UNIVERSITY OF CALIFORNIA SAN DIEGO

High-performance Software and Hardware Designs for Genomics and Proteomics

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science (Computer Engineering)

by

Weihong Xu

Committee in charge:

Professor Tajana Šimunić Rosing, Chair
Professor Mingu Kang
Professor Niema Moshiri
Professor Jishen Zhao

2024

The Dissertation of Weihong Xu is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2024

DEDICATION

To my parents

For their endless love, support, and encouragement to go on and complete this journey

EPIGRAPH

Victory Belongs To The Most Tenacious.

—Roland Garros

Stay Hungry Stay Foolish.

—Steve Jobs

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my Ph.D. advisor, Professor Tajana Šimunić Rosing, for her invaluable guidance and unwavering support throughout my Ph.D. journey. Her mentorship has been instrumental in shaping both the direction and the outcome of my research. Without her encouragement, insightful critiques, and exceptional leadership, this dissertation would not have been possible, and the experience would not have been as enriching and memorable as it has been. I have learned immensely from her wisdom and the example she sets through her great personality. I would also like to extend my heartfelt thanks to my doctoral committee members, Professor Mingu Kang, Professor Jishen Zhao, and Professor Niema Moshiri, for their constructive feedback, which greatly enhanced the quality of this dissertation.

I have been fortunate to collaborate with esteemed researchers including Professor Shimeng Yu from Georgia Institute of Technology, Professor Niema Moshiri from UCSD, Professor Mingu Kang from UCSD, Professor Kevin Skadron from the University of Virginia, Professor Vikram Adve from UIUC, and Professor Wout Bittremieux from the University of Antwerp. Their enthusiasm for sharing ideas and their willingness to offer help were critical to the success of this interdisciplinary research. My academic journey has been enriched by the opportunity to experience a unique blend of both academic and industry-based research. I am deeply grateful to Dr. Carlos H Diaz at TSMC and Dr. Ameen Akel, Sean Eilert, Justin Eno, and Ken Curewitz at Micron. The hands-on experience and professional insights gained through our collaboration have been a significant motivation for my research.

Additionally, I would like to thank all my colleagues in SEELab at UCSD over the years, including Minxuan Zhou, Jaeyoung Kang, Sumukh Pinge, Youhak Lee, Xuan Wang, Saransh Gupta, Xiaofan Yu, Derek Jones, Flavio Ponzina, Behnam Khaleghi, Tianqi Zhang, Chang Eun Song, Haichao Yang, Keming Fan, Yue Pan, Haein Choi, Anthony Thomas, and many others. Their camaraderie and support have made this journey even more rewarding.

Last but not least, I owe an immense debt of gratitude to my family, Jinnen and Meifang,

whose constant love and support have been the cornerstone of my academic pursuits. Their unwavering belief in me has provided the strength and motivation necessary to achieve this milestone.

Research in this thesis was generously supported by funding from the National Science Foundation; CRISP, one of six centers in JUMP 1.0; PRISM and CoCoSys, centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by the Defense Advanced Research Projects Agency (DARPA); and SRC-Global Research Collaboration grants.

The material in this dissertation is based on the following publications:

Chapter 2, in full, is a reprint of the material "RAPIDx: High-performance ReRAM Processing in-Memory Accelerator for DNA Alignment", by Weihong Xu, Saransh Gupta, Niema Moshiri, and Tajana Rosing, which appears in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2023. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in full, is a reprint of the material "HyperGen: Compact and Efficient Genome Sketching using Hyperdimensional Vectors", by Weihong Xu, Po-kai Hsu, Niema Moshiri, Shimeng Yu, and Tajana Rosing, which appears in Bioinformatics, 2024. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in full, is a reprint of the material "HyperSpec: Fast Mass Spectra Clustering in Hyperdimensional Space", by Weihong Xu, Jaeyoung Kang, Wout Bittremieux, Niema Moshiri, and Tajana Rosing, which appears in Journal of Proteome Research, 2023. The dissertation author was the primary investigator and author of this paper.

Chapter 5, in full, is a reprint of the material "A Near-Storage Framework for Boosted Data Preprocessing of Mass Spectrum Clustering", by Weihong Xu, Jaeyoung Kang, and Tajana Rosing, which appears in IEEE/ACM Design Automation Conference (DAC), 2022. The dissertation author was the primary investigator and author of this paper.

VITA

2017	Bachelor of Engineering, Southeast University
2020	Master of Engineering, Southeast University
2024	Doctor of Philosophy, University of California San Diego

PUBLICATIONS

Weihong Xu, Saransh Gupta, Justin Morris, Xincheng Shen, Mohsen Imani, Baris Aksanli, and Tajana Rosing, "Tri-HD: Energy-Efficient On-Chip Learning With In-Memory Hyperdimensional Computing", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 2024.

Weihong Xu, Po-kai Hsu, Niema Moshiri, Shimeng Yu, and Tajana Rosing, "HyperGen: Compact and Efficient Genome Sketching using Hyperdimensional Vectors", Bioinformatics, 2024.

Weihong Xu, Jaeyoung Kang, and Tajana Rosing, "AttBind: Memory-Efficient Acceleration for Long-Range Attention Using Vector-Derived Symbolic Binding", Design, Automation and Test in Europe Conference (DATE), 2024.

Haichao Yang, Chang Eun Song, **Weihong Xu**, Behnam Khaleghi, Uday Mallappa, Monil Shah, Keming Fan, Mingu Kang, and Tajana Rosing, "FSL-HDnn: A 5.7 TOPS/W End-to-end Few-shot Learning Classifier Accelerator with Feature Extraction and Hyperdimensional Computing", European Solid-State Electronics Research Conference (ESSERC), 2024.

Lingxi Wu, Minxuan Zhou, **Weihong Xu**, Ashish Venkat, Tajana Rosing, and Kevin Skadron, "Abakus: Accelerating k-mer Counting With Storage Technology", ACM Transactions on Architecture and Code Optimization (TACO), 2024.

Minxuan Zhou, Yujin Nam, Pranav Gangwar, **Weihong Xu**, Arpan Dutta, Chris Wilkerson, Rosario Cammarota, Saransh Gupta, and Tajana Rosing, "MatHE: A Near-Mat Processing In-Memory Accelerator for Fully Homomorphic Encryption", ACM/IEEE Design Automation Conference (DAC), 2024.

Jaeyoung Kang, **Weihong Xu**, Wout Bittremieux, Niema Moshiri, and Tajana Rosing, "DRAM-Based Acceleration of Open Modification Search in Hyperdimensional Space", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 2024.

Jaeyoung Kang, You Hak Lee, Minxuan Zhou, **Weihong Xu**, and Tajana Rosing, "HygHD: Hyperdimensional Hypergraph Learning", Design, Automation and Test in Europe Conference (DATE), 2024.

Pinge, Sumukh, **Weihong Xu**, Jaeyoung Kang, Tianqi Zhang, Niema Moshiri, Wout Bittremieux, and Tajana Rosing, "SpecHD: Hyperdimensional Computing Framework for FPGA-based Mass Spectrometry Clustering", Design, Automation and Test in Europe Conference (DATE), 2024.

Weihong Xu, Saransh Gupta, Niema Moshiri, and Tajana Rosing, "RAPIDx: High-performance ReRAM Processing in-Memory Accelerator for DNA Alignment", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 2023.

Weihong Xu, Viji Swaminathan, Sumukh Pinge, Sean Fuhrman, and Tajana Rosing, "HyperMetric: Robust Hyperdimensional Computing on Error-prone Memories using Metric Learning", IEEE International Conference on Computer Design (ICCD), 2023.

