepts from Tomasulo Algorithm [19], we exploit a table to track each library's searching "progress". The contents of the tracking table include library index, chunk index, library density, and validness of current table entry. Here the library index is used to construct the global memory address so the kernel can send memory requests via the AXI interface (further discussed in IV-C. For each of the loading library chunks, it will have its corresponding refer chunks. Here, a select IP is used to map refer chunks to each library chunks based on the N library index. $N$ parallel ALU IPs are used to conduct similarity search (❷) operation between query and library chunks. Inside each ALU, an $M$-byte multipliers is concatenated with an $M$ inputs tree adder. One thing that need to notice is that, here the $M$-byte multipliers is a tuple to tuple direct multiplier. Suppose each tuple's size is 1 bit , then the tuple to tuple multiply operation will becomes single bit XNOR operation. After each ALU's similarity search process, one byte will be generated. Totally, $N$ bytes will be generated as a similarity search result. Those similarity search results will be accumulated to its corresponding library (❸). There are N accumulation IPs (ACC IP). Inside the ACC IP, each accumulating operation has two steps. Since each library's density ranges from 1 to N, for each memory request cycle, we use $log_2 N$ different tree adder inside the Acc IP to reduce

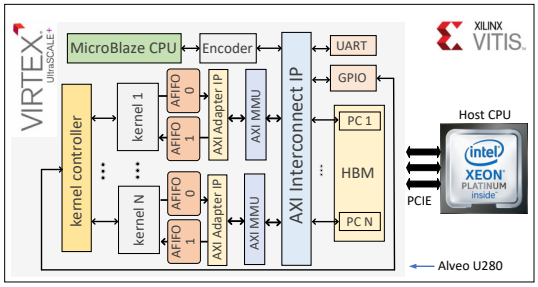


Fig. 6. System design with HBM.

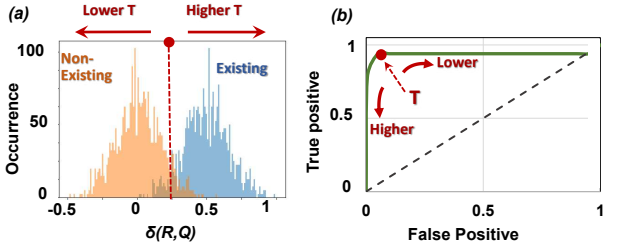


Fig. 7. (a) Distribution of existing and non-existing patterns in reference hypervector. (b) The ROC curve showing trade-off between true and false positive rates based on threshold value.

the accumulation time (❹). The Two-level interconnect IP will route searching results to their corresponding Tree Adder based on $index\ begin$ and $size$. After summing the similarity search result, a comparator is used to decide whether this library's search process will continue or just stop (❺). The library sum on-chip storage (RAM) will also be flushed with 0 if it is decided to be dropped out.

Based on accumulation and comparison results, the tracking table is updated entry by entry (❻). Suppose the library is decided to be dropped. In that case, its corresponding tracking table entry will be replaced with another library index or directly invalid if there are no more new libraries inside global memory. When it is replaced with another library index, the other parameters of this entry, such as chunk index, will be set as 0 since the search needs to start over again for this new library. In contrast, if the library is decided to be kept, there are also two situations. The first is that all chunks of this library have been searched. The kernel will keep this library's index inside the on-chip BRAM since this library has a high chance to be the successful matching one. Otherwise, the library's corresponding chunk index will increase by its current library's chunk density, and the search will continue. After updating all the entries of the tracking table, the kernel will send a new memory request (❼) to the global memory based on current tracking table information.

### C. System Design with HBM

To avoid HBM cross channel access problem, each accelerator kernel have its own corresponding HBM PCs, as shown in Figure 6. Since on Xilinx Ultrascale+ FPGA, the HBM's PCs have AXI interface, the connection between accelerator kernel and HBM is bridged by AXI interconnect IP. Since inside accelerator kernel, we only store library index, chunk index, and chunks size as shown in Figure 5, to realize AXI burst reading operation, a customized AXI adapter IP is designed. The core function inside the adapter is:$ARADDR = ADDR_{base} + index_{lib} * D + index_{chunk}$. Besides kernel accelerator and HBM, here we also give a quick introduction of other IP used in Figure 6. First, a logic controller is designed to start and stop the kernel execution. To load genome library HDC vector into HBM, a Mciroblaze softcore CPU is also used here. MicroBlaze CPU realizes HBM writing operation through AXI interconnect IP. Besides loading data, Microblaze also realizes the logic control of the accelerator via GPIO IP. After loading genome data into HBM, MicroBlaze will send a high pulse into logic controller IP via GPIO. We synthesize and generate the bitstream from Xilinx Vivado software, and other genome matching C program is tested on Xilinx Vitis unified software platform. We debug and check the C program output information on the Linux terminal via UART IP. One of the most interesting design details we want to mention is that our kernel accelerator's operating frequency differs from HBM IP's frequency. Based on Xilinx official documents, the maximum frequency of HBM can achieve 900 MHz, which is far beyond our accelerator's operation range. To overcome this cross-domain clock (CDC) problem, we adapt two asynchronous first in first out interface (AFIFO) between accelerator kernel and AXI adapter IP to avoid metastability [20].

