DRAM-based Acceleration of Open Modification Search in Hyperdimensional Space

Jaeyoung Kang, Weihong Xu, Wout Bittremieux, Niema Moshiri, and Tajana Rosing, *Fellow, IEEE*

***Abstract*—Mass spectrometry, commonly used for protein identification, generates a massive number of spectra that need to be matched against a large database. In reality, most of them remain unidentified or mismatched due to unexpected post-translational modifications. Open modification search (OMS) has been proposed as a strategy to improve the identification rate by considering changes in spectra, but it expands the search space exponentially. In this work, we propose HyperOMS, an algorithm-hardware co-design for boosted OMS, to cope with the enlarged database and expanded search space. HyperOMS encodes spectral data into binary vectors and performs the efficient OMS in high-dimensional space. We accelerate the HyperOMS algorithm using a DRAM-based PIM accelerator, which combines processing-using-memory and near-memory processing technologies. In order to maximize the parallelization and efficiency of the accelerator, we optimize the data allocation and devise an approximation strategy for similarity computation. Experimental results show that the HyperOMS accelerator yields up to $3.8\times$ speedup and $119\times$ higher energy efficiency compared to running HyperOMS on GPU, and up to $99\times$ speedup and $1984\times$ higher energy efficiency over the state-of-the-art OMS tool, ANN-SoLo [1], while providing comparable search quality to competing tools.**

***Index Terms*—Processing-in-memory, Spectral library search, Mass spectrometry-based proteomics, Hyperdimensional computing**

I. Introduction

PROTEOMICS plays an essential role in understanding the molecular mechanisms of proteins, which are responsible for various tasks in a life of a cell. Protein biomarkers are used to predict disease progression and severity. It can provide early diagnosis and aid the therapeutic strategy design. Mass spectrometry (MS) is one of the most popular and reliable approaches to identifying and quantifying proteins and peptides in biological samples. A typical tandem mass spectrometry (MS/MS) experiment generates millions of spectra data. Researchers determine peptide annotations of the MS/MS spectra via spectral library searching. Peptide sequences are assigned to experimental MS/MS spectra by matching them against a spectral library of known peptides (see Fig. 1).

The spectral library searching is challenging since conventional similarity metrics, such as cosine similarity, cannot be used to identify MS/MS spectra pairs [2] . Proteins undergo one or more post-translational modifications (PTMs), which change their mass and MS/MS fragmentation pattern. PTMs can be introduced during sample preparation as an artifact of MS measurement, or biologically relevant PTMs arise *in vivo*. However, spectral libraries mainly contain reference spectra for unmodified peptides, so PTMs make experimental spectra difficult to identify as they no longer exactly match the reference spectra.

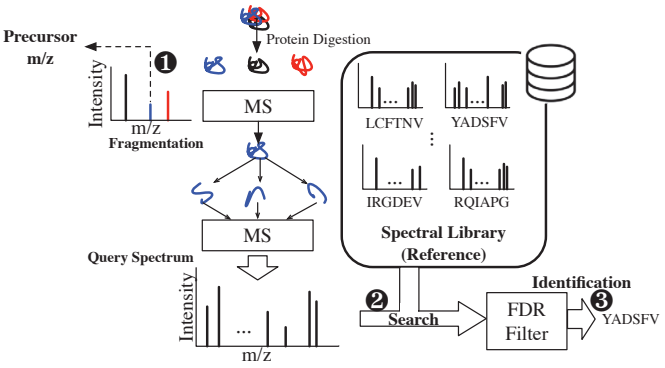


Fig. 1: Overview of spectral library search. Standard search uses a narrow precursor m/z tolerance, while OMS uses a wide precursor m/z tolerance during the searching.

Open modification searching (OMS) has been emerged to circumvent this limitation and identify modified spectra [3]. Standard spectral library searching only compares experimental spectra to reference spectra with a similar precursor mass, i.e., the mass of the unfragmented peptide, as matching peptides should have an identical mass. In contrast, OMS performs spectra matching on a wider range of reference spectra. It compares modified query spectra to their unmodified reference variants, even when their precursor mass differs due to PTMs. The higher identification capability of OMS enables the study of more complex protein interactions [4].

Compared to standard searching, the OMS suffers from low speed due to the drastically increased search space [5]. This problem is further exacerbated by the increasing spectral data due to the cost reduction in the MS experiment ($2\times$ in recent two years) [6], [7]. Also, large spectral libraries created by repository-scale mining of open MS data become available [8]. For example, the size of human HCD (higher energy collisional dissociation) spectral libraries hold 2.15 million data points, which is $4\times$ larger than the previous NIST-HCD [8]. MassIVE repository [9] contains 6.8 billion spectra, which corresponds to over 560TB in size (as of March 2024).

Several tools have been introduced to efficiently perform OMS [1], [5], [10]–[12]. These tools use various techniques to refine the search space, such as fragment ion indexing [10], nearest neighbor searching [1], [5], or tag-based filtering [11]. For example, the state-of-the-art OMS tool ANN-SoLo performs nearest neighbor searching using GPU and computes shifted cosine similarities on candidates [1]. The current solutions involve a complex execution pipeline, have limited

J. Kang, W. Xu, N. Moshiri, and T. Rosing are with the University of California San Diego (e-mail: tajana@ucsd.edu).
W. Bittremieux is with the University of Antwerp.

data parallelism, and necessitate high-precision floating-point (FP32) arithmetic for optimal search quality, such as shifted cosine similarity [5].

Our previous work [13] proposed hyperdimensional computing (HDC)-inspired massively parallel OMS algorithm that encodes spectra into high-dimensional (HD) binary vectors. It addresses the search space challenges in OMS by approximating possible MS peak changes; spectra can be identified with a single similarity computation. Therefore, [13] simplifies the execution pipeline and maximizes the computation efficiency and parallelism by replacing FP32 operations with simple Boolean arithmetic, achieving up to $17\times$ speedup and $6.4\times$ energy efficiency improvement on NVIDIA Geforce GTX 1080Ti GPU. Nevertheless, as novel spectral libraries keep growing in size, we face a "memory wall", with the runtime dominated by IO operations and limited GPU memory capacity. Our profiling results show that running [13] on GPU is bound by memory bandwidth (see Section IV-A1).

Memory-centric computing systems, like the processing-in-memory (PIM), are being developed as potential solutions to the "memory wall" problem. They offer extensive parallelism with scalable memory bandwidth, and reduce overhead caused by data movement between processor and memory. Several accelerators [14]–[16] use near-memory processing (NMP) technique, which integrates computing logic and buffer in an advanced IC package and exposes large internal memory bandwidth. Meanwhile, other PIM-based accelerators, like those described in [17], [18], process data inside memory cells using analog technology, specifically categorized as processing-using-memory (PuM). Several studies [19]–[21] demonstrated that the integration of PuM and NMP technology can effectively manage each stage of the algorithm. However, implementing existing OMS solutions that use high-precision floating-point arithmetic [1], [5], [11], [12], on current memory-centric architectures is challenging since they require costly peripheral circuits, such as analog-to-digital/digital-to-analog converters [22] and floating-point units [23]. Our previous work [13], uses lightweight Boolean arithmetic with high parallelism and a simplified execution pipeline, which can minimize the necessity of peripheral circuits. This approach is ideal for PIM-based hardware acceleration due to its memory-centric and highly parallel nature.

In this work, we propose a novel DRAM-based PIM accelerator that maximizes the efficiency of HyperOMS. Specifically, we develop an accelerator with PuM-NMP hybrid processing on DRAM that provides significantly higher internal memory bandwidth, lower data movement cost, and extensive data parallelism. To summarize, our contributions are as follows:

- We propose a novel HDC-inspired hardware-friendly OMS algorithm that encodes spectra to a binary vector. Our method reflects the spatial and value locality of peaks in the spectrum, making the encoded data resilient to peak shifts and intensity changes.
- Based on the GPU profiling result, we identify that HyperOMS algorithm is a memory-intensive workload. Thus, we design a DRAM-based HyperOMS accelerator that combines PuM and NMP technologies for extended memory bandwidth. To the best of our knowledge, this is the first work that exploits DRAM technology to accelerate an HDC-based algorithm.
- To address hardware utilization challenges that come from the filtering step in OMS, we introduce a scheme for optimizing data organization. The proposed data organization scheme effectively tackles the issue of bank under-utilization that arises during reference filtering, resulting in $3.7\times$ speedup.
- For the first time, HyperOMS tackles a large-scale pattern-matching problem with HD vectors. To alleviate the pressure from large dimensionality in our algorithm, we introduce a strategy that computes a sub-vector similarity only unless the result is larger than the given threshold to accommodate more parallelism on the NMP hardware. We show that it can reduce the execution time by $1.8\times$ with only 0.4% identification loss.
- Our evaluation result demonstrates that the HyperOMS accelerator provides up to $3.8\times$ ($99\times$) speedup and $119\times$ ($1984\times$) enhanced energy efficiency over HyperOMS running on GPU (ANN-SoLo on GPU [1]).

The rest of this paper is organized as follows. Section II describes the background of MS spectral library searching and related work. Section III specifies the algorithmic details of HyperOMS. Next, Section IV describes the HyperOMS accelerator and our optimization schemes. Our evaluation environment and results are described in Section V. Finally, Section VI concludes this paper.

II. Background and Related Work

A. Spectral Library Searching in MS-based Proteomics

MS is used to study the biological process in proteomics via the analysis of protein expression or state in cells or tissue. Proteins are ubiquitous building blocks of life, and they are composed of peptides, which are chains of amino acids, which can be described as a string of letters.

During MS data acquisition, peptides are ionized to receive a charge, and their mass-over-charge (m/z) is measured. First intact ions are measured in an MS scan using data-dependent acquisition, and the resulting MS spectrum contains the corresponding m/z values. The most intense peaks in the MS spectrum are selected. It is further analyzed in MS/MS scans, i.e., the second mass spectrometer. Ions with matching m/z are isolated and fragmented to generate MS/MS spectra. Fragmentation occurs along the peptide backbone in between its constituent amino acids. Peptides are split into their possible amino acid subsequences. We record the m/z and intensity values of all fragments, and the measured spectrum forms a *unique fingerprint of the measured peptide*. Thus, each MS/MS spectrum consists of *peaks* (m/z and intensity pairs), *spectrum charge*, *precursor* m/z (intact m/z from the preceding MS scan) (see Fig. 1-❶).