Weihong Xu, Jaeyoung Kang, Wout Bittremieux, Niema Moshiri, and Tajana Rosing, "HyperSpec: Fast Mass Spectra Clustering in Hyperdimensional Space", Journal of Proteome Research, 2023.

Weihong Xu, Jaeyoung Kang, and Tajana Rosing, "FSL-HD: Accelerating Few-Shot Learning on ReRAM using Hyperdimensional Computing", Design, Automation and Test in Europe Conference (DATE), 2023.

Po-kai Hsu, **Weihong Xu**, Tajana Rosing, and Shimeng Yu, "An In-Storage Processing Architecture with 3D NAND Heterogeneous Integration for Spectra Open Modification Search", In Proceedings of the International Symposium on Memory Systems (MemSys), 2023.

Jaeyoung Kang, **Weihong Xu**, Wout Bittremieux, Niema Moshiri, and Tajana Rosing, "Accelerating open modification spectral library searching on tensor core in high-dimensional space", Bioinformatics,2023.

Weihong Xu, Jaeyoung Kang, and Tajana Rosing, "A Near-Storage Framework for Boosted Data Preprocessing of Mass Spectrum Clustering", IEEE/ACM Design Automation Conference (DAC), 2022.

Jaeyoung Kang, Minxuan Zhou, Abhinav Bhansali, **Weihong Xu**, Anthony Thomas, and Tajana Rosing, "Relhd: A graph-based learning on fefet with hyperdimensional computing", International Conference on Computer Design (ICCD), 2022.

Jaeyoung Kang, **Weihong Xu**, Wout Bittremieux, and Tajana Rosing, "Massively parallel open modification spectral library searching with hyperdimensional computing", International Conference on Parallel Architectures and Compilation Techniques (PACT), 2022.

Arpan Dutta, Saransh Gupta, Behnam Khaleghi, Rishikanth Chandrasekaran, **Weihong Xu**, Tajana Rosing, "Hdnn-pim: Efficient in memory design of hyperdimensional computing with feature extraction", The Great Lakes Symposium on VLSI, 2022.

Minxuan Zhou, **Weihong Xu**, Jaeyoung Kang, and Tajana Rosing, "TransPIM: A Memory-based Acceleration via Software-Hardware Co-Design for Transformers", IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2022.

Minxuan Zhou, Yunhui Guo, **Weihong Xu**, Bin Li, Kevin Eliceiri, and Tajana Rosing, "Mat: Processing in-memory acceleration for long-sequence attention", ACM/IEEE Design Automation Conference (DAC), 2021.

Xiaofan Yu, **Weihong Xu**, Ludmila Cherkasova, Tajana Rosing, "Automating Reliable and Fault-Tolerant Design of LoRa-based IoT Networks", International Conference on Network and Service Management (CNSM), 2021.

ABSTRACT OF THE DISSERTATION

High-performance Software and Hardware Designs for Genomics and Proteomics

by

Weihong Xu

Doctor of Philosophy in Computer Science (Computer Engineering)

University of California San Diego, 2024

Professor Tajana Šimunić Rosing, Chair

Genomics and proteomics are at the forefront of innovations in precision medicine and drug discovery. However, the rapid data expansion in these fields presents significant computational challenges, emphasizing the need for more efficient algorithm and hardware designs. Current research overlooks systematic acceleration from both software and hardware aspects. This dissertation bridges these gaps by presenting high-performance designs that enhance the efficiency, accuracy, and scalability of data analysis in genomics and proteomics.

Genome alignment is crucial for evaluating sequence similarity in genomics, but existing solutions are hindered by high memory footprints and computational complexity. To address these

challenges, this thesis introduces RAPIDx, an algorithm and hardware co-design that enhances the efficiency and throughput of genome alignment. RAPIDx leverages Processing-in-Memory (PIM) techniques for in-situ computation, significantly boosting energy efficiency. It also employs an adaptive banded alignment algorithm tailored for ReRAM-based PIM architectures, reducing computational complexity and memory requirements while maintaining high accuracy. The proposed PIM architecture achieves up to 131.1× and 46.8× throughput improvement over the state-of-the-art CPU and GPU implementations, respectively.

RAPIDx delivers high accuracy across various genome analysis tasks, but its substantial memory consumption makes it unsuitable for latency-sensitive scenarios or resource-constrained hardware. To address these limitations, this thesis proposes HyperGen, a memory-efficient genome sketching tool that eliminates the need for the costly alignment. HyperGen leverages hyperdimensional computing (HDC) to significantly improve runtime performance, memory efficiency, and accuracy in large-scale genomic analyses, enabling rapid and precise Average Nucleotide Identity (ANI) estimation. The tool demonstrates superior performance in both genome sketching and database search tasks.

Proteomics, using mass spectrometry (MS) to analyze proteins, provides deep insights into cellular functions and disease mechanisms. MS clustering is crucial for organizing and interpreting these datasets, enabling more efficient identification of proteins and peptides. However, the demand for accurate, fast, and scalable algorithms presents a significant challenge for large-scale analyses. To address this, this thesis introduces HyperSpec, a high-performance tool that accelerates spectral clustering by leveraging the lightweight, parallelizable nature of HDC. HyperSpec reduces clustering runtime while maintaining high quality, cutting the processing time of 21 million spectra from 4 hours to just 24 minutes.

Despite HyperSpec's significant speedup to MS clustering, our profiling analysis reveals that MS data preprocessing remains the primary bottleneck, due to the inefficient data path of conventional Von Neumann architecture. To overcome this, a near-storage accelerator, MSAS, is presented to speed up MS data preprocessing. By processing spectra close to the storage medium,

MSAS minimizes costly data movement between storage and computation units. Its channel-level design achieves up to 187× speedup compared to CPU-based preprocessing and outperforms existing in-storage computing solutions. When integrated into existing MS clustering tools, MSAS enhances overall system performance, yielding 3.5× to 9.8× improvements in speed and 2.8× to 11.9× gains in energy efficiency.

Chapter 1

Introduction

The fields of genomics and proteomics have become foundational pillars in the pursuit of precision medicine, a discipline that aims to tailor medical treatments to individual genetic profiles and molecular characteristics [19]. The integration of genomics and proteomics is essential for a comprehensive understanding of biological systems and the development of effective therapeutic strategies. While genomics provides the foundational blueprint by identifying the potential genetic underpinnings of health and disease, proteomics offers critical insights into the actual biological processes that occur at the protein level. Genomics primarily focuses on the study of DNA sequences [20], while proteomics depend heavily on mass spectrometry (MS) to decipher the complex structures of proteins and other molecules within cells [21–23].

The fields of genomics and proteomics have experienced unprecedented growth over the past few decades. For genomics, the next-generation sequencing (NGS) [24–26] has enabled researchers to generate vast amounts of genomic data at a pace and scale previously unimaginable. The applications of NGS are broad and profound, ranging from identifying genetic variants associated with diseases to understanding the evolutionary history of organisms. Similarly, in proteomics, mass spectrometry (MS) [22, 23] is a critical tool for analyzing the protein mixtures present in biological samples, facilitating the identification of biomarkers, drug targets, and therapeutic proteins. The decreasing cost of MS experiments has fueled the growth of publicly available MS data [21, 27]. For example, the MassIVE database contains over 600TB MS

data [28] for analysis.