## V. Evaluation

### A. Experimental Setup

All kernel IPs are coded either using Verilog or SystemVerilog. We verified and tested our design on the Xilinx Alveo U280 accelerator card. The C program running on the MicroBlaze CPU is written from the host CPU via Xilinx Vitis unified software platform. Two main tasks are conducted in the C program. The first is written HD genome library into HBM. Two AXI interconnect IP are used; each of them has 16 AXI input and output ports. Since one of the 32 ports is kept for Microblaze, there will be 31 AXI channels in our system. Thus, a total of 31 independent accelerators will be deployed. To avoid cross channel access, each channel only accesses its own corresponding HBM PC. The second task is to send and receive signals from the accelerator kernel via GPIO. We test HyMATCH efficiency on popular genomic data, including dataset such as Escherichia coli [21], Human chromosome 14 [21], COVID-19 DNA reference [22]. All reported results are averaged among the above-listed datasets.

### B. Quality of HyMATCH Sequence Searching

As is shown in Figure 2.(b), when tuning the threshold (T), the pattern match accuracy will also change. For the rest of the discussions, we make sure that T is low enough so the accuracy is high. Besides threshold, in the reference library, each hypervector has a limited memorization capacity. This capacity is determined by the hypervector dimensionality and precision. Low dimensional hypervectors can store a limited amount of information. Similarly, hypervectors with lower precision have less capacity to store information. To show the capacity of each reference library, Figure 7a shows the distribution of the existing (blue) and non-existing (orange) queries in each reference hypervector. We can identify the existence of a query in a reference using a single threshold method. If a similarity of a query and reference hypervector is larger than a defined threshold ($\delta(\vec{\mathcal{R}}, \vec{\mathcal{Q}}) > T$), we can consider the query as an existing pattern in the reference. The threshold value is a key in determining the correctness of the match. It also creates a trade-off between true and false-positive matches.

Figure 7b is a Receiver Operating Characteristic (ROC) curve that shows the quality of the match based on false

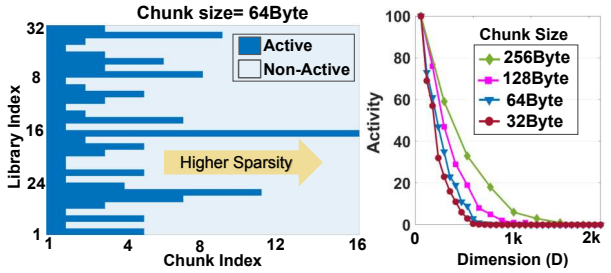


Fig. 8. (a) visualizing chunk-based search. (b) Average activity of libraries using different chunk sizes.

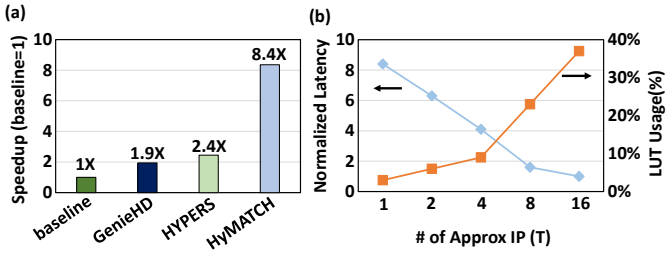


Fig. 9. (a) Encoding speedup comparison. (b) Encoding latency and LUT usage varying Approximate IP's number.