Spectral library searching determines which peptide corresponds to the measured spectra. (Fig. 1-❷). A spectral library contains reference spectra, each with known peptide labels. We first select the reference candidates with a similar precursor m/z to a query spectrum. Next, similarities between the query and all candidates are computed. Finally, the query spectrum is assigned the same peptide label as its highest-scoring reference match. Here, we apply a false discovery rate (FDR) filter on

search results [24] (Fig. 1-❸), which is called target–decoy strategy [24] in MS/MS analysis to reduce false positives. Decoy spectra that cannot exist are added to the spectral library besides the real (target) spectra. Decoy spectra selected by the search tool are filtered out. The number of target SSMs and decoy SSMs at a specific score can be used to compute the FDR. Typically, an FDR threshold of 1% is used to minimize the number of incorrect identifications. The quality of different search tools can be compared by *the number of identified spectra* at a fixed FDR threshold.

A **standard searching** strategy can identify directly matching spectra. It assumes that precursor m/z of query and matched reference spectra are similar (narrow precursor m/z tolerance). However, as spectral libraries mainly contain unmodified reference spectra, they cannot be used to identify modified ones. Modified ones have a different intact mass, as the modifications induce mass shifts. **Open modification searching** addresses these issues by (1) using a wide precursor m/z tolerance that exceeds mass shifts induced by modifications to select reference candidates [3], and (2) using alternative spectrum similarity measures that take peak shifts due to modifications into account [5]. Using a wide precursor m/z tolerance enables finding (partial) matches between unmodified reference spectra and their modified variants. However, a large number of candidates need to be evaluated for each query spectrum, which can be computationally demanding.

B. Accelerated Spectral Library Searching

OMS has recently become an increasingly popular search strategy, and there have been several studies to accelerate searches on parallel hardware platforms other than CPU. Several studies have focused on accelerating spectral library searching using GPUs for efficient spectrum–spectrum similarity computation [25]. [26] used a CPU-FPGA architecture in which multiple FPGAs are used for scalability and parallelism. However, none of these studies have tackled the OMS. ANN-SoLo [1] is a state-of-the-art OMS tool that uses GPU-powered nearest neighbor searching. ANN-SoLo vectorizes spectra and creates approximate nearest neighbor searching using FAISS [27] on GPUs. The result is transferred back to the host side. The shifted cosine similarity score between those candidates and queries to derive the most similar reference spectra. However, GPU-based ANN-SoLo [1] suffers from limited memory capacity and high data movement cost when handling large databases, i.e., repository-scale spectral library searching. Unlike previous approaches, the proposed accelerator tackles these challenges with PIM technologies and offers scalable memory bandwidth, memory capacity, and promising efficiency.

C. Memory-centric Computing and HDC

Previous works have proposed a memory-centric architecture to alleviate the "memory wall" challenge by moving data operations closer to the memory module. There are various choices to realize the computation: (1) integrating computing logic near the bank IO or near-subarray circuits [15], [28], or (2) computing within the memory array using memory commands [29]–[31].

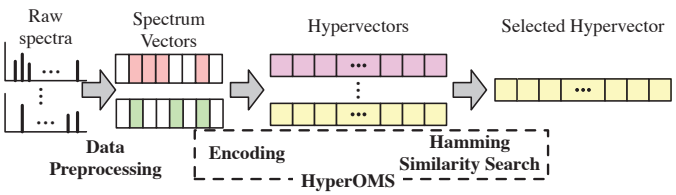


Fig. 2: Overview of OMS process using HyperOMS.

Besides, existing works have shown that running HDC on PIM hardware is much faster and more energy-efficient than on other parallel hardware platforms like GPUs. For instance, [18], [32]–[34] used ReRAM and recently, [35] used FeFET to enable HDC-based machine-learning.

However, as we tackle the large-scale search problem, these technologies are not suitable. The accelerator needs to support a large memory capacity as well as high in-memory computation parallelism for large-scale mass spectrometry data. Among various memory types for PIM, DRAM has more mature manufacturing techniques, making it practical to implement DRAM-based PIM for our use case. Additionally, DRAM provides faster writing speed and high-density memory at a low cost and is easier to scale up. Furthermore, it offers higher internal data parallelism, as it allows for the activation and access of more than thousands of bit lines simultaneously [16]. As such, we exploit DRAM to accelerate the HyperOMS algorithm in memory-centric architectures.

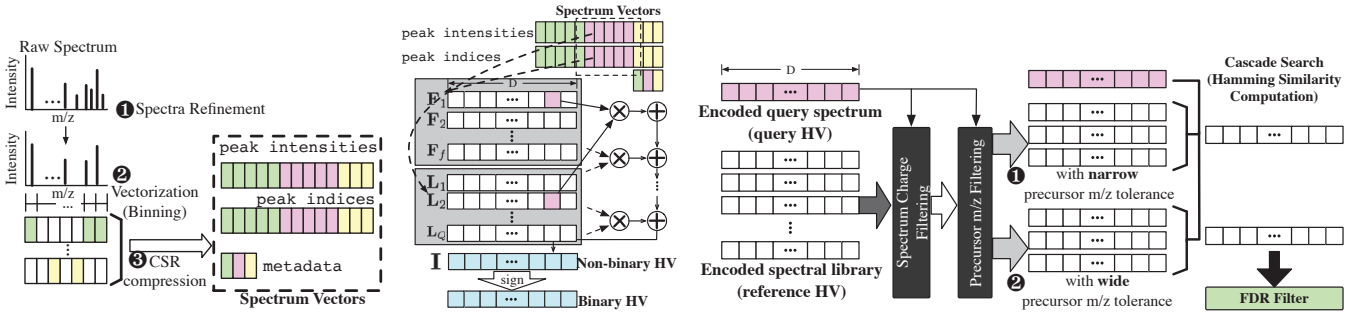
III. HyperOMS Algorithm

In this section, we introduce the HyperOMS algorithm. HyperOMS encodes spectral data into binary HD vectors called *hypervectors* (HVs) and performs OMS on them. During the encoding process, HyperOMS captures the positions and intensities of peaks while also considering their spatial and value locality. Although peaks may be shifted or have varying intensities due to PTMs, the similarity between a query spectrum and a matching reference spectrum remains stable. Additionally, since a binary vector representation is used, HyperOMS enables searching using a simple Hamming similarity computation.

Fig. 2 shows the flow of HyperOMS. It starts with a data preprocessing step, a common step for OMS. It refines and vectorizes raw spectra and compresses them, resulting in spectrum vectors. HyperOMS encodes spectrum vectors into HV during the encoding step. Next, the Hamming similarity search step finds the most similar spectra by filtering reference spectra according to the query's precursor m/z and spectrum charge and computing Hamming similarity between query HVs and candidate reference HVs.

A. Data Preprocessing

The preprocessing step (1) refines the raw spectra by removing redundant peaks and (2) vectorizes refined spectra (see Fig. 3(a)). First, raw spectra are refined to gather meaningful peaks (❶). We remove peaks whose intensity is below 1% of the most intense peaks. Low-intensity peaks are considered noise. In turn, we retain 50 to 150 most intensive peaks of the spectra. Existing studies [1], [5], [12] have shown that we can effectively refine spectra in this manner.



(a) Data preprocessing. (b) Encoding. (c) Hamming similarity search.

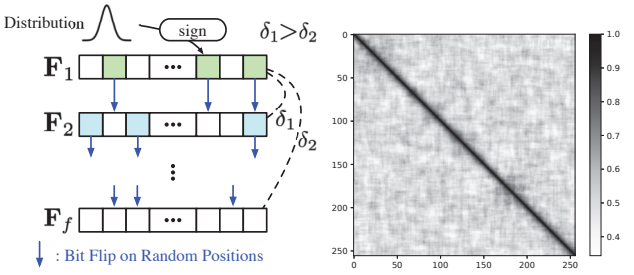
Fig. 3: Data preprocessing and stages of HyperOMS algorithm.

Next, we vectorize the filtered spectra (❷). The peaks are discretized by binning the m/z range to represent a spectrum into a sparse vector of floating-point intensity values, called a *spectrum vector*. If multiple peaks are assigned to the same m/z bin, we sum their intensity values. A large bin width can lead to a loss of information when peaks are grouped into a single bin. For example, the mass range between 0 m/z and 2000 m/z and bin width 0.04 (based on the resolution of the mass spectrometer) results in a dimensionality of $50,000$. The resulting spectrum vectors have sparsity less than 1%; there are 50 to 150 peaks for each spectrum vector and its dimensionality is $20,000$ to $50,000$. We compress spectrum vectors in a compressed sparse row (CSR) format (❸). The preprocessing step is normally run offline, and the output is stored as a binary file for future use. In the following, we discuss the HyperOMS algorithm, which first encodes spectrum vector to HV and performs Hamming similarity search on hypervectors.

B. Encoding: Spectrum Vectors to Hypervectors

HyperOMS encodes the data into a *binary vector representation*, which can enhance the computation efficiency. There have been several efforts to represent raw data in an HD binary vector, using Locality Sensitive Hashing (LSH) [18], [36] or HDC [18], [37], [38]. However, these strategies do not reflect the characteristics of OMS, including peak shifts and intensity changes. For example, they treat each feature position (corresponding to peak indices in the spectrum vector) as orthogonal. Peak shifts can lead to significant changes in similarity. Conversely, the proposed encoding takes both *spatial locality* (for peak shift) and *value locality* (for peak intensity change from instrument error or noise) of each feature into account. As a result, we can preserve similarity despite peak changes.

Fig. 3(b) shows the encoding process of HyperOMS. Unique *position HVs* $\mathbf{F}$ are assigned for each index in a spectrum vector, i.e., $\mathbf{F}_i$ corresponds to index i, and $\mathbf{F} \in \{\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_f\}$ where f is the dimensionality of spectrum vector. Similarly, we use *level HVs* $\mathbf{L}$ to capture different intensity values in each index. We quantize intensity range into Q levels, and $\mathbf{L}_i$ is assigned to each quantization level i where $i \in [0, Q]$. Given two sets of HVs, $\mathbf{F}$ and $\mathbf{L}$, a spectrum vector is encoded into an HV $\mathbf{I}$ as follows. Let $\mathbb{P}$ be the set of peaks in the spectrum vector, consisting of tuples (i, j), with i the peak index and j the step value of its intensity. $\mathbf{I}$ is computed as $\mathbf{I} = \sum_{(i,j)\in\mathbb{P}} \mathbf{F}_i \odot \mathbf{L}_j$, where $\odot$ indicates element-wise multiplication. In turn, we binarize the $\mathbf{I}$ for the computational efficiency on hardware; all positive elements are mapped to $+1$ and -1 otherwise. The final representation of the HV is a binary vector. Spectrum vectors corresponding to the query and reference spectra are encoded to query HVs and reference HVs, respectively. Encoding of reference spectra is done only once. The reference HVs are reused for subsequent runs since they are already identified and unlikely to change.



(a) (b)

Fig. 4: Position HV generation. (a) Strategy overview. (b) Pairwise similarity (Hamming similarity normalized by the HV dimension size) between position HVs.