These advancements have revolutionized our ability to generate vast amounts of data. However, this rapid expansion of data brings with it significant computational challenges. High-throughput sequencing technologies generate billions of reads, requiring analysis algorithms to process vast amounts of data efficiently. Additionally, the need to perform these computations in a timely manner, especially in clinical contexts, underscores the necessity for high-performance software and hardware solutions that can accelerate genome alignment while maintaining accuracy and scalability. On the other hand, modern MS experiments generate millions of tandem mass spectra, which are rich in information but also highly redundant [29]. The analysis of these spectra is crucial for identifying protein structures and understanding biological functions, but the process is extremely time-consuming. For instance, clustering large-scale proteomic datasets, such as a draft human proteome dataset containing 25 million spectra and 131 GB of data, can take several hours to days [12, 13, 15], posing a major bottleneck in research and drug discovery.

There is a need for innovative computational frameworks that can efficiently handle the large-scale data generated by NGS and MS technologies. These frameworks must not only accelerate the analysis process but also improve the accuracy and energy efficiency of the computations involved. This thesis addresses two fundamental problems: 1. efficient sequence similarity evaluation in genomics and 2. boosted data preprocessing and clustering in proteomics. These problems are critical for the acceleration of genomics and proteomics because they are not only central to a wide range of downstream applications but also contribute to significant runtime overhead. This thesis presents a series of high-performance software and hardware designs aimed at overcoming these computational hurdles, with the goal of enabling more effective and scalable analysis in genomics and proteomics. The contributions of this thesis can be summarized as follows:

1.1 Acceleration for Genomics

1.1.1 Processing-in-memory Acceleration for Genome Alignment

Genome alignment is critical for identifying genetic variants, understanding evolutionary relationships, and many other applications. Traditional methods for sequence alignment [30, 31] are computation- and memory-intensive because the advancement of sequencing technologies produces a tremendous amount of data, making sequence alignment a critical bottleneck in bioinformatics analysis. The existing hardware accelerators for alignment [7, 10, 32] suffer from limited on-chip memory, costly data movement, and poorly optimized alignment algorithms. They cannot afford to concurrently process the massive amount of genome data.

In Chapter 2, we propose a ReRAM-based accelerator, RAPIDx, using processing in-memory (PIM) for sequence alignment. RAPIDx achieves superior efficiency and performance via software-hardware co-design. First, we propose an adaptive banded parallelism alignment algorithm suitable for PIM architecture. Compared to the original dynamic programming-based alignments [30, 31], the proposed algorithm significantly reduces the required complexity, data bit width, and memory footprint at the cost of negligible accuracy degradation. Then we propose the efficient PIM architecture that implements the proposed algorithm. The data flow in RAPIDx achieves four-level parallelism and we design an in-situ alignment computation flow in ReRAM, delivering 5.5-9.7× efficiency and throughput improvements compared to our previous PIM design, RAPID. The proposed RAPIDx is reconfigurable to serve as a co-processor integrated into the existing genome analysis pipeline to boost sequence alignment or edit distance calculation. On short-read alignment, RAPIDx delivers 131.1× and 46.8× throughput improvements over state-of-the-art CPU [8] and GPU [7] libraries, respectively. As compared to ASIC accelerators [10] for long-read alignment, the performance of RAPIDx is 1.8-2.9× higher.

1.1.2 High-performance Software for Genome Sketching

Genome alignment is renowned for its high accuracy across a wide range of genome analysis tasks. However, its substantial memory consumption and computational demands make it impractical for latency-sensitive scenarios or resource-constrained hardware environments. To overcome these challenges, in Chapter 3, we present HyperGen that improves accuracy, runtime performance, and memory efficiency for large-scale ANI estimation. Unlike existing genome sketching algorithms [33–36] that convert large genome files into discrete k-mer hashes, HyperGen leverages the emerging hyperdimensional computing (HDC) to encode genomes into quasi-orthogonal vectors (Hypervector, HV) in high-dimensional space. HV is compact and can preserve more information, allowing for accurate ANI estimation while reducing required sketch sizes. In particular, the HV sketch representation in HyperGen allows efficient ANI estimation using vector multiplication, which naturally benefits from highly optimized general matrix multiply (GEMM) routines. As a result, HyperGen enables the efficient sketching and ANI estimation for massive genome collections. We evaluate HyperGen's sketching and database search performance using several genome datasets at various scales. HyperGen is able to achieve comparable or superior ANI estimation error and linearity compared to other sketch-based counterparts [33–36]. The measurement results show that HyperGen is one of the fastest tools for both genome sketching and database search. Meanwhile, HyperGen produces memory-efficient sketch files while ensuring high ANI estimation accuracy.

1.2 Acceleration for Mass Spectrometry-based Proteomics

1.2.1 High-performance Software for Mass Spectrometry Clustering

As current MS experiments can produce gigabytes to terabytes of data per hour [29], processing these massive data volumes has become progressively more challenging. Spectral clustering is an effective approach to speed up downstream data processing by merging highly similar spectra to minimize data redundancy. However, state-of-the-art MS tools [14, 15] take

many hours to run spectrum clustering. In Chapter 4, we present a fast spectral clustering tool based on HDC, HyperSpec. HDC shows promising clustering capability while only requiring lightweight binary operations with high parallelism that can be easily accelerated in hardware, making it possible to run HyperSpec on GPU to achieve extremely efficient spectral clustering performance. Additionally, HyperSpec includes optimized data preprocessing modules to reduce the spectrum preprocessing time, which is a critical bottleneck during spectral clustering. Based on experiments using various mass spectrometry datasets, HyperSpec produces results with comparable clustering quality as state-of-the-art spectral clustering tools [12–14, 37], while achieving speedups by orders of magnitude, shortening the clustering runtime of over 21 million spectra from 4 hours to only 24 minutes.

1.2.2 Near-storage Acceleration for Mass Spectrometry Preprocessing

The previous work demonstrate that spectra data loading and preprocessing consumes the majority of total execution time and energy during MS analysis [29]. The software-level optimizations fail to fully exploit the parallelism and memory bandwidth of low-level hardware. We propose a near-storage framework, MSAS, to further speed up spectrum preprocessing in Chapter 5. Instead of loading data into host memory and CPU, MSAS processes spectra near storage, thus reducing the expensive cost of data movement. We present two types of accelerators that leverage internal bandwidth at two storage levels: SSD and channel. The accelerators are optimized to match the data rate at each storage level with negligible overhead. Our results demonstrate that the channel-level design yields the best performance improvement for preprocessing - it is up to 187× and 1.8× faster than the CPU [13] and the state-of-the-art in-storage computing solution [18], respectively. After integrating channel-level MSAS into existing MS clustering tools, we measure system level improvements in speed of 3.5× to 9.8× with 2.8× to 11.9× better energy efficiency.

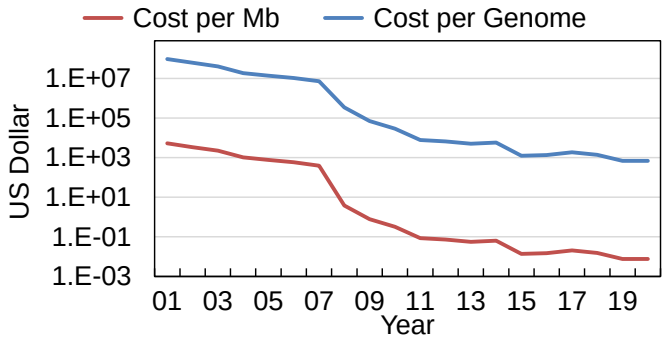
Chapter 2

Processing In-Memory Acceleration for Genome Alignment

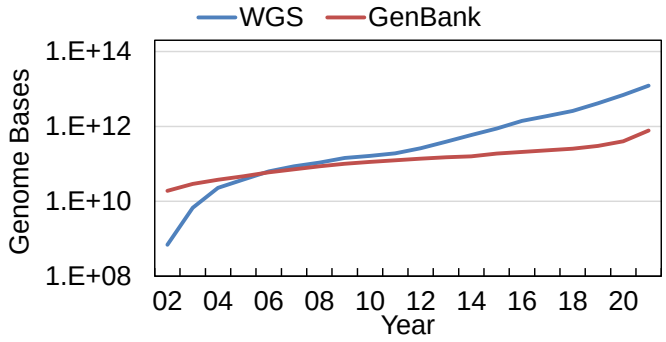
2.1 Introduction

The high-throughput next-generation sequencing (NGS) technology [38] has led to the generation of massive amounts of genomic data at an unprecedented rate. The generated genome data rely on genome alignment, a critical step in genomics [8] to identify genetic variations, disease associations, and evolutionary relationships. The significance of genome alignment continues to grow for two reasons. First, various types of sequencing errors can occur when the genome is read by sequencing machines. Additionally, genetic mutations and variations further introduce differences in sequences.