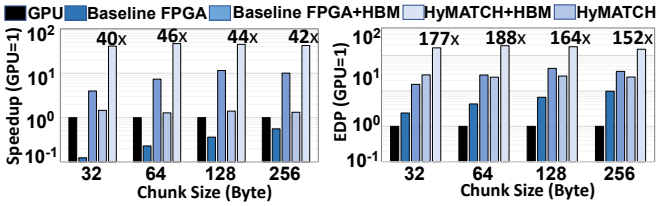


Fig. 10. Performance and energy consumption of HyMATCH on different platforms. Here GPU represents [15]'s GPU acceleration result.

positive and true positive using different threshold values. We have this trade-off since the non-existing patterns overlap with the main signal. The reason for the overlap is that the reference hypervector stored more patterns than its theoretical capacity. As the graphs show, lowering the threshold value results in a higher true positive rate. This can be achieved at the cost of a higher false-positive rate. Similarly, a higher threshold value reduces the true positive rate as the penalty of a lower true positive. In the context of our genome matching application, the low threshold is equivalent to reference libraries that wrongly match with a query, but we can ensure the correct matches are all selected. This improves our computational cost as one needs to process and verify the filtered patterns to ensure the existing. On the other hand, a higher threshold reduces the number of references that match a query, thus reducing post-processing cost. But that could result in lower quality of the match. Depending on the application requirements and available resources, one can select a suitable threshold value.

### C. Statistical Result

Figure 8a visualizes our chunk-based search using 64-dimension chunk size. Our evaluation shows that a search starts initially from dense computation in the first chunk. Going further through the chunks, the search will terminate in many libraries. This makes the distance computation highly sparse. Figure 8b shows the percentage of active libraries during a chunk-based search. The results are reported for different chunk sizes. Our results indicate that using smaller chunk size, our search has a higher capability of early termination.

### D. Accelerator Performance

This section presents the accelerator's performance based on three genome libraries' test average results. We first report the accelerator's encoding runtime and then the matching runtime. Finally, we will provide the total execution latency of genome sequence matching of three datasets. In Figure 9.a, we compare HyMATCH's encoding runtime with the two latest HDC-based genome matching accelerator works GenieHD [15] and HYPERS [16]. Here the baseline is the FPGA acceleration result of the vanilla HDC encoding without any hardware and software optimization. The number of Approx IP (T) that we choose in Figure 9.a is 8. HyMATCH's encoding performance is better than the other two HDC works since, on the one hand, HyMATCH's parallel the hypervector encoding process, and on the other hand, the association computing inside each Approx IP of HyMATCH is accelerated by a fine-tuned systolic array. However, the increase of parallelism results in more LUT usage, as shown in Figure 9.b. When deploying HyMATCH on a FPGA platform with limited resources condition, a reasonable T needs to be decided.

Five different platforms are investigated in this paper for matching speed, including the state of the art HDC based sequence matching application acceleration on GPU [15], basic parallel FPGA accelerator (baseline FPGA), baseline FPGA with HBM, HyMATCH, and HyMATCH with HBM. Here we present the latency and energy efficiency result in Figure 10. It is worthy to make it clear that although in [15], the authors present their work on AISC, FPGA, and GPU, in this paper we only choose the GPU version as the baseline. As is reported in [15], the acceleration speedup on GPU is better than on FPGA. Here we focus on the comparison of HDC based methods. In Table.I, we compare the performance of our HyMATCH with the state of the art FPGA accelerators for genome searching. More discussion of Table.I will be included in VI.

As is shown in Figure 10, the performance of GPU is better than the basic FPGA accelerator. Although for basic design, the HDC vector similarity search process is parallel but due to the low frequency of FPGA, as shown in TABLE.II, standing on the execution latency point, the FPGA's performance is not as great as GPU. After deploying HBM, data-level parallelism is significantly improved, the FPGA's performance is better than GPU for both execution latency and energy efficiency. Due to statistical analysis in Figure 8, for the HDC pattern-matching task, it is unnecessary to conduct the whole HDC vector searching process. From an FPGA acceleration perspective, loading the whole HDC vector from global memory into the kernel is a really high burden even the FPGA is integrated with HBM. In such a case, we can see after adopting an early dropping mechanism, HyMATCH achieves over 40% improvement over GPU for searching speed even though our accelerator is provided with a much lower frequency. From an energy efficiency perspective, HyMATCH achieved up to 180× improvement.