1) Reflecting Spatial Locality: We introduce a novel position HV generation method to reflect the spatial locality. Previous studies [38] have used a permutation-based or random generation method, which makes $\mathbf{F}_i$ and $\mathbf{F}_j$ $(i \neq j)$ nearly orthogonal. However, they are vulnerable to peak shifts that accompany changes in i; the change of position HV results in significant similarity value changes of matched pairs.

Fig. 4(a) shows the proposed position HV generation strategy. We randomly generate $\mathbf{F}_1 = \{+1, -1\}^D$. In turn, we flip α components in random positions. As more flips occur, the similarity between the original vector and the flipped vector decreases. For example, the similarity (δ_1) between $\mathbf{F}_1$ and $\mathbf{F}_2$ is larger than the similarity (δ_2) between $\mathbf{F}_1$ and $\mathbf{F}_f$. The proposed encoding method reflects the characteristics of peak shifts in OMS well: (1) neighboring positions should have spatial locality to deal with peak shifts, while (2) distant positions need to have adequate orthogonality since a dramatic peak shift rarely occurs in nature. The peak shift changes the index value corresponding to the intensity in the spectrum vector. With the proposed method, position HVs do not change significantly even if peak shifts occur; thus, the resulting representation can be tolerable to them. As depicted in Fig. 4(b), for $\mathbf{F}_i$ and $\mathbf{F}_j$, the pairwise similarity has a high value when

Algorithm 1 Hamming Similarity Search Stage in HyperOMS

```
procedure HAMMING SIMILARITY SEARCH
    for c in list of all charge values C do
        Load reference HVs R with spectrum charge c
        Load query HVs Q with spectrum charge c
        x ← None
        Compute Hamming distance matrix H
        for Hamming distances h_(q,·) of query q do
            x ← arg max_i h_(q,i) with narrow precursor m/z tolerance
            if x is None then
                x ← arg max_i h_(q,i) with wide precursor m/z tolerance
            end if
        end for
    end for
end procedure
```

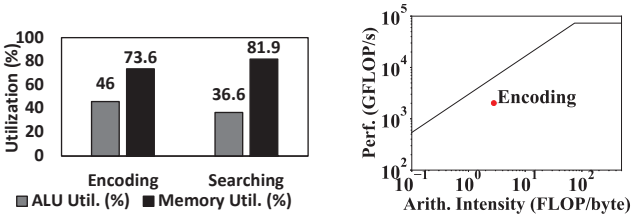
$i \approx j$ and is maximum when $i = j$ (diagonal elements). Note that we scaled down the f to 128 and D to 256 for better visibility.

2) Reflecting Value Locality: The intensity information of the spectrum vectors is captured. We use the level HVs generation method used in [37], [38]. We allocate a single bit to each of the HV components, i.e., $\mathbf{L}_i \in \{-1, 1\}^D$. $\mathbf{L}$ that is assigned to each quantization level needs to reflect the closeness of the intensity. The similarity between $\mathbf{L}_i$ and $\mathbf{L}_{i+1}$ should be higher than the similarity between $\mathbf{L}_i$ and $\mathbf{L}_{i+100}$. For instance, for the target level p in percentage, $\mathbf{L}_p$, we can represent this by flipping $(D/2) \times (p/100)$ elements of $\mathbf{L}_0$.

C. Hamming Similarity Search

As shown in Algorithm 1, HyperOMS finds the matched reference HV that is most similar to the query HV. It uses *Hamming similarity* (defined by the number of equal components in vector pairs) as a similarity metric. Here, reference spectra that need to be compared primarily need to satisfy spectrum charge and precursor m/z condition per query as discussed in Section II. We gather reference spectra that (1) have the same spectrum charge as the query spectra (Algorithm 1-L2) and (2) satisfy the precursor m/z tolerance (precursor m/z difference between query and reference) condition (Algorithm 1-L8, L10).

OMS assumes that precursor m/z of selected reference spectra and query spectra can have a large difference. A wide precursor m/z tolerance is used to match modified spectra to their unmodified variants. However, we may miss the case of a reference spectrum with a similar precursor m/z that can pass through the FDR filter with high similarity. To avoid such misidentifications, we adopt *cascade search* [39]. A narrow precursor m/z tolerance is used for standard search and FDR filtration is applied (Fig. 3(c)-❶). In turn, remaining unidentified spectra are processed with a wide precursor m/z tolerance (Fig. 3(c)-❷). Unlike existing tools, HyperOMS implements two steps of the cascade search in parallel as they share same similarity value (Algorithm 1-L8, L10). The spectrum identifications are merged at the end since both steps are computationally independent.



(a) Utilization (b) Roofline

Fig. 5: Profiling on NVIDIA Geforce RTX 4090

IV. HyperOMS Accelerator Architecture

A. Motivation

1) Insights from HyperOMS on GPU: HyperOMS algorithm is a good match for highly parallel computing platforms. We implemented HyperOMS on GPU using GPU-based HDC framework [40] and optimization strategies in [41]. Note that to maximize the computation efficiency, we represent binary HV as a 32bit integer array using bit packing, and similarity score computation is done by CUDA intrinsic. Our profiling results of [13] using `NSight Compute` on NVIDIA RTX 4090 shown in Fig. 5(a) suggest that every stage of HyperOMS has low ALU utilization (41% on average) compared to memory utilization (78% on average). The roofline from `Nsight Compute` in Fig. 5(b) shows that the encoding GPU kernel is memory bound. Note that the profiler does not support roofline analysis for the similarity computation kernel because it does not involve floating-point computations. Furthermore, two stages of the HyperOMS algorithm theoretically have low computational density: the encoding step requiring 8ops/byte and the Hamming similarity calculation requiring 16ops/byte. This implies that these steps are limited by the memory bandwidth. Each kernel implementation leverages the CUDA memory hierarchy to maximize data reuse and minimize access to the global memory for optimized performance.

2) Why we need DRAM-based PIM accelerator?: Based on the analysis of the proposed algorithm and its GPU implementation, we leverage a PIM technology to accelerate the HyperOMS algorithm. Among various memory technologies for PIM, DRAM benefits from its mature manufacturing techniques, offering high-density memory cost-effectively and excellent scalability. It offers not only scalable memory bandwidth but also provides a large memory capacity to accommodate the massive spectral data.

B. HyperOMS Accelerator Design

Existing works [19]–[21] have shown that the combination of different PIM techniques, such as PuM and NMP, can optimize the efficiency of the PIM accelerator. Inspired by this, the proposed HyperOMS accelerator is a hybrid PuM-NMP design which leverages the advantages of both PuM and NMP. Specifically, the HyperOMS accelerator uses the PuM technique for encoding and NMP for Hamming similarity search, respectively. The hybrid design is based on the following analysis. First, the encoding stage only takes a smaller portion (less than 20% in the case of larger scale HEK293) of the total

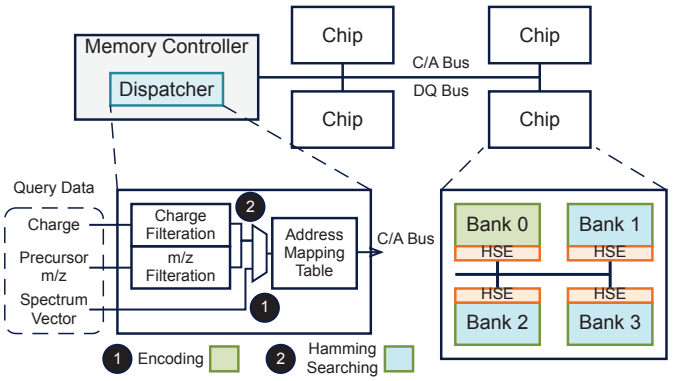


Fig. 6: Overview of the HyperOMS accelerator architecture. Encoding and Hamming similarity search step is performed in PuM and NMP, respectively.

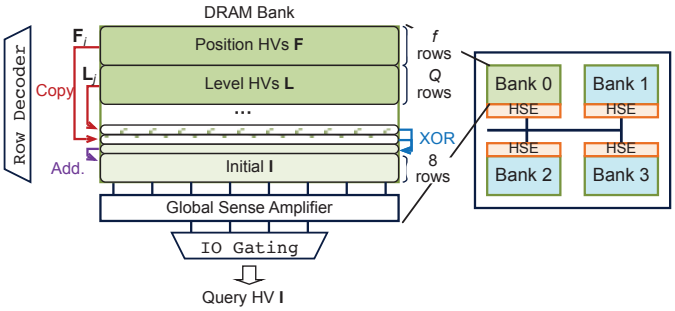


Fig. 7: In-memory encoding scheme for spectrum vectors.

runtime (see Fig. 14). Therefore, it is acceptable to adopt the PuM technique to avoid additional peripheral circuits at the cost of a longer processing time compared to NMP. In comparison, the search phase is much more time-sensitive as it takes the majority of the time, especially in larger scale datasets (see Fig. 14). As such, we accelerate the Hamming similarity search using the NMP technique because the NMP-based design is over $6\times$ faster than the PuM-based design according to the analysis in Section IV-D.

Fig. 6 illustrates the system diagram of the proposed HyperOMS accelerator that is composed of three main parts: (1) HyperOMS dispatcher, (2) encoding banks (Section IV-C), and (3) Hamming search banks (Section IV-D2). We design the accelerator to be compatible with the DDR4 standard [42]. Each DRAM chip has one encoding bank, and the rest of the banks are working under the Hamming search mode. The encoding bank handles the encoding stage of the HyperOMS algorithm using PuM primitives. Each DRAM bank includes a near-bank Hamming similarity search engine (HSE). HSE can compute similarity values for multiple pairs simultaneously by adopting the HV folding strategy (refer to Section IV-D1). The dispatcher is implemented in the memory controller (MC), working as the high-level scheduler and controller for query data fetching from the host, control/address (C/A) command generation, and query allocation to the DRAM banks. The query data (spectrum charge, precursor m/z, and spectrum vector) are fed into the HyperOMS accelerator via the dispatcher.

Dataflow: HyperOMS follows a map-reduce manner to fully utilize the internal data parallelism of DRAM. The encoding and Hamming similarity search phases are performed locally within **the encoding banks** and **Hamming search banks**, respectively. First, during the encoding phase (Fig. 6-❶), the dispatcher converts the peak intensities and indices into corresponding memory addresses to perform in-memory encoding in the encoding banks. In turn, The encoded query HVs in the encoding banks are collected by the dispatcher and then broadcast to Hamming search banks. The dispatcher first performs the spectrum charge and precursor m/z filtering (Fig. 6-❷). It generates the destination memory address and distributes the encoded query from the encoding bank to Hamming similarity search banks based on the spectrum charge and precursor m/z information. The address mapping table stores two types of mapping data: (1) the memory address of position and level HVs in the encoding bank and (2) the mapping between the reference's spectrum charges/precursor m/z and their memory addresses. Each Hamming search bank searches and finds the most similar reference HV among the stored reference data. The local search within each bank is performed asynchronously. After all Hamming search banks finish their searching process, the dispatcher gathers the results from each bank and further selects the best-matched reference HV in the reduction step. Here, spectrum charge and precursor m/z filtering can cause a bank under-utilization, which leads to speed and efficiency degradation. In order to balance the assigned workloads in banks and raise the bank utilization, we optimize the mapping of the reference data and the query dispatching, as described in Section IV-E.