Genome alignment is the process of finding the optimal mapping between query and reference sequences. Needleman–Wunsch (NW) [30] and Smith-Waterman (SW) [31], are the two most commonly used alignment algorithms for this purpose. Genome alignment has become a significant bottleneck as it is both computation- and memory-intensive, consuming 60-80% of the runtime of popular genome analysis tools [7, 8, 11, 39]. This challenge is exacerbated by the continuous expansion in the size and complexity of genomic datasets. Figure 2.1 shows the unit cost of genome sequencing has plunged by over $10^4\times$ during the last decade. As a result, the genome data volume of whole genome sequencing (WGS) and GenBank [2] have expanded by $10^2\times$ to $10^4\times$.



(a) Squencing cost.



(b) Genome data volume.

Figure 2.1: Trend of unit sequencing cost [1] and genome data volume [2] over the past decade.

The exponential growth in genomic data has outpaced Moore's Law as well as the capabilities of traditional computational methods, necessitating acceleration solutions with orders of magnitude higher efficiency. Processing in-memory (PIM) is promising to mitigate the data movement issue and provides massive parallelism. PIM enables in-situ data computation inside memory, thereby throttling the latency and energy of data movement [40–43]. This chapter explores emerging resistive random-access memory (ReRAM) and PIM techniques to accelerate genome sequence alignment. We propose RAPIDx, an algorithm and hardware co-design that implements an optimized banded alignment algorithm with reduced complexity on ReRAM-based PIM hardware. RAPIDx fully leverages ReRAM's internal bandwidth and parallelism to achieve superior throughput and energy efficiency.

Various algorithmic optimizations have been developed for alignment software [8, 11, 39, 44]. However, the limited computing resources of CPU severely restrict the achievable

performance. These works fail to generate satisfactory processing throughput and energy efficiency. To this end, many efforts have been made to design acceleration solutions on ASIC [9, 10, 32], GPU [7, 45], or FPGA [46] platforms. Through optimizing algorithm and hardware architecture, these accelerators have shown significant improvements in terms of efficiency and processing speed. However, the memory-intensive nature of genome alignment algorithms makes them suffer from the limited on-chip as well as expensive data movement between off-chip memory and processing cores, incurring energy overhead caused by data movement.

Existing PIM-based accelerators for genome analysis [3–5, 47–49] take PIM's advantages of high data parallelism and low-cost data movement, showing orders of magnitude efficiency and performance improvements over CPU and GPU. The previous PIM architecture for sequence alignment, RAPID [3], which computes genome alignment in memory, has the following deficits. First, the original DP algorithm [30] used by RAPID is sub-optimal since it is unable to measure the affine gap penalty, which has been widely used in software libraries [7, 8] and shown optimal alignment quality [50]. Second, RAPID does not consider software-hardware co-optimization, thereby wasting a large amount of energy and computing resources on redundant computations. Recent works [39, 51] demonstrate DP alignment algorithm exhibits great redundancy, and most of computation can be skipped using banded alignment [52] to accelerate the alignment process at the cost of negligible accuracy degradation.

In this chapter, we propose a software-hardware co-design, RAPIDx, that exploits digital PIM techniques on ReRAM to enable a highly parallel and more energy-efficient acceleration for sequence alignment. The key contributions can be summarized as follows:

- **PIM-friendly dynamic programming (DP) alignment**: We consider the affine gap penalty to construct more accurate scoring functions. Then we propose the adaptive banded parallelized DP alignment that is friendly for PIM implementation. The proposed alignment algorithm reduces the required arithmetic precision from 32-bit to only 5-bit

and obtains higher data parallelism. Meanwhile, the adaptive wavefront direction and bandwidth schemes significantly reduce memory footprint and computational complexity by over 10× at the cost of < 0.15% accuracy loss.

- **High-performance PIM architecture**: We propose efficient PIM architecture for RAPIDx, which achieves four-level data parallelism. RAPIDx leverages in-situ PIM operations [53] to perform low-energy and row-parallel in-memory alignment. Our peripheral circuits implement fast traceback as well as complex functions not friendly for PIM. Compared to previous RAPID [3], RAPIDx shows 5.5× latency reduction and 6.2× energy improvements.

- **System optimization and reconfigurable design**: We design novel PIM computing operations that are reconfigurable to support multiple types of alignment scoring as well as edit distance computation. This makes RAPIDx a multi-purpose accelerator that is flexible to support alignment and edit distance computations. We also analyze several possible limiting factors when integrating RAPIDx into existing computing system, including ReRAM cell's limited endurance, switching speed, and system considerations.

- **Improvements and accelerations**: We compare RAPIDx with state-of-the-art CPU baselines (Minimap2 [8] and Edlib [11]), GPU baseline (GASAL2 [7]), and ASIC baselines (ABSW [10] and GenASM [9]) on various workloads. For short-read alignment, RAPIDx delivers an average 131.1× and 46.8× higher throughput compared to Minimap2 [8] and GASAL2 [7], respectively. For long-read alignment, 1.8× to 2.9× throughput improvements are observed over ABSW [10] and GenASM [9]. For edit distance calculation, RAPIDx obtains up to 321× speedup over Edlib [11].

2.2 Related Work

2.2.1 Software for Sequence Alignment

Several software libraries [7, 8, 11, 39] have been developed for boosted genome analysis. The main point is optimizing the SW algorithm and CPU/GPU datapath to deliver accurate and

fast sequence alignment. BWA-MEM [39] is software to map genome sequences against large reference genomes. BWA-MEM aligns the given sequences using Burrows-Wheeler Transform (BWT) [54]. However, the memory footprint of aligning long genome is large and the irregular memory access of BWT limits the processing speed. Edlib [11] is a C++ library that exploits Myers's bit-vector algorithm [55] to parallelize the SW-based alignment. To realize more accurate and efficient alignment, Minimap2 [8] introduces two promising optimization strategies, banded alignment [52] and difference-based SW [56], which can be fitted into the datapath of single instruction, multiple data (SIMD). Minimap2 generates over 10× speedup over BWA-MEM. Even though these software libraries achieve fine-grain optimization, the limited computing resources on CPU fail to provide opportunities for further acceleration. Some researchers shift the focus to GPU-based acceleration. CUDAlign 4.0 [45] increases the parallelism by splitting each SW alignment into multiple GPUs and reducing the data dependency of the traceback process. GASAL2 [7] optimizes the data organization and develops efficient kernels for multiple sequence alignment workloads. These libraries exploit the abundant computing resources on GPU. But the resulted efficiency is not high because optimizations for SW algorithms are lacked due to the architectural limitations of GPU. RAPIDx is a software and hardware co-design that realizes algorithm and hardware optimizations at the same time.