The optimized FPGA accelerator achieves excellent improvement on both speed and energy efficiency. Besides the most straightforward performance result, there are also some interesting points. (i) Since there is only very basic multiplication and addition operation involved in HDC, there is no DSP resources utilization for our HDC based FPGA accelerator which makes the kernel power consumption less than 3W based on Xilinx Power Estimator (XPE). The total Alveo U280 board power consumption is around 23.65W (ii)

TABLE I
PEAK GCUPS OF DIFFERENT FPGA ACCELERATION OF GENOME PAIRS MATCHING METHODS.

| | Year | Device | Freq(MHz) | GCUPS |
|---|---|---|---|---|
| Ours | 2022 | Xilinx Virtex Ultrascale+ XCU280 | 140 | 3440.6 |
| [6] | 2021 | 2× Xilinx Virtex U+ XCVU37P | 200 | 2073.7 |
| [6] | 2021 | 1× Xilinx Virtex U+ XCVU37P | 200 | 1251.7 |
| [4] | 2021 | Xilinx Virtex Ultrascale+ XCU280 | 225 | 270.3 |
| [23] | 2018 | Altera Stratix V | N/A | 58.4 |

TABLE II
XILINX ALVEO U280 FPGA RESOURCE UTILIZATION WHEN D = 2K BYTES, N = 32, T = 8, AND M = 64 BYETS.

| | LUTs | FF | DSP | BRAM | Power(W) | Frequency |
|---|---|---|---|---|---|---|
| **Encoder** | 277040 | 35852 | 0 | 48 | 1.046 | 140MHz |
| **Accelerator Kernel** | 393344 | 736320 | 0 | 62 | 2.107 | 140MHz |
| **Peripheral IP** | 117803 | 112171 | 0 | 20 | 21.543 | 400MHz |
| **Overall** | 788187 | 884343 | 0 | 130 | 24.696 | - |
| Utilization | 60.4% | 33.9% | 0 | 6.04% | - | - |

Although HBM improves the data parallelism, it significantly increases the power consumption. (iii) We found that we can achieve maximum accelerator performance by using chunks size equals 64 bytes. Restricted by Xilinx AXI Interconnect IP bandwidth, further increasing the chunk size will saturate the on-chip memory bandwidth.

### *E. Comparison with State-of-the-Art*

The comparison of HyMATCH with previous FPGA accelerators [4], [6], [23] is shown in Table.I. Here we use Giga Cell Per Second (GCUPS) [24] as the performance metric to compare our HyMATCH with the other works. One thing that needs to notice is that the best performance of [6] is achieved by using two Xilinx Virtex U+ FPGAs. For single FPGA acceleration performance, our work achieves 2.7× more GCUPS than [6]. When compared with [4], our design achieved 12.7× more GCUPS. We believe, compared to traditional SWA-based matching algorithm, HDC based sequence searching operation has linear time complexity and much simpler operation, therefore resulting in much better acceleration performance on FPGA.

## VI. RELATED WORK

Sequence matching acceleration on FPGA has been widely studied by recent works [4], [25], [26], [6], [27], [23], [28]. Hyperdimensional computing (HDC) is also introduced as promising solution to accelerate genome sequence matching [15], [16]. Compared to previous traditional sequence matching algorithms [4], [25], HDC-based matching process happens in high-dimensional space, where the original recursive DP-based alignment process is transferred into hypervector to hypervector similarity search. However, the existing HDC-based genome matching techniques [15], [16] are only focused on accelerating the encoding process. However, for the genome matching task, HDC encoding is a one-time task. Therefore, it is also crucial to accelerating the hyperdimensional sequence matching process. Besides, the encoding method presented in work [15] and work [16] are not hardware friendly. Work [15] tried to eliminate redundant encoding operations, but its algorithm is sequential. Work [16] tries to parallel the encoding process based on hypervector approximate matching. However, its hardware design is too simple due to the lack of considering the FPGA platform's parallelism.

## VII. CONCLUSION

In this paper, we propose a Hyper-Dimensional genome analysis platform. Instead of working with original sequences, our method maps the genome sequences into high-dimensional space and performs sequence matching with simple and parallel similarity searches. At the algorithm level, we revisit the sequence searching with brain-like memorization that HDC natively supports. We then design a density-aware FPGA implementation. Our solution searches the similarity of an encoded query and large-scale genome library through chunks.

## VIII. ACKNOWLEDGEMENT

This work was supported in part by DARPA, National Science Foundation #2127780 and #2312517, Semiconductor Research Corporation (SRC), Office of Naval Research, grants #N00014-21-1-2225 and #N00014-22-1-2067, the Air Force Office of Scientific Research under award #FA9550-22-1-0253, and generous gifts from Xilinx and Cisco.