Memory Modifications: To enable fast Hamming similarity search and extended bandwidth, HyperOMS accelerator involves three major memory modifications. Firstly, we add isolation transistors between subarrays to support fast inter-subarray row copy, which results in an additional area overhead of only about 0.8% [43]. Secondly, we implement the HSE design, which is connected to the local buffer of the bottom subarray in each bank. Finally, we add a new memory controller command (similar to [44]) to support data broadcast from the encoding banks to Hamming search banks. These modifications effectively balance the additional overhead and computation efficiency.

C. In-memory Encoding

The encoding process of query data is computed in the assigned encoding banks (see the green-colored bank in Fig. 6) in conjunction with the PuM primitives [29]–[31]. We exploit the PuM technique since the encoding stage only needs simple XOR and addition operations that are well supported by existing PuM primitives [29]–[31]. Also, it needs to store intermediate results. By utilizing the PuM technique, the accelerator eliminates the need for a buffer to hold these results, which is beneficial for minimizing the density impact on DRAM. The encoding is a memory-intensive process characterized by massive data movement that arose from fetching the D-bit position or level HVs over $|\mathbb{P}|$ times each. The in-situ property of PuM avoids data fetching, thereby saving memory bandwidth as well as energy. Although PuM increases the processing time compared to NMP, the encoding process only occupies a small fraction of the overall runtime (see Fig. 14).

The encoding bank execution includes three steps: (1) receiving the query data, (2) in-memory encoding, and (3)

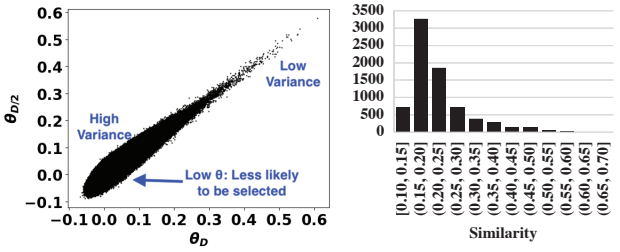
broadcasting encoded data to the query HV register within the HSE of other banks. As shown in Fig. 7, the encoding bank pre-stores the position and level HVs in a total of $f + Q$ rows. In the first step, the DRAM bank receives the row addresses of $(i, j) \in \mathbb{P}$ from HyperOMS dispatcher. Then the corresponding position and level HVs, $\mathbf{F}_i$ and $\mathbf{L}_j$, are copied to new rows. The copy process can be achieved by the fast row copy in LISA [43]. Second, we compute the XOR for each pair of copied position and level HVs using the majority-based arithmetic in [29], [31]. The third step is to aggregate the XOR results. The basic memory commands to compute in-memory bit-wise XOR and bit-serial addition are two back-to-back activation commands followed by a precharge command (AAP). Specifically, n-bit bit-serial addition needs $4n + 1$ AAP commands (XOR is regarded as 1-bit addition). We initialize the result rows with -$|\mathbb{P}|$/2 in bit-serial and 2's complement format. The mentioned three steps are repeated $|\mathbb{P}|$ times to obtain the final aggregation results. The sign bit of the result rows is selected for the binarization. After the encoding, the encoded query HV needs to be broadcast to the query HV register within the HSE of other Hamming similarity search banks. The encoded HV in a row is burst out from the encoding bank to the HyperOMS dispatcher via the inter-bank dataline. It helps to broadcast results at minimal cost without conflicting with DDR4 standard [42]. Finally, the dispatcher transmits the collected query HV to other Hamming search banks in a sequential manner.

D. Near-memory Hamming Similarity Search

The Hamming similarity search involves calculating the similarity between the query and reference HVs, which can be implemented either in-memory or near-memory. If we implement the Hamming similarity computation using the PuM technique, it incurs a long processing latency since 5 AAP commands need to be issued by the DRAM bank to compute the bit-wise XOR between two HVs. The DDR4-2400 4Gb $\times 8$ needs around 390ns to perform the PuM-based XOR operation. On the other hand, the NMP-based design can provide shorter latency for the XOR computation and has higher internal memory bandwidth for pattern-matching workloads [15]. The long latency of PuM-based computation is not favorable for Hamming similarity search as the search step in HyperOMS accounts for the majority of the overall runtime. Hence, we choose to accelerate it using a near-memory HSE module as shown in Fig. 9.

To exploit the internal bandwidth of DRAM, we locate one HSE next to the local row buffer of the last and bottom subarray in each DRAM bank. Implementing only one HSE for each bank avoids adding excessive area overhead to the original DRAM. We can use one precharge and one activates command (< 60ns) to latch the data into the local sense amplifier (SA) and row buffer (RB). The HSE directly fetches the D-bit latched data from local RB at one time (D is the size of RB), and the processing latency can be hidden by the memory access time.

Meanwhile, the HSE computes the Hamming similarity of only one pair if it is implemented naively; the parallelism is proportional to the number of HSE. Also, the XOR computation



(a) (b)

Fig. 8: Statistics of Hamming similarity values. (a) Relevance between θ_D and $\theta_{D/2}$. (b) Similarity between query and matched reference spectrum (normalized to D).

of HV pairs can be computationally demanding due to its dimensionality. As such, we devise a strategy called *HV folding* to increase the parallelism within HSE and the design of HSE.

1) HV Folding: The Hamming similarity is evaluated by XOR results between the reference HV and query HV. The naive way to get the Hamming similarity is to count the 0s of the XOR HV. However, the high dimensionality of the XOR HV with a few thousand bits makes it slow to count the accurate number of 1s. We propose the **HV folding** scheme to reduce the computation of the Hamming similarity and accommodate more parallelism. The parameters for the HV folding are static, and they are used for optimizing HSE design.

The basic idea of HV folding is to calculate only partial bits of the XOR HV (called subsequence) and approximate the actual Hamming similarity. Specifically, instead of calculating the Hamming similarity (θ_D) for the entire HV with dimensionality D, we only compute the Hamming similarity (θ_K) of the first K bits. Note that D is a multiple of K to make it evenly divisible and avoid padding. Let us assume that $K = D/2$. Fig. 8(a) shows the relationship between θ_D and $\theta_{D/2}$, which are values obtained from randomly sampled query and reference pairs. Here, θ_D and $\theta_{D/2}$ are roughly proportional. The trend is similar in the $K = D/4$ or $D/8$ case. When θ_D is small, it shows high variance, while showing low variance when θ_D is large. Fig. 8(b) shows a histogram of normalized Hamming similarity values of query and the matched reference pair in the spectrum charge +2 of the iPRG2012 dataset used in Section V. It has been noted that for the other datasets as well, the matched reference and query pairs exhibit a normalized Hamming similarity of at least 0.1. Hence, we can approximate θ_D as $\theta_{D/2}$ since we pick a spectrum with a high similarity value.

The goal of HV folding is to predict θ_D using θ_K and to handle more pairwise similarity computations in parallel within HSE. The HSE only computes θ_K if it is less than the predefined threshold T. Otherwise, HSE computes θ_D. When θ_K successfully approximates θ_D, the same reference HV needs to be selected in either case. We define this success rate as the folding success rate (FSR). It is essential to choose appropriate values for K and T and maximize the FSR. We discuss the hyperparameter search experiment in Section V-F.

2) Hamming Similarity Search Engine: The Hamming similarities between query HVs and reference HVs are computed in near-bank HSE (see blue-colored banks in Fig. 6). In the memory subarray, each D-bit reference HV is arranged into

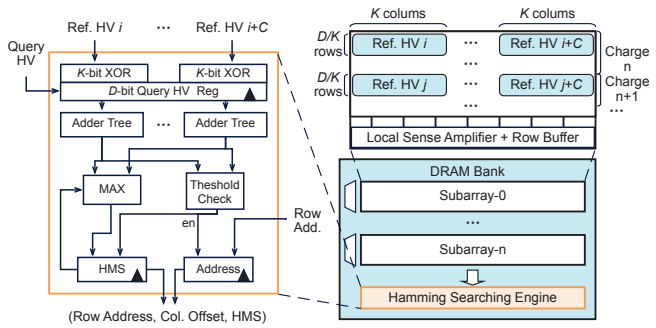


Fig. 9: Near-memory HSE design.

D/K consecutive rows, with each row occupying K columns as shown in Fig. 9. This arrangement simplifies reference indexing and allows for continuous memory access. To further streamline the process, we (1) organize the data by grouping references from the same spectrum charge (see Section IV-E) and (2) sort spectra based on their precursor m/z values within each spectrum charge. This enables all Hamming search banks to share the same address space of reference HVs, with the memory controller only needing to send identical starting and ending row addresses to all banks. Each bank then performs an independent search within the given address range. To achieve this, each HSE within each bank requires its independent memory controller and address generator to execute asynchronously. The additional overhead of the bank controller and address generator is negligible [45].

The HSE involves loading a row of data from a DRAM bank into a local RB with N_{col} columns, where the RB contains C $(= N_{\text{col}}/K)$ subsequences, each with a length of K bits. HSE accesses the entire row data every t_{RC} ns and generates the Hamming similarity before new data comes in t_{RC} to avoid any stalls between consecutive DRAM row accesses.

The encoded query bits are broadcast to each bank through the DQ bus and cached in the query HV register with the same size as N_{col}. The query HV is then XORed with the C loaded reference subsequences in parallel using K-bit XOR modules. To reduce latency, the XORed results pass through C adder trees, each consisting of K bit-serial adders. Although many adders are used, the design has low hardware complexity because most adders' bit width is low. The obtained partial Hamming similarities are checked to see if they satisfy the threshold condition in HV folding. If the similarity values for subsequences are all less than T, the row for the next pairs of reference HVs is loaded. Otherwise, for pairs that do not satisfy the remaining K-bit reference HV, it continues to be loaded and compute θ_D. The results are compared with the current best stored in the Hamming similarity (HMS) buffer, and the HMS buffer is updated with the maximum Hamming similarity. Meanwhile, the associated row address and column offset are recorded in the address buffer. In the following, we present a data organization optimization scheme that tackles the hardware under-utilization challenges caused by the reference HV filtration.