2.2.2 Hardware Acceleration for Sequence Alignment

ASIC Accelerator: Various hardware accelerators [5, 9, 10, 32, 47–49, 53] have be presented to obtain higher energy efficiency and speedup for genome analysis. For ASIC designs, one challenge is how to realize long-read alignment under the constraints of limited on-chip memory. Darwin [32] proposes near-optimal tiling methods to align arbitrary sequence lengths, only requiring constant memory space. ABSW [10] leverages the tiling schemes [32] and implements an adaptively banded alignment on ASIC, achieving significant efficiency improvement. GenASM [9] proposes an approximate string matching algorithm and a systolic-array-based accelerator to increase data parallelism while reducing memory footprint. Although

prior works employ a variety of optimizations, the limited on-chip memory is still the bottleneck when aligning long sequences.

PIM Accelerator: PIM is a promising solution to increase data parallelism and energy efficiency via computing data in situ [4–6, 42]. The PIM-based alignment designs proposed in PRINS [47] and BioSEAL [48] accelerate algorithms using resistive content addressable memory (CAM). But the sequential associative search incurs a large amount of write operation and internal data movement, degrading efficiency, lifetime, and storage efficiency. Another set of works accelerates short read alignment, where long sequences are broken down into smaller sequences and heuristic methods are applied. AlignS [4], AligneR [5] and PIM-Aligner [6] exploit FM-index algorithm and PIM to realize short-read alignment. However, FM-index incurs irregular memory access, and is hard to exploit the data parallelism of PIM. RAPID [3] is a ReRAM-based PIM accelerator to implement in-situ alignment computation in the memory, which drastically reduces the data movement. However, the adopted algorithm in RAPID is sub-optimal and requires quadratic complexity, limiting its capability of aligning long sequences. In this chapter, we present several optimizations for alignment algorithms and hardware architecture to fully leverage the highly parallel PIM while providing satisfactory alignment quality. Our design, RAPIDx, delivers up to 9.3× alignment efficiency improvement compared to other PIM baselines.

2.3 Background

2.3.1 Genome Sequence Analysis

Overall Pipeline

A typical pipeline of modern genome sequencing analysis [8, 39, 44] involves indexing, seeding, filtering, and read alignment steps as shown in Figure 2.2-(a). For the indexing phase, the entire reference sequence is stored into special data structures, like BWT [54] and FM-indexing. The indexing is for quickly obtaining the location of query sequence in the reference sequence. Then, the seeding process uses the indexing information to query the potential mapping locations

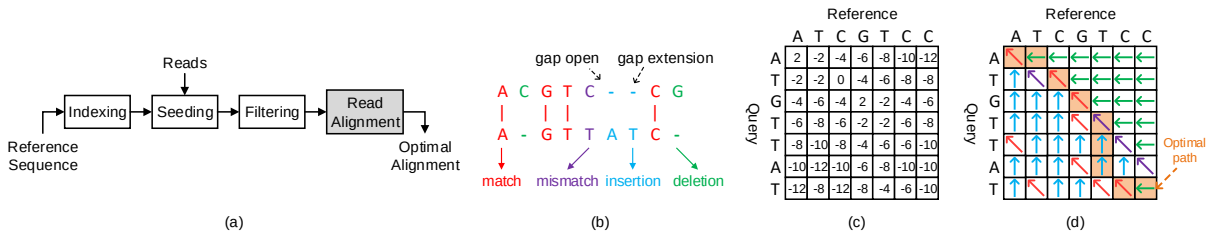


Figure 2.2: (a) The pipeline of genome sequence analysis. (b) Alignment example of sequences ACGTCCG and AGTTATC with affine gap penalties, (c) Score matrix, (d) Traceback matrix.

of genome reads. The filtering step discards invalid candidates or combines nearby candidates from the seeding step. Finally, the genome reads are aligned against the reference sequence around the candidate location using the SW algorithm. Among these steps, the most time-consuming step is read alignment used to determine how the read sequence can be optimally mapped to the reference sequence.

Sequence Alignment with Affine Gap Penalty

The sequence alignment can be described as finding the maximum alignment score between the reference sequence $R = r_1, r_2, ..., r_m$ and the query sequence $Q = q_1, q_2, ..., q_n$. Natural evolution and mutation as well as experimental errors during sequencing poses two types of changes in sequences - substitutions and indels. A substitution changes a base of the sequence with another, leading to a mismatch whereas an indel either inserts or deletes a base. Figure 2.2-(b) shows the comparison of two sequences, R = ACGTCCG and Q = AGTTATC. The left part rigidly compares the ith base of Q with R, where match and mismatch are considered. The right part assumes a different alignment that involves insertion and deletion. Note that the notation of dashes ($-$) is conceptual, and are used to illustrate a potential scenario that one sequence has been (or can be) evolved to the other.

Most sequence alignments are categorized into global or local alignment. The global and local alignments can be optimally addressed by NW algorithm [30] and SW algorithm [31], respectively. NW and SW both build up and compute the optimal alignment sequence based on DP [57, 58]. DP-based methods involve forming alignment matrices, which are used to compute

scores of various alignments based on a pre-defined scoring function. The scoring function is essential for accurate alignment since it is used to update the scoring matrix in DP. The previous work [59] mostly uses the scoring function with linear gap penalty, where the penalty is increasing linearly with the gap length. However, the linear gap penalty is insufficient to accurately evaluate the alignment scores for those sequences with the same total gap length. The gap-less sequence is normally more biologically meaningful compared to the sequence with more gaps. In this chapter, we adopt the scoring function with affine gap penalties [60] that consider the number and length of gaps. Figure 2.2 shows an example of alignment between sequence R = ACGTCCG and Q = AGTTATC using affine gap penalties. The updating rules for scoring matrices in DP with affine gap penalty can be expressed as:

$$E_{i,j} = \max \begin{cases} H_{i-1,j} - o \\ E_{i-1,j} - e \end{cases} \quad F_{i,j} = \max \begin{cases} H_{i,j-1} - o \\ F_{i,j-1} - e \end{cases} \tag{2.1}$$

$$H_{i,j} = \max\{E_{i,j}, F_{i,j}, H_{i-1,j-1} - s(r_j, q_i)\}$$

where E and F denote the alignment matrices that store the scores of insertion and deletion, respectively. H is the alignment score matrix that stores the total scores. $s(r_j, q_i)$ denotes the score of match A or mismatch B by comparing r_j and q_i. The gap opening penalty is o while e denotes the gap extension penalty. Figure 2.2-(c) shows an example of score matrix H calculated using Eq (2.1) with penalties $A = 2, B = 4, o = 4, e = 2$. A traceback phase is required to construct the optimal alignment path after the computation for all alignment matrices. The traceback matrix in Figure 2.2-(d) stores the path information. For global alignment, the traceback starts from the cell at the bottom-right corner while local alignment starts from the cell with the maximum score.