## REFERENCES

[1] T. Madden, "The blast sequence analysis tool," *The NCBI handbook*, vol. 2, pp. 425–436, 2013.

[2] S. B. Needleman *et al.*, "A general method applicable to the search for similarities in the amino acid sequence of two proteins.," *Journal of molecular biology*, vol. 48 3, pp. 443–53, 1970.

[3] T. F. Smith *et al.*, "Identification of common molecular subsequences," *Journal of molecular biology*, vol. 147, no. 1, pp. 195–197, 1981.

[4] A. Zeni *et al.*, "The importance of being x-drop: High performance genome alignment on reconfigurable hardware," in *IEEE FCCM 2021*, pp. 133–141, IEEE, 2021.

[5] J. Cong *et al.*, "Smem++: a pipelined and time-multiplexed smem seeding accelerator for genome sequencing," in *FPL 2018*, pp. 210–2104, IEEE, 2018.

[6] A. Haghi *et al.*, "An fpga accelerator of the wavefront algorithm for genomics pairwise alignment," in *FPL 2021*, pp. 151–159, IEEE, 2021.

[7] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *CC*, 2009.

[8] M. Imani *et al.*, "Exploring hyperdimensional associative memory," in *HPCA*, pp. 445–456, IEEE, 2017.

[9] A. Rahimi *et al.*, "Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of exg signals," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 123–143, 2019.

[10] Z. Zou *et al.*, "Biohd: an efficient genome sequence search platform using hyperdimensional memorization," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pp. 656–669, 2022.

[11] H. Chen, M. H. Najafi, E. Sadredini, and M. Imani, "Full stack parallel online hyperdimensional regression on fpga," in *IEEE ICCD 2022*, pp. 517–524, IEEE, 2022.

[12] H. Chen *et al.*, "Darl: Distributed reconfigurable accelerator for hyperdimensional reinforcement learning," in *IEEE/ACM ICCAD 2022*, pp. 1–9, 2022.

[13] M. Imani *et al.*, "Neural computation for robust and holographic face detection," in *ACM/IEEE DAC 2022*, pp. 31–36, 2022.

[14] D. Abhi *et al.*, "Bridging the gap between advanced memory and heterogeneous architectures," in *IEEE FCCM 2018*, pp. 226–226, IEEE, 2018.

[15] Y. Kim *et al.*, "Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing," in *DATE*, pp. 115–120, IEEE, 2020.

[16] P. Poduval *et al.*, "Cognitive correlative encoding for genome sequence matching in hyperdimensional system," in *2021 DAC*, pp. 781–786, 2021.

[17] Y.-k. Choi *et al.*, "When hls meets fpga hbm: Benchmarking and bandwidth optimization," *arXiv preprint arXiv:2010.06075*, 2020.

[18] N. Srivastava *et al.*, "Dropout: a simple way to prevent neural networks from overfitting.," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[19] K. L. McMillan, "Verification of an implementation of tomasulo's algorithm by compositional model checking," in *International Conference on Computer Aided Verification*, pp. 110–121, Springer, 1998.

[20] C. E. Cummings, "Clock domain crossing (cdc) design & verification techniques using systemverilog," *SNUG-2008, Boston*, 2008.

[21] "Ncbi support center." https://www.ncbi.nlm.nih.gov.

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

[23] E. Rucci *et al.*, "Oswald: O pencl s mith–w aterman on a ltera's fpga for l arge protein d atabases," *The International Journal of High Performance Computing Applications*, vol. 32, no. 3, pp. 337–350, 2018.

[24] B. Chapman *et al.*, *Parallel Computing: From Multicores and GPU's to Petascale*, vol. 19. IOS Press, 2010.

[25] E. Rucci *et al.*, "Swifold: Smith-waterman implementation on fpga with opencl for long dna sequences," *BMC systems biology*, vol. 12, no. 5, pp. 43–53, 2018.

[26] L. Di Tucci *et al.*, "Architectural optimizations for high performance and energy efficient smith-waterman implementation on fpgas using opencl," in *DATE 2017*, pp. 716–721, IEEE, 2017.

[27] K. Koliogeorgi *et al.*, "Dataflow acceleration of smith-waterman with traceback for high throughput next generation sequencing," in *FPL 2019*, pp. 74–80, IEEE, 2019.

[28] M. Lo *et al.*, "Algorithm-hardware co-design for bqsr acceleration in genome analysis toolkit," in *FCCM 2020*, pp. 157–166, IEEE, 2020.