E. Data Organization Optimization

Efficient performance in the HyperOMS accelerator heavily relies on the data allocation since two filtration steps, spectrum

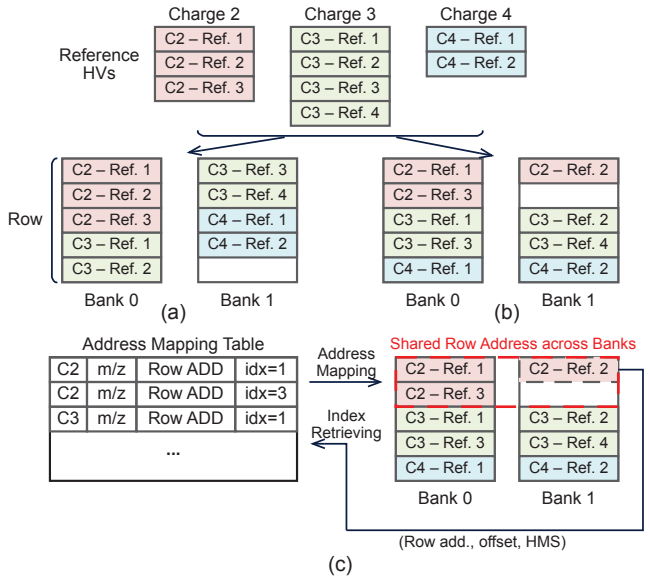


Fig. 10: (a) Uniform mapping. (b) Proposed reference data mapping scheme. (c) Address mapping process and reference index retrieving process based on (b).

charge and precursor m/z filter, which may cause workload imbalance and low bank utilization. The dispatcher distributes the encoded query HV to banks based on its spectrum charge and precursor m/z. Therefore, we optimize HyperOMS's data organization in two aspects: (1) the reference data mapping and (2) the query dispatching scheme.

1) Reference Data Mapping: The naive way to map the reference HVs is uniformly distributing the reference data to each bank in a round-robin manner. Fig. 10(a) gives an example of mapping nine reference HVs from spectrum charge +2 (C2) to +4 (C4) to two memory banks. For each spectrum charge, HVs are sorted based on their precursor m/z. The uniform mapping scheme allocates HVs to each bank based on spectrum charges. Here, each bank is unable to access references HVs from all spectrum charges, leading to low bank utilization. For example, only one bank is activated during the search process for query with C2 or C4 because only Bank 0 or Bank 1 contains HVs of C2 or C4, yielding only 50% bank utilization.

We propose the **reference data mapping scheme** in Fig. 10(b) to increase bank utilization. The reference HVs are assigned across the bank such that each DRAM bank stores HVs from all spectrum charges. By distributing the reference HVs to different banks, the proposed data mapping scheme enhances the bank utilization for queries from different spectrum charges. In this case, the total searching time is evenly amortized to each memory bank, thus reducing the processing latency. It also effectively reduces the overhead of the query dispatching policy.

2) Query Dispatching Policy: The dispatcher needs to properly allocate the given query to target banks to maximize bank utilization. Also, it needs to generate memory commands with addresses compatible with current DRAM standards [42] based on the query's spectrum charge and precursor m/z. As such, we design **query dispatching policy** to cooperate with the proposed reference data mapping scheme. As shown in Fig. 10(c), the addresses and indexes for reference HVs are

stored in the address mapping table within the dispatcher, which is designed to be compatible with current DRAM standards [42]. The address mapping table needs to store the information (spectrum charge, m/z, row address, and index) for the first memory bank (Bank 0) since the consecutive reference HVs across different banks share the same row address space. For example, Ref.1 and Ref.2 of spectrum charge +2 (C2) have identical row addresses. The dispatcher generates the starting and ending row addresses of Hamming search based on the filtering results of the query's spectrum charge and precursor m/z. The addresses with control signals are broadcast to all search banks via the C/A bus. All Hamming search banks perform Hamming similarity searches in parallel.

The Hamming similarity search returns the most similar reference HV stored in each bank. The dispatcher needs to retrieve the index of reference HVs. Fig. 10(c) shows how we retrieve the index. Assuming Bank 1 finds C2 Ref.2 as the best-matched result, Bank 1 returns the tuple (Row address, offset, HMS) that represents the associated row address, column offset, and Hamming similarity. The dispatcher first retrieves the reference index $idx = 1$ in the first bank by accessing the address mapping table. Since the dispatcher knows the bank address, then the index (idx) of C2 Ref.2 is computed using $idx +$ Bank address $+$ offset. The offset is used to index the two reference HVs organized over the same DRAM row in Fig. 9. Similarly, the reference information from other banks can be inferred based on the returned row address and offset.

The dispatcher has an overhead from the buffer, as it needs to store the address mapping table. Assume that the size of reference data is N, each entry takes $3+32+\lceil \log_2 N \rceil$ bits in the buffer, where 3 is for spectrum charge information, 32 is for m/z in a single FP32 number, and $\log_2 N$ is for index bits. Here, the row address does not need to be explicitly stored since it is identical to the entry address in the buffer. If we have B DRAM banks, the required buffer size is $\frac{N}{B} \times (3 + 32 + \lceil \log_2 N \rceil)$. For the HEK293 dataset in Table II with around 3 million reference spectra, the estimated table size is 163KB for 128 banks. The buffer size decreases linearly as the number of banks increases, which is attractive for large-scale processing.

V. Evaluation

A. Methodology

PIM Design. The specifications for DRAM and HyperOMS are summarized in Table I. The required number of DRAM chips is determined by the number of reference spectra. The reference library [46] of Kim2014 dataset [47] in Table II contains approximately 4.2 million reference spectra, which require 8 chips with a total of 256 DRAM banks to store all reference data. We use the 22nm DRAM process with DDR4-2400 standard [42], and the DRAM parameters are summarized in Table I. Among all banks, we assign one bank for encoding while the rest perform the Hamming similarity search. This is because encoding consumes only 5% of the total execution time. More DRAM banks can be added to support a larger dataset.

The HSE components of HyperOMS are implemented using Verilog HDL and synthesized using the Synopsys Design Compiler, using the TSMC 28nm technology node. The clock is set to 1.2GHz, matching the IO clock frequency of DRAM. To provide a fair comparison, we scale the area and power to 22nm to align with the memory technology. Moreover, since DRAM generally uses fewer metal layers compared to the generic ASIC, we evaluate the overhead caused by the process difference between logic and DRAM using the method in [48]. The dispatcher uses a 256KB SRAM to store the address mapping table, and the area and power are estimated using CACTI-3DD [49].

TABLE I: System specifications of HyperOMS.

DRAM Parameters	
DDR4-2400, 4Gb $\times 8$, Rows = 2^{15}, Row size = 1KB 4 banks/bank group, 4 bank groups, tCLK=1.2GHz	
DRAM Timing	
nRC=55, nRP=16, nRAS=39, nCCDS=4, nCCDL=6 nWR=18, nWTRS=3, nWTRL=9, nRTP=12, nFAW=26	
HyperOMS	
Total 256 banks and 8GB (16 chips, each chip with 16 banks)	
NMP Components	
HSE	Area per HSE: 42,680um^2 Total area: 10.93mm^2, Peak Power: 10,926mW
Dispatcher	256KB SRAM, Word size = 64b Area: 0.234mm^2, Power:77.4mW

We develop a Python in-house simulator to emulate the behavior of the HyperOMS accelerator. The latency and energy parameters of DRAM operations [29], [50] and additional near-memory ASIC components are loaded into the developed simulator to calculate runtime and energy consumption for HD encoding and searching. The OMS latency and energy numbers are calculated in two steps: First, we generate the memory traces from HyperOMS 's GPU implementation. Then, the traces and HD encoding/searching parameters are used to calculate the overall OMS latency and energy.

System Environment. The evaluation was performed on a system equipped with Intel i7-11700K with 64GB RAM and NVIDIA Geforce RTX 4090. Since the GPU has limited memory, we split the reference and the query data into batches. We set the batch size to use the maximum amount of VRAM for GPU-based solutions. We measured the GPU power consumption using `nvidia-smi`.

Workloads. We evaluated HyperOMS on three real-world datasets. The datasets are commonly used for benchmarking OMS tool performance [1], [5] and include decoy spectra with the same ratio as the existing spectral libraries using the shuffle-and-reposition method [51] for FDR filtering, which removes false-positive identifications. **iPRG2012 dataset** combines yeast spectral library [52] with the human HCD spectral library (total spectra: $1,162,392$) as the reference libraries, and the iPRG2012 dataset [53] (total spectra: $15,867$) as a query. For **HEK293 dataset**, we used the human spectral library [8] (total spectra: $2,992,672$) and a HEK293 (Human Embryonic Kidney 293) dataset [2] (total spectra: $46,665$), as reference and query spectra, respectively. Also, we reanalyze a sampled **Kim2014 dataset** [47] (total spectra: $9,976$ on average) using MassIVE-KB [46] with decoy spectra (total spectra: $4,197,746$) as a reference library. We preprocessed query and reference spectra in a similar fashion to existing works [1], [5], [12], using the widely used configurations listed in Table II.

TABLE II: Spectrum preprocessing settings.

	Dataset		
	iPRG2012	HEK293	Kim2014
Max peaks in spectra	50		
Min/max m/z	101/1500		
Bin Size	0.05	0.04	0.05
Precursor m/z tolerance (narrow)	20ppm	5ppm	10ppm
Precursor m/z tolerance (wide)	500Da	500Da	500Da

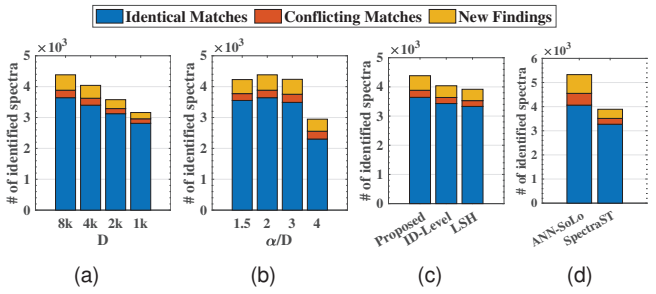


Fig. 11: Search quality comparison on the iPRG2012 dataset. (a) HyperOMS with different HV dimensionality. (b) HyperOMS with different number of bit flips. (c) HyperOMS with various encoding strategies. (d) Search quality of baseline tools.

Benchmarks. We compare the search quality of HyperOMS to existing search tools, including (1) SpectraST [12] on CPU and (2) the state-of-the-art OMS tool, ANN-SoLo, running on GPU [1]. We count the number of identifications to compare the search quality in the same way as the state of the art. All search results are evaluated at a fixed 1% FDR threshold.

B. Impact of Encoding Configuration

HV dimensionality. The HV dimensionality (D) plays a critical role in search quality. A low dimensionality limits separability. As demonstrated in Fig. 11(a), the higher D leads to a higher number of identifications. However, the excessive D leads to an increase in computation and capacity demand. We set the D to 8192 (8k) which is equal to the number of columns in the DRAM subarray, which satisfies both the hardware efficiency and the search quality.