2.3.2 Difference-based Dynamic Programming (DP) Alignment

The updating function in Eq. (2.1) has the following limitations. The maximum value in the alignment matrix scales up linearly with the matrix dimension. The data bit width needs

to be increased as the sequence length increases to avoid computation overflow. Previous accelerations [10, 32] use a fixed bit width in the worst case, resulting in low computation efficiency. To resolve this issue, the original DP updating is rewritten into a computation-efficient form, named the difference-based formulation [56]. The basic idea is to store and compute the value difference of adjacent elements instead of the full-precision value in the alignment matrix, thus reducing the required arithmetic precision. As shown in the left side of Eq. (2.2), four matrices ΔH, ΔV, ΔE, and ΔF are used to store the difference values. After substituting the four difference matrices into Eq. (2.1), the alignment matrices (H, E, and F) are converted into the following difference-based formulation:

$$
\begin{cases}
\Delta H_{i,j} = H_{i,j} - H_{i-1,j} \\
\Delta V_{i,j} = H_{i,j} - H_{i,j-1} \\
\Delta E_{i,j} = E_{i+1,j} - H_{i,j} \\
\Delta F_{i,j} = F_{i,j+1} - H_{i,j}
\end{cases}
\Rightarrow
\begin{cases}
A_{i,j} = \max \begin{cases} s(i,j), \\ \Delta E_{i-1,j} + \Delta V_{i-1,j}, \\ \Delta F_{i,j-1} + \Delta H_{i,j-1} \end{cases} \\
\Delta H_{i,j} = A_{i,j} - \Delta V_{i-1,j} \\
\Delta V_{i,j} = A_{i,j} - \Delta H_{i,j-1} \\
\Delta E_{i,j} = \max\{-o, \Delta E_{i-1,j} - \Delta H_{i,j}\} - e \\
\Delta F_{i,j} = \max\{-o, \Delta F_{i,j-1} - \Delta V_{i,j}\} - e
\end{cases}
\tag{2.2}
$$

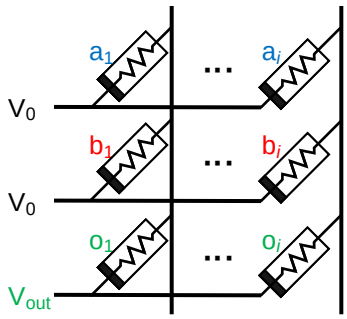
where an intermediate variable $A_{i,j}$ is added to the computation. It should be noted that Eq. (2.2) only changes the expression of original DP in Eq. (2.1) while retaining the identical information. Eq. (2.2) can generate the identical alignment results as Eq. (2.1).

There are two benefits of the difference-based alignment in Eq. (2.2). First, the arithmetic precision requirement is significantly reduced. According to [8, 56], the data range of $\Delta H_{i,j}$ and $\Delta V_{i,j}$ are bounded by $[-o-e, -e]$ while $\Delta E_{i,j}$ and $\Delta F_{i,j}$ are bounded by $[-o-e, M+o+e]$, where M denotes the maximum value of $s(i,j)$. Compared to the full-precision alignment, the difference-based representations only needs $\lceil \log_2(M+2o+2e+1) \rceil$-bit integer to calculate the alignment. Second, the required data precision is only determined by the used affine gap scores while independent with the sequence length. This property allows us to use a unified data bit width for different sequence lengths. For example, we use 5-bit integer for computing alignment and 3-bit integer for calculating edit distance as introduced in Section 2.5.4.

2.3.3 Digital Processing In-Memory (PIM)

Various types of memory devices are used for PIM to resolve the "memory wall" problem, including MRAM [4, 6], PCM, and SRAM [61]. MRAM suffers from severe read disturbance when the memory density increases [62]. ReRAM has higher memory density than MRAM and SRAM because the ReRAM cell is much smaller than MRAM and SRAM. Moreover, ReRAM has lower leakage power compared to other devices, making it an energy-efficient candidate for PIM. FeFET [63] and NAND flash [64] are the other two potential PIM candidates that are still in early development phase while ReRAM has been physically verified at scale [65]. ReRAM has higher error rates, but this is not a significant issue for alignment as alignment algorithms are already statistical in nature, and can tolerate significant errors at bit level. Considering all these benefits, we choose ReRAM-based PIM.

Traditionally, PIM with memristors is based on reading currents through different cells. However, some recent work has demonstrated ways, both in literature [53, 66, 67] and by fabricating chips [68], to implement logic using memristor switching. Digital PIM exploits variable switching of memristors. The output device switches whenever the voltage across it exceeds a threshold [69]. This property can be exploited to implement a variety of logic functions inside memory [53, 66]. Figure 2.3 shows an example of implementing NOR operation



(a)

a_i	b_i	o_{init}	o_i
0	0	1	1
0	1	1	0
1	0	1	0
1	1	1	0

(b)

Figure 2.3: Implementing NOR operation using ReRAM-based digital processing in-memory (PIM).

using ReRAM-based PIM [53]. A voltage V_0 is in parallel applied to the rows that contain the operand cells a_i and b_i. The output cell o_i switches to low voltage status (logical '0') from initial logical '1' whenever one or more inputs are '1's, resulting in logical `NOR` operation. Since `NOR` is a functionally complete logic gate, it can be used to implement other logic operations like addition [66] and multiplication [70]. For example, 1-bit addition (inputs being A, B, C) can be represented in the form of `NOR` as:

$$\begin{aligned} C_{out} &= ((A+B)' + (B+C)' + (C+A)')' \\ S &= (((A'+B'+C')' + ((A+B+C)' + C_{out})')')' \end{aligned} \tag{2.3}$$

where C_{out} and S are the generated carry and sum bits of addition. $(A+B+C)'$, $(A+B)'$, and A' represent $NOR(A,B,C)$, $NOR(A,B)$, and $NOR(A,A)$, respectively.

In-memory operations are in general slower than the corresponding CMOS-based implementations because memristor devices switch slowly. However, PIM architectures can provide significant speedup when it is exposed massive parallelism. Meanwhile, the long processing latency is amortized due to the high parallelism. RAPIDx utilizes two types of PIM operations (XOR and addition) introduced in FELIX [53] to perform in-memory alignment computation. This is because FELIX's PIM primitives achieve the same or significantly better latency, memory consumption, and efficiency than other digital PIM schemes [66, 71]. The other

digital PIM scheme [72] for floating-point arithmetic is not suitable for the fixed-point arithmetic in RAPIDx.

Specifically, the XOR and 1-bit addition are realized through:

- **XOR:** XOR ($\oplus$) can be expressed by OR (+), AND (.), and NAND ($(.)'$) as $A \oplus B = (A+B).(A.B)'$. We first calculate OR and then use its output cell to implement NAND. This operation is executed in parallel over all the columns of two rows. This logic just requires 2 cycles and one additional memristor device, which acts as the output cell.

- **Addition:** Let A, B, and C_{in} be 1-bit inputs of addition, and S and C_{out} the generated sum and carry bits respectively. Then, S is implemented as two serial in-memory XOR operations (A$\oplus$B)$\oplus$C. C_{out}, on the other hand, can be executed by inverting the output of the Min function proposed in [53]. Addition takes a total of 6 cycles and similar to XOR, we parallelize it over all columns in two rows.

2.4 Efficient Alignment in RAPIDx

In this section, we first analyze the challenges of realizing efficient in-memory alignment using digital PIM. Then we propose the adaptive banded parallelized DP alignment to balance performance and accuracy loss.

2.4.1 Challenges of Alignment using PIM

Data Bit Width and Latency

Compared to CMOS-based circuits, the slow switching speed of ReRAM cells incurs long latency when implementing PIM operations in Section 2.3.3. For example, 1-bit PIM addition takes 6 to 12 clock cycles [53]. As discussed in Section 2.3.2, the data bit width and range grow linearly with the sequence length. The previous accelerators [3, 48] adopt the original DP algorithm which uses 32-bit integers to guarantee lossless alignment. However, 32-bit integer is over-designed and incurs long processing latency when aligning short sequences (<1kbp)

since the lower 12-bit width is enough to provide sufficient data dynamic range [10]. Therefore, developing an alignment algorithm using low bit-width data is beneficial to reduce PIM latency. The difference-based DP alignment in Section 2.3.2 is a potential solution to alleviate this as it needs fixed data width independent of sequence length.

Data Parallelism

ReRAM-based PIM architectures [3, 6, 48, 49] offer substantial opportunities of extending the data parallelism. High parallelism amortizes the incurred long latency of PIM operations. One example is the row-parallel PIM operation [48, 53], where the bit-serial computation can be performed in the entire memory row simultaneously. How to exploit the architectural parallelism of ReRAM is key to attaining high alignment throughput. The other challenge from the algorithm is how to expose enough parallelism to ReRAM. For DP alignment, adjacent cells in alignment matrices exhibit data dependency. Previous works [8, 10, 32, 48, 56] utilize the wavefront parallelism based on the fact that cells over anti-diagonal have no data dependency. Unfortunately, this parallelism is far enough for PIM architecture.

Complexity and Accuracy

Figure 2.4-(a) illustrates the full DP alignment using Eq. (2.1), where all cells in the matrices with shape $m \times n$ need to be computed (m and n denote the lengths of reference and query sequences, respectively). The complexity is prohibitive when aligning long sequences. Banded alignment [51, 52] is an effective method to reduce the complexity from quadratic to near-linear. It should be noted that the banded approach is an approximate algorithm that may introduce accuracy degradation. One simple solution is to use a fixed and wide bandwidth ($B = 128$) as [10]. But this degrades the throughput and performance gain since wider bandwidth leads to higher complexity. The challenge is how to select narrow bandwidth for various lengths while ensuring the optimality of results.

2.4.2 Adaptive Banded Parallelized DP Alignment

We propose the adaptive banded parallelized DP alignment to resolve the above-mentioned challenges. The difference-based alignment in Eq. (2.2) relaxes the requirement of data precision and reduces the bit width for DP alignment. However, the computation of $\Delta H_{i,j}, \Delta V_{i,j}, \Delta E_{i,j}$, and $\Delta F_{i,j}$ can only be accomplished in a serial manner. Specifically, $A_{i,j}$ needs to be first computed before updating $\Delta H_{i,j}$ and $\Delta V_{i,j}$. Then the values of $\Delta V_{i,j}$ and $\Delta E_{i,j}$ require the newly updated $\Delta H_{i,j}$ and $\Delta V_{i,j}$. Consequently, parallelizing the computation for each updating step is difficult due to the inherent data dependency. We resolve this issue through further transforming Eq. (2.2) into a parallelized version similar to [56]. The variables in Eq. (2.2) are rewritten as the top part of Eq. (2.4), where auxiliary o and e values are added to each variable in Eq. (2.2). After substituting it into Eq. (2.2), we have the parallelized difference-based alignment as follows:

$$
\begin{cases}
A'_{i,j} = A_{i,j} + 2o + 2e \\
\Delta H'_{i,j} = \Delta H_{i,j} + o + e \\
\Delta V'_{i,j} = \Delta V_{i,j} + o + e \\
\Delta E'_{i,j} = \Delta E_{i-1,j} + \Delta V_{i-1,j} + 2o + 2e \\
\Delta F'_{i,j} = \Delta F_{i,j-1} + \Delta H_{i,j-1} + 2o + 2e
\end{cases}
\\
\Rightarrow
\begin{cases}
A'_{i,j} = \max\{s'(i,j), \Delta E'_{i-1,j}, \Delta F'_{i,j-1}\} \\
\Delta H'_{i,j} = A'_{i,j} - \Delta V'_{i-1,j} \\
\Delta V'_{i,j} = A'_{i,j} - \Delta H'_{i,j-1} \\
\Delta E'_{i,j} = \max\{A'_{i,j}, \Delta E'_{i-1,j} + o\} - \Delta H'_{i,j-1} \\
\Delta F'_{i,j} = \max\{A'_{i,j}, \Delta F'_{i,j-1} + o\} - \Delta V'_{i-1,j}
\end{cases}
\tag{2.4}
$$

where $\Delta H'_{i,j}$ and $\Delta V'_{i,j}$ only depend on new $A'_{i,j}$ and previous $\Delta V'_{i-1,j}$ and $\Delta H'_{i,j-1}$, respectively. Likewise, $\Delta E'_{i,j}$ and $\Delta F'_{i,j}$ can be calculated by the old $\Delta H'_{i,j-1}$ and $\Delta V'_{i-1,j}$ from the previous

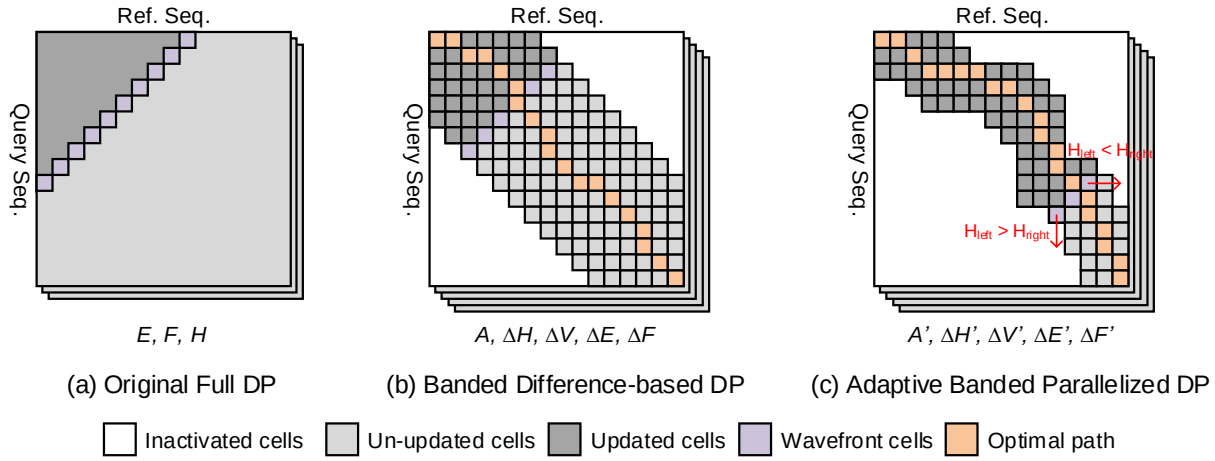


Figure 2.4: Illustration of three variants of DP alignment algorithms. Bandwidth $B = 6$ in (b) and $B = 3$ in (c).

iteration. In this case, the relaxed data dependency between four alignment matrices provides higher computation parallelism. After obtaining $A'_{i,j}$, the computation of $\Delta H'_{i,j}$, $\Delta V'_{i,j}$, $\Delta E'_{i,j}$, and $\Delta F'_{i,j}$ can be conducted in parallel to shorten the processing latency. We call this the alignment matrix level parallelism. The data range of four alignment matrices is shifted to $[0, M+2o+2e]$ from $[-o-e, M+o+e]$, requiring the same bit width as Eq. (2.2).