Flipped bits. The number of flipped bits (α) controls the balance between orthogonality and correlation between bins. A high α increases the orthogonality of each position, and a low α helps a more number of adjacent bins to have a correlation (spatial locality). We measured the number of identifications according to the ratio of flips to D, i.e., α/D. As shown in Fig. 11(b), an adequate α/D leads to high search quality. But if we flip a small number of bits, HyperOMS cannot clearly differentiate the peak position. Also, since the peak shifts due to PTMs are not significant, spatial locality for a limited range is required. In the rest of our evaluation, we use $\alpha = D/2$, which shows the highest quality.

Encoding strategy. We compare the search quality of HyperOMS with the different binary encoding strategies. As discussed in Section III-B, LSH [18], [36], ID-Level HDC encoding [37], [38] can be used alternatively to encode raw data to HD binary vectors. Fig. 11(c) compares the search quality of HyperOMS with the (1) proposed encoding method, (2) ID-Level HDC encoding that can capture the position of feature and its value, and (3) random projection-based LSH approach. Our encoding offers the best search quality compared to baselines.

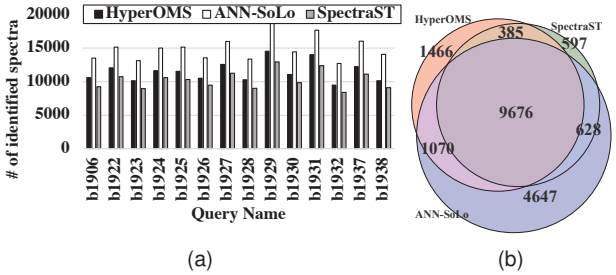


Fig. 12: Search result analysis from HyperOMS and baseline tools on the HEK293 dataset. (a) Total number of identifications. (b) Venn diagram of b1927 query search result.

Quantization level. High quantization levels may not be flexible to the peak intensity changes due to noise and PTMs. Low Q leads to low resolution in intensity capturing of the encoder. The quantization level Q did not significantly affect the search quality unless it falls in the range of $[8, 32]$. Therefore, we use $Q = 16$.

C. Search Quality

We search the iPRG2012 dataset against the yeast spectral library. As no ground truth information is available when analyzing complex biological data, instead, we compare our search quality with a list of consensus identifications produced by multiple search tools during the iPRG2012 study [53]. Fig. 11(d) shows the search result of baseline tools. Among 7841 identifications in the iPRG2012 consensus result, HyperOMS is able to correctly identify 4141 spectra (Fig. 11(b)). SpectraST and ANN-SoLo manage to identify 3891 and 5327 identifications, respectively.

We compare the performance of HyperOMS with the results from existing tools, including SpectraST and ANN-SoLo using the HEK293 dataset. We use similar configurations for all tools, listed in Table II. Fig. 12 shows the number of identifications from the different search tools. HyperOMS offers a higher search quality than SpectraST, i.e., more identified spectra. ANN-SoLo managed to identify more spectra than our HyperOMS. Nevertheless, as described in Fig. 12(b), HyperOMS can identify spectra that other tools can find (overlapped area). HyperOMS represents spectra in a way robust to PTMs, which is approximated form of the original data. Compared to ANN-SoLo, which uses floating-point representation, HyperOMS shows slightly degraded search quality.

The identification rate of HyperOMS can be improved by increasing the HV capacity. This can be done by (1) increasing the HV dimensionality D or (2) increasing the number of bits of each component in the HV. For example, increasing D from 8k (8192) to 16k (16384) can yield up to 10% more identifications. However, it raises the hardware cost, computational complexity, and energy consumption of the accelerator. Since our main goal is to maximize the speed and energy efficiency of the OMS while achieving reasonable quality in a biological sense, we use a $D = 8192$.

A ramification of lower search quality could be missing potentially relevant biomarker proteins in the context of a

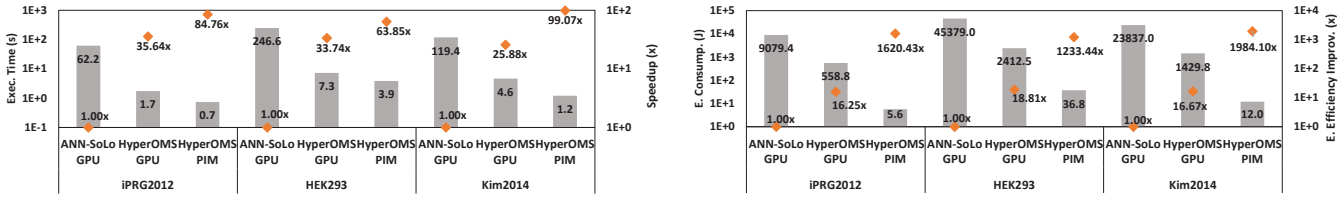


Fig. 13: Performance and energy efficiency comparison.

healthy versus diseased study, or missing data similarly impacting other downstream biological interpretations. Nevertheless, the HyperOMS identification rate is within the range of the state-of-the-art in MS identification. For example, we can typically expect an identification rate of 33–66% currently for human samples that we have used, and HyperOMS satisfies the expected range criterion. One advantage of HyperOMS is that there is a search quality–efficiency trade-off that can be tuned using the hyperparameters. The user can decide between different search engines based on their requirements; HyperOMS runs much faster with superior energy efficiency compared to competing OMS tools (Section V-D). It could be used to efficiently process extremely large proteomics datasets consisting of tens of thousands of query files, which are being generated increasingly often recently.

D. HyperOMS Performance Comparison on GPU

We compare the execution time and the energy consumption of HyperOMS on GPU to the state-of-the-art OMS tool ANN-SoLo, which offers a faster search speed with the GPU acceleration than other baseline [1]. Note that we measured the second run of the ANN-SoLo since the reference data is likely to be pre-encoded in reality. ANN-SoLo saves the pre-indexed information of the reference library on the first run and reuses it in the subsequent run. For the HEK293 and Kim2014 dataset, we averaged the measurements from multiple queries.

Fig. 13 compares the end-to-end runtime. The HyperOMS encoding is parallelized over HV dimensions and datapoints. HyperOMS uses HD binary vector and easily parallelizable Hamming similarity computation, while ANN-SoLo uses FP32 vector. The search process of HyperOMS on GPU achieves on average $46\times$ speedup ANN-SoLo [1]. However, HyperOMS needs the encoding of query and reference spectra. Nevertheless, for the end-to-end execution, HyperOMS on GPU gains an average speedup of $31\times$ over the state-of-the-art OMS tool running on the same GPU.

HyperOMS running on the GPU requires more power than the ANN-SoLo, as it has high parallelism. However, the increased power consumption is compensated by reduced execution time, improving energy efficiency. Overall, HyperOMS results in $17\times$ energy efficiency improvement on average (see Fig. 13).

E. Performance Improvement of HyperOMS Accelerator

We evaluated the speed and efficiency of the HyperOMS accelerator. We set the HV folding parameter to $K = D/2$ and $T = 0.05$ based on the experiment in Section V-F. The speed and energy consumption are shown in Fig. 13. Compared to HyperOMS on GPU, our DRAM-based accelerator offers $2.4\times$, $1.9\times$, and $3.8\times$ speedup on iPRG2012, HEK293, and

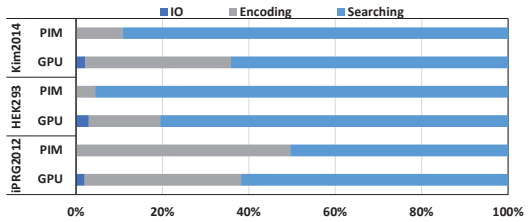


Fig. 14: HyperOMS runtime breakdown.

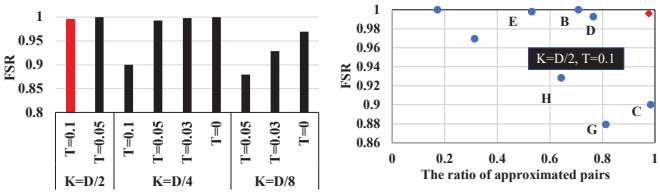


Fig. 15: HV folding parameter search. (a) HV folding success rate (FSR) according to K, T. (b) Parameter selection.

Kim2014 datasets, respectively. Since modern GPUs offer substantial memory bandwidth, the speedup from a PIM-based accelerator can be limited. However, most computing units are underutilized as HyperOMS is composed of bit-wise operations. As such, our accelerator significantly improves the energy efficiency by up to $119\times$, which is mainly powered by the energy-efficient datapath and light-weight computing units. During the search process, the query HV stays stationary in the HSE's register while the reference HVs only need to be fetched from DRAM once. The in-situ similarity search in DRAM banks and HSE dramatically reduces the required time and energy for data movement. The HV folding scheme also saves about 50% energy spent on Hamming similarity computation.

Unlike conventional HDC-based algorithms, we tackle the large-scale pattern-matching problem. Hence, the search stage rather than the encoding stage occupies a larger percentage of the total runtime. As shown in Fig. 14, our PIM design improves both stages. The accelerator offers $5.3\times$ speedup on average for the encoding stage compared to GPU. The Hamming similarity search is $2.3\times$ faster than GPU because we extended the achievable bandwidth for HSE.

F. HV Folding Parameter Search

Fig. 15(a) compares the FSR for different combinations of K and T across datasets used for evaluation. Increasing T makes the approximation in HV folding less accurate, resulting in a low FSR. Additionally, as K decreases, we need to set a lower T to maintain a high FSR. However, for low T, the chances of computing θ_D increase. As shown in Fig. 15(b), the best performance is achieved by setting K to $D/2$ and T to 0.05, with 97.6% of query-reference pairs has a 99.6% of FSR.

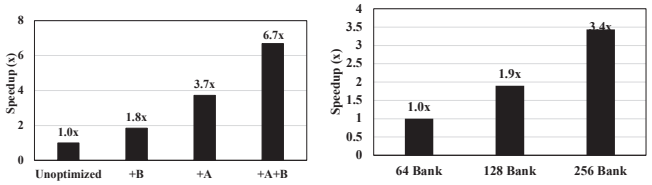


Fig. 16: Speed comparison of HyperOMS with different configurations. (a) The cumulative effects of optimization strategies (A: Data organization optimization, B: HV folding). (b) Impact of the number of banks.

When two subsequences are processed in parallel ($K = D/2$), the probability of successful folding is $0.976^2 = 0.952$, as both pairs need to satisfy the threshold check in HSE. Setting $K = D/4$ can improve throughput since more queries can be processed in parallel. However, the accelerator will be underutilized, with a probability of $1 - (0.765)^4 = 0.658$. Hence, we choose $K = D/2$ and $T = 0.05$. Note that the optimal K and T were consistent across datasets used for our experiment.

G. Effectiveness of Optimization Schemes

We show the effectiveness of our optimization strategies on our 128-bank accelerator design. Fig. 16(a) shows speedup according to the optimization schemes. The unoptimized design denotes the HyperOMS accelerator using the uniform reference data mapping in Fig. 10(a) and without the HV folding.

HV folding. We first examine the performance gain by applying the HV folding scheme. For the $K = D/2$ case, 48.8% of Hamming similarity computation is saved, offering nearly $2\times$ speedup as the HSE can accommodate more parallelism. As shown in Fig. 15, decreasing K can lower the probability of passing the threshold check module. We compare the execution time and the energy consumption of the HSE with $K = D/2$ and $K = D/4$. $K = D/4$ allows the HSE to process four pairs in parallel. All pairs need to pass the threshold check to maintain parallelism. In contrast, when $K = D/2$, the probability of all pairs passing the threshold check module is higher than the $K = D/4$ case (see Section V-F), yielding 26% shorter execution time and 25% higher energy efficiency.

Data organization optimization. Secondly, we observe the effectiveness of the proposed data organization optimization. The dispatcher efficiently distributes the query to memory banks and helps to maintain nearly full hardware utilization. It leads to $3.7\times$ speedup on 128-bank configuration. After applying all optimization strategies, the cumulative speedup over the unoptimized design is $6.7\times$.

H. Scalability and Overhead Analysis

Scalability. Our DRAM-based HyperOMS accelerator provides scalable memory capacity and bandwidth to effectively handle the increasing sizes of spectral data. As shown in Fig. 16(b), HyperOMS accelerator runs faster when the number of banks increases. For example, the accelerator with the 128 bank configuration is $1.9\times$ faster than the 64 bank case. Note that increasing banks helps reduce the search time linearly but does not change the encoding time since the number of encoding banks is fixed, leading to a sub-linear speedup. Since the dispatcher buffer size is mainly determined by the reference data size and inversely proportional to the number of banks as analyzed in Section IV-E, when we raise the number of banks while keeping the data size unchanged, the required buffer size decreases, which is beneficial for OMS with a large-sized database. Considering that the HyperOMS is bounded by memory bandwidth and needs a large memory capacity to cope with expanding the database, the proposed accelerator is a promising solution.

Area overhead. We use the 4Gb $\times 8$ DDR4 DRAM chip in [54] as the baseline to study the area overhead of HyperOMS's peripheral circuits. The DRAM chip size [54] is scaled to 22nm. The 5.462mm^2 HSEs contribute to an additional 1.67% overhead of the overall chip size. The low area overhead results from the low-complexity bit-serial adder trees. The other area overhead is from the isolation transistors to realize fast row copy across subarrays. According to [43], the additional area is less than 1%.

Energy overhead. The energy overhead of HyperOMS is evaluated by comparing the energy consumed by each HSE with DRAM's row activation energy. Each HSE only incurs additional 3.62% energy compared to the row activation energy; each HSE only needs 16 cycles to compute the Hamming similarity and corresponding the threshold check, implying that the HSE is idle over 60% of the time. Thus, the infrequent circuit switching leads to a low energy overhead.

VI. Conclusion

In this paper, we proposed HyperOMS, which accelerates OMS in MS-based proteomics by leveraging HDC. HyperOMS algorithm encodes spectra into binary HVs, considering spatial and value locality of peaks. It maximizes the computation efficiency by replacing floating-point operations in OMS with Boolean operations. We further accelerate HyperOMS on a DRAM-based PIM accelerator. The proposed accelerator exploits PuM and NMP technologies to deal with latency-area overhead trade-offs based on each stage's characteristics. In addition, we optimize the data organization and propose HV folding that can increase the hardware utilization and accommodate more parallelism in the proposed accelerator. Our evaluation results show that HyperOMS offers comparable search quality to existing OMS tools. HyperOMS PIM accelerator provides up to $3.8\times$ speedup and $119\times$ better energy efficiency over HyperOMS running on the GPU. Compared to the state-of-the-art GPU-based OMS tool, the proposed accelerator is up to $99\times$ faster and $1,984\times$ more energy efficient.

Acknowledgments

We thank Jeong-Hoon Kim and Minxuan Zhou at UCSD for valuable comments. We also appreciate Ameen Akel, Justin Eno, and Ken Curewitz at Micron Technology for their valuable comments on the performance profiling. This work was supported in part by PRISM and CoCoSys, centers in JUMP 2.0, an SRC program sponsored by DARPA; SRC Global Research Collaboration (GRC) grant; and National Science Foundation (NSF) grants #1826967, #1911095, #2052809, #2112665, #2112167, and #2100237.

REFERENCES

[1] W. Bittremieux, K. Laukens, and W. S. Noble, "Extremely fast and accurate open modification spectral library searching of high-resolution mass spectra using feature hashing and graphics processing units," *Journal of Proteome Research*, vol. 18, no. 10, pp. 3792–3799, Aug. 2019.

[2] J. M. Chick, D. Kolippakkam, D. P. Nusinow, B. Zhai, R. Rad, E. L. Huttlin, and S. P. Gygi, "A mass-tolerant database search identifies a large proportion of unassigned spectra in shotgun proteomics as modified peptides," *Nature Biotechnology*, vol. 33, no. 7, pp. 743–749, Jun. 2015.

[3] S. Na and E. Paek, "Software eyes for protein post-translational modifications," *Mass Spectrometry Reviews*, vol. 34, no. 2, pp. 133–147, Apr. 2015.

[4] M. Heck and B. A. Neely, "Proteomics in non-model organisms: A new analytical frontier," *Journal of Proteome Research*, vol. 19, no. 9, pp. 3595–3606, 2020, pMID: 32786681.

[5] W. Bittremieux, P. Meysman, W. S. Noble, and K. Laukens, "Fast open modification spectral library searching through approximate nearest neighbor indexing," *Journal of Proteome Research*, vol. 17, no. 10, pp. 3463–3474, Sep. 2018.

[6] Y. Perez-Riverol, J. Bai, C. Bandla, D. García-Seisdedos, S. Hewapathirana, S. Kamatchinathan, D. J. Kundu, A. Prakash, A. Frericks-Zipper, M. Eisenacher *et al.*, "The pride database resources in 2022: a hub for mass spectrometry-based proteomics evidences," *Nucleic acids research*, vol. 50, no. D1, pp. D543–D552, 2022.

[7] W. Xu, J. Kang, and T. Rosing, "A near-storage framework for boosted data preprocessing of mass spectrum clustering," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 313–318.

[8] M. Wang, J. Wang, J. Carver, B. S. Pullman, S. W. Cha, and N. Bandeira, "Assembling the community-scale discoverable human proteome," *Cell Systems*, vol. 7, no. 4, pp. 412–421.e5, Oct. 2018.

[9] (2022) MassIVE: Mass Spectrometry Interactive Virtual Environment. [Online]. Available: https://massive.ucsd.edu/

[10] A. T. Kong, F. V. Leprevost, D. M. Avtonomov, D. Mellacheruvu, and A. I. Nesvizhskii, "MSFragger: ultrafast and comprehensive peptide identification in mass spectrometry–based proteomics," *Nature Methods*, vol. 14, no. 5, pp. 513–520, Apr. 2017.

[11] A. Devabhaktuni, S. Lin, L. Zhang, K. Swaminathan, C. G. Gonzalez, N. Olsson, S. M. Pearlman, K. Rawson, and J. E. Elias, "TagGraph reveals vast protein modification landscapes from large tandem mass spectrometry datasets," *Nature Biotechnology*, vol. 37, no. 4, pp. 469–479, Apr. 2019.

[12] H. Lam, E. W. Deutsch, J. S. Eddes, J. K. Eng, N. King, S. E. Stein, and R. Aebersold, "Development and validation of a spectral library searching method for peptide identification from MS/MS," *PROTEOMICS*, vol. 7, no. 5, pp. 655–667, Mar. 2007.

[13] J. Kang, W. Xu, W. Bittremieux, and T. Rosing, "Massively parallel open modification spectral library searching with hyperdimensional computing," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '22. New York, NY, USA: Association for Computing Machinery, 2023, p. 536–537.

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

[15] L. Wu, R. Sharifi, M. Lenjani, K. Skadron, and A. Venkat, "Sieve: Scalable in-situ dram-based accelerator designs for massively parallel k-mer matching," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 251–264.

[16] W. Huangfu, X. Li, S. Li, X. Hu, P. Gu, and Y. Xie, "Medal: Scalable dimm based near data processing accelerator for dna seeding algorithm," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 587–599.

[17] Z. Zou, H. Chen, P. Poduval, Y. Kim, M. Imani, E. Sadredini, R. Cammarota, and M. Imani, "Biohd: an efficient genome sequence search platform using hyperdimensional memorization," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 656–669.

[18] M. Imani, S. Pampana, S. Gupta, M. Zhou, Y. Kim, and T. Rosing, "Dual: Acceleration of clustering algorithms using digital-based processing in-memory," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 356–371.

[19] M. Zhou, W. Xu, J. Kang, and T. Rosing, "Transpim: A memory-based acceleration via software-hardware co-design for transformer," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 1071–1085.

[20] W. Xu, S. Gupta, N. Moshiri, and T. Rosing, "Rapidx: High-performance reram processing in-memory accelerator for sequence alignment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[21] S. Roy, M. Ali, and A. Raghunathan, "Pim-dram: Accelerating machine learning workloads using processing in commodity dram," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 4, pp. 701–710, 2021.

[22] 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 *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 27–39.

[23] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin *et al.*, "Hardware architecture and software stack for pim based on commercial dram technology: Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 43–56.

[24] J. E. Elias and S. P. Gygi, "Target-decoy search strategy for mass spectrometry-based proteomics," *Proteome bioinformatics*, pp. 55–71, 2010.

[25] Y. Li, H. Chi, L. Xia, and X. Chu, "Accelerating the scoring module of mass spectrometry-based peptide identification using GPUs," *BMC Bioinformatics*, vol. 15, no. 1, p. 121, 2014.

[26] M. Yang, T. Chen, X. Zhou, L. Zhao, Y. Zhu, and L. Wang, "A complete cpu-fpga architecture for protein identification with tandem mass spectrometry," in *2019 International Conference on Field-Programmable Technology (ICFPT)*, 2019, pp. 295–298.

[27] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *arXiv preprint arXiv:1702.08734*, 2017.

[28] Y.-C. Kwon, S. H. Lee, J. Lee, S.-H. Kwon, J. M. Ryu, J.-P. Son, O. Seongil, H.-S. Yu, H. Lee, S. Y. Kim, Y. Cho, J. G. Kim, J. Choi, H.-S. Shin, J. Kim, B. Phuah, H. Kim, M. J. Song, A. Choi, D. Kim, S. Kim, E.-B. Kim, D. Wang, S. Kang, Y. Ro, S. Seo, J. Song, J. Youn, K. Sohn, and N. S. Kim, "25.4 a 20nm 6gb function-in-memory DRAM, based on HBM2 with a 1.2tflops programmable computing unit using bank-level parallelism, for machine learning applications," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*. IEEE, Feb. 2021.

[29] M. F. Ali, A. Jaiswal, and K. Roy, "In-memory low-cost bit-serial addition using commodity dram technology," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 1, pp. 155–165, 2019.

[30] 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*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 100–113.

[31] N. Hajinazar, G. F. Oliveira, S. Gregorio, J. D. Ferreira, N. M. Ghiasi, M. Patel, M. Alser, S. Ghose, J. Gómez-Luna, and O. Mutlu, "Simdram: a framework for bit-serial simd processing using dram," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 329–345.

[32] S. Gupta, M. Imani, and T. Rosing, "Felix: Fast and energy-efficient logic in memory," in *Proceedings of the International Conference on Computer-Aided Design*, 2018.

[33] T. F. Wu, H. Li, P.-C. Huang, A. Rahimi, J. M. Rabaey, H.-S. P. Wong, M. M. Shulaker, and S. Mitra, "Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, 2018, pp. 492–494.

[34] H. Li, T. F. Wu, A. Rahimi, K.-S. Li, M. Rusch, C.-H. Lin, J.-L. Hsu, M. M. Sabry, S. B. Eryilmaz, J. Sohn, W.-C. Chiu, M.-C. Chen, T.-T. Wu, J.-M. Shieh, W.-K. Yeh, J. M. Rabaey, S. Mitra, and H.-S. P. Wong, "Hyperdimensional computing with 3d vrram in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition," in *2016 IEEE International Electron Devices Meeting (IEDM)*, 2016, pp. 16.1.1–16.1.4.

[35] J. Kang, M. Zhou, A. Bhansali, W. Xu, A. Thomas, and T. Rosing, "Relhd: A graph-based learning on fefet with hyperdimensional computing," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*. IEEE, 2022, pp. 553–560.

[36] M. Norouzi, D. J. Fleet, and R. R. Salakhutdinov, "Hamming distance metric learning," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012.

[37] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *2017 IEEE*

International Conference on Rebooting Computing (ICRC). IEEE, 2017, pp. 1–8.

[38] S. Salamat, M. Imani, B. Khaleghi, and T. Rosing, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2019, pp. 53–62.

[39] A. Kertesz-Farkas, U. Keich, and W. S. Noble, "Tandem mass spectrum identification via cascaded search," *Journal of Proteome Research*, vol. 14, no. 8, pp. 3027–3038, Aug. 2015.

[40] J. Kang, B. Khaleghi, T. Rosing, and Y. Kim, "Openhd: A gpu-powered framework for hyperdimensional computing," *IEEE Transactions on Computers*, vol. 71, no. 11, pp. 2753–2765, 2022.

[41] J. Kang, B. Khaleghi, Y. Kim, and T. Rosing, "Xcelhd: An efficient gpu-powered hyperdimensional computing with parallelized training," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022, pp. 220–225.

[42] D. S. Standard and D. SDRAM, "Standard jesd79-4b," *JEDEC, Jun*, 2017.

[43] K. K. Chang, P. J. Nair, D. Lee, S. Ghose, M. K. Qureshi, and O. Mutlu, "Low-cost inter-linked subarrays (lisa): Enabling fast inter-subarray data movement in dram," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 568–580.

[44] B. Kim, J. Chung, E. Lee, W. Jung, S. Lee, J. Choi, J. Park, M. Wi, S. Lee, and J. H. Ahn, "Mvid: Sparse matrix-vector multiplication in mobile dram for accelerating recurrent neural networks," *IEEE Transactions on Computers*, vol. 69, no. 7, pp. 955–967, 2020.

[45] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A case for exploiting subarray-level parallelism (salp) in dram," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2012, pp. 368–379.

[46] (2022) MassIVE-KB. [Online]. Available: https://massive.ucsd.edu/ProteoSAFe/static/massive.jsp

[47] M.-S. Kim, S. M. Pinto, D. Getnet, R. S. Nirujogi, S. S. Manda, R. Chaerkady, A. K. Madugundu, D. S. Kelkar, R. Isserlin, S. Jain, J. K. Thomas, B. Muthusamy, P. Leal-Rojas, P. Kumar, N. A. Sahasrabuddhe, L. Balakrishnan, J. Advani, B. George, S. Renuse, L. D. N. Selvan, A. H. Patil, V. Nanjappa, A. Radhakrishnan, S. Prasad, T. Subbannayya, R. Raju, M. Kumar, S. K. Sreenivasamurthy, A. Marimuthu, G. J. Sathe, S. Chavan, K. K. Datta, Y. Subbannayya, A. Sahu, S. D. Yelamanchi, S. Jayaram, P. Rajagopalan, J. Sharma, K. R. Murthy, N. Syed, R. Goel, A. A. Khan, S. Ahmad, G. Dey, K. Mudgal, A. Chatterjee, T.-C. Huang, J. Zhong, X. Wu, P. G. Shaw, D. Freed, M. S. Zahari, K. K. Mukherjee, S. Shankar, A. Mahadevan, H. Lam, C. J. Mitchell, S. K. Shankar, P. Satishchandra, J. T. Schroeder, R. Sirdeshmukh, A. Maitra, S. D. Leach, C. G. Drake, M. K. Halushka, T. S. K. Prasad, R. H. Hruban, C. L. Kerr, G. D. Bader, C. A. Iacobuzio-Donahue, H. Gowda, and A. Pandey, "A draft map of the human proteome," *Nature*, vol. 509, no. 7502, pp. 575–581, May 2014.

[48] Y.-B. Kim and T. W. Chen, "Assessing merged dram/logic technology," *Integration*, vol. 27, no. 2, pp. 179–194, 1999.

[49] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-3dd: Architecture-level modeling for 3d die-stacked dram main memory," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2012, pp. 33–38.

[50] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Rowclone: Fast and energy-efficient in-dram bulk data copy and initialization," in *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2013, pp. 185–197.

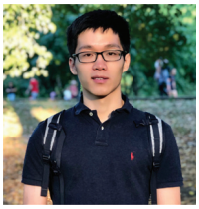
[51] H. Lam, E. W. Deutsch, and R. Aebersold, "Artificial decoy spectral libraries for false discovery rate estimation in spectral library searching in proteomics," *Journal of proteome research*, vol. 9, no. 1, pp. 605–610, 2010.

[52] N. Selevsek, C.-Y. Chang, L. C. Gillet, P. Navarro, O. M. Bernhardt, L. Reiter, L.-Y. Cheng, O. Vitek, and R. Aebersold, "Reproducible and consistent quantification of the saccharomyces cerevisiae proteome by SWATH-mass spectrometry," *Molecular & Cellular Proteomics*, vol. 14, no. 3, pp. 739–749, Mar. 2015.

[53] R. J. Chalkley, N. Bandeira, M. C. Chambers, K. R. Clauser, J. S. Cottrell, E. W. Deutsch, E. A. Kapp, H. H. N. Lam, W. H. McDonald, T. A. Neubert, and R.-X. Sun, "Proteome Informatics Research Group (iPRG)_2012: A study on detecting modified peptides in a complex mixture," *Molecular & Cellular Proteomics*, vol. 13, no. 1, pp. 360–371, Jan. 2014.

[54] K. Sohn, T. Na, I. Song, Y. Shim, W. Bae, S. Kang, D. Lee, H. Jung, S. Hyun, H. Jeoung *et al.*, "A 1.2 v 30 nm 3.2 gb/s/pin 4 gb ddr4 sdram with dual-error detection and pvt-tolerant data-fetch scheme," *IEEE journal of solid-state circuits*, vol. 48, no. 1, pp. 168–177, 2012.

Jaeyoung Kang Jaeyoung Kang received a B.E. degree in electrical engineering from Korea University, Seoul, South Korea, in 2019. Currently, he is a fourth-year Ph.D. candidate in Electrical and Computer Engineering at the University of California San Diego, La Jolla, CA, USA. His research interests include deep learning-based algorithm acceleration on heterogeneous system architecture, GPU-based acceleration for big data analysis in bioinformatics, and hyperdimensional computing.



Weihong Xu Weihong Xu received the B.E. degree in information engineering and M.E. in information and communication engineering from Southeast University, Nanjing, China, in 2017 and 2020. He is currently a third-year Ph.D. student in Computer and Computer Engineering at the University of California San Diego, La Jolla, CA, USA. His research interests include in-memory and in-storage architecture for deep learning, bioinformatics, and hyperdimensional computing.

Wout Bittremieux Wout Bittremieux received his Ph.D. degree from the University of Antwerp, Antwerp, Belgium, in 2017. He is currently an assistant research professor in the Adrem Data Lab at the University of Antwerp. His research employs advanced computational techniques to solve fundamental biological questions by developing algorithmic solutions and machine learning methods for the analysis of mass spectrometry-based proteomics and metabolomics data. He has developed several innovative tools to analyze large mass spectral data volumes, has contributed to the development of mass spectrometry data standards, and is actively engaged in the computational mass spectrometry community.

Niema Moshiri Niema Moshiri received his Ph.D. degree in Bioinformatics and Systems Biology from the University of California at San Diego, La Jolla, CA, USA, in 2019. He is an Assistant Teaching Professor in the Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA, USA. He works on computational biology, with a research focus on building massively-scalable tools to enable ultra-large real-time analyses in viral phylogenetics and molecular epidemiology. He also places a heavy emphasis on teaching, namely on the development of online educational content, primarily Massive Adaptive Interactive Texts (MAITs).

Tajana Rosing Tajana Šimunić Rosing (Fellow, IEEE) received her Ph.D. degree from Stanford University, Stanford, CA, USA, in 2001. She is a Professor, a Holder of the Fratamico Endowed Chair, and the Director of System Energy Efficiency Laboratory, University of California at San Diego, La Jolla, CA, USA. From 1998 to 2005, she was a full-time Research Scientist with HP Labs, Palo Alto, CA, USA, while also leading research efforts with Stanford University, Stanford, CA, USA. She was a Senior Design Engineer with Altera Corporation, San Jose, CA, USA. She is leading a number of projects, including efforts funded by DARPA/SRC JUMP 2.0 PRISM program with focus on design of accelerators for analysis of big data, DARPA and NSF funded projects on hyperdimensional computing, and SRC funded project on IoT system reliability and maintainability. Her current research interests include energy-efficient computing, cyber–physical, and distributed systems.