The banded alignment [52] significantly reduces the complexity based on the observation that the optimal alignment path normally locates not far away from the diagonal of alignment matrices. The reduction is achieved by limiting the cells in alignment matrices that need to be computed. Figure 2.4-(b) shows the banded DP alignment that only computes the cells located within a bandwidth $B = 6$ of the diagonal, whereas the rest cells are inactivated. In this way, only B wavefront cells (the cells that are updated simultaneously) are computed and moved over the main diagonal in each iteration. Bandwidth and wavefront direction are the two key factors that determine the accuracy and efficiency of banded alignment. The adaptive banded parallelized alignment adopted by RAPIDx is adaptive in the sense of bandwidth and wavefront direction as follows:

Table 2.1: Comparison of DP alignment algorithms in Figure 2.4

Algorithm	Complexity		Critical Path	Accuracy
	Computation	Memory		
Full DP	$O(mn)$	$O(mn)$	5×32 bit	High
Banded Difference-based DP	$O(mB)$	$O(mB)$	8×5 bit	Low
Adaptive Banded Parallelized DP	$O(mB)$	$O(mB)$	4×5 bit	High

Adaptive Bandwidth

A narrow bandwidth $B \ll m,n$ helps to perform a low-complexity alignment as the banded DP has $O(mB)$ complexity. To balance the algorithm efficiency and accuracy, the bandwidth B used in RAPIDx is adaptive based on the processed sequence length. The other factor to consider when choosing the bandwidth is the inflexibility of ReRAM-based PIM. The proper bandwidth needs to be determined before alignment computation. To this end, we express the relationship between bandwidth B and sequence length L as $B = \min(w + 0.01 \times L, 100)$, where w denotes the base bandwidth that determines the narrowest bandwidth while B is set to the multiple of w. The function limits the maximum bandwidth to 100 because previous BWA-MEM [39] shows $B = 100$ is enough to guarantee optimal alignment for all sequence lengths. On the other hand, a band with less than 20 is enough for over 99% cases as demonstrated in [51] but a too narrow band may not guarantee the optimality of alignment for long reads. This is because current long-read techniques (see Table 2.2) incur much more errors and the narrow band can not fully cover the optimal path. Thus, we empirically select the 0.01 coefficient to adaptively determine the minimum bandwidth that provides negligible degradation according to L. Based on the length of the given sequences, the bandwidth B can be pre-determined before alignment. We provide detailed experiments in Section 2.6.2 to guide the selection of the 0.01 coefficient and the best w that only introduce negligible accuracy loss.

Adaptive Wavefront Direction

The wavefront cells in Figure 2.4-(b) and (c) can move either rightward or downward in each iteration. The alignment tools, like Minimap2 [8] and BWA-MEM [39], mostly use a

pre-defined direction in Figure 2.4-(b), such that the wavefront moves towards the main diagonal. When we use narrow bandwidth ($B = 3$) in Figure 2.4-(c), simply computing the wavefront over the main diagonal may not obtain the optimal results because the fixed wavefront direction lacks flexibility and is unable to cover the optimal path. To this end, we use a simple adaptive wavefront direction scheme to dynamically adjust the moving direction of wavefront cells as in Figure 2.4-(c). The direction is decided based on the comparison result of two edge cells in the band of score matrix. Specifically, if the value of the rightmost cell is greater than the leftmost cell, this suggests the optimal path is more likely to go rightward [73]. Hence, the current wavefront is moved rightward. Otherwise, the wavefront is moved downward. The adaptive wavefront direction scheme only needs one comparison each iteration but effectively improves the accuracy of long-read alignment according to our test results in Table 2.5.

We conduct an algorithmic analysis for the aforementioned DP algorithms and compare their complexity, data parallelism, and critical path in Table 2.1. The critical path is defined as the longest data path needed to accomplish one iteration of cell updating. Thanks to the alignment matrix parallelism, the proposed adaptive banded parallelized alignment only needs half of the critical path of Eq. (2.2). More importantly, the adaptive wavefront direction compensates for the accuracy loss caused by narrow bandwidth, allowing the proposed algorithm to generate near-optimal results using near-linear complexity.

2.5 In-Memory Architecture of RAPIDx

We propose the PIM-based ReRAM accelerator, RAPIDx to implement the adaptive banded parallelized DP alignment in Section 2.4. RAPIDx utilizes the in-site PIM-based alignment algorithm and the efficient data flow with four-level parallelism to boost alignment process.

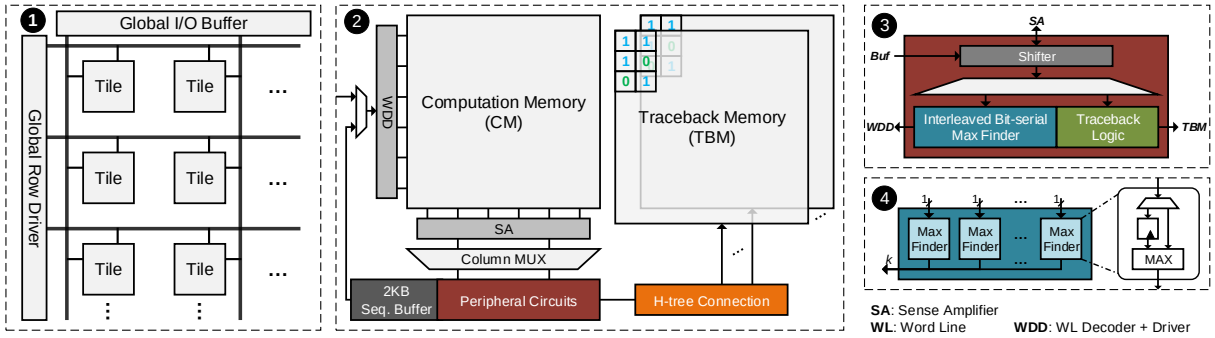


Figure 2.5: RAPIDx architecture. ❶ ReRAM memory organization of RAPIDx. ❷ Internal architecture of RAPIDx tile. ❸ Peripheral circuits (shifter, interleaved bit-serial max finder, and traceback logic). ❹ Interleaved bit-serial max finder.

2.5.1 Overview

As shown in ❶ of Figure 2.5, RAPIDx is a ReRAM-based PIM accelerator for genome sequence alignment. The algorithm in Section 2.4.2 exhibits various data parallelisms, including wavefront and alignment matrix levels. RAPIDx is organized in a multi-level hierarchy to extend the data parallelism. RAPIDx consists of 64 tiles, each RAPIDx tile independently receiving and transferring genome data through global I/O buffer and global row driver. The read genome sequences are stored in the sequence buffer within each tile. To minimize the data movement, the forward DP cells updating and traceback computation happen locally in each tile. There is no communication between tiles. We conduct design space exploration in Section 2.6.3 to choose the hardware configurations resulting in the best efficiency.

Figure 2.5-❷ shows the internal structure of RAPIDx tile, where one computation memory (CMs) and multiple traceback memories (TBMs) are implemented. One CM is connected to 15 TBMs through the H-tree connection, allowing low-latency and high-bandwidth data transfer between CMs and TBMs. The number of TBM is more than CM because most of the memory is used for storing traceback information. Each CM fetches the reference and query sequences from the 2KB sequence buffer. Then CM calculates A', $\Delta H'$, $\Delta V'$, $\Delta E'$, and $\Delta F'$ matrices in Eq. (2.4) using PIM-based XOR and addition operations combined with peripheral circuits. Each CM is able to access TBMs and transfer traceback data through the H-tree routing. Although the

ReRAM subarray exhibits high data parallelism, some computations of alignment and traceback can not be efficiently realized in CM. For example, finding the point-wise maximum values of two vectors in [3] is complex, requiring both leading one detector and bit-wise logical operations. PIM operations [53] is unable to support low-latency traceback in Eq. (2.5) as well as the adaptive wavefront direction scheme. In RAPIDx, we connect peripheral circuits to sense amplifier (SA) and offload these operations to the peripheral circuits, consisting of the shifter, interleaved bit-serial max finder, and traceback logic as shown in Figure 2.5-❸ and ❹.

In the peripheral circuits, we identify the max finder accounts for the largest area and has the most complex structure. The design of max finder faces several challenges. First, the additional overhead should be as low as possible to ensure not significantly sacrificing ReRAM memory density. Second, the max finder should match the processing rate of CM while minimally impacting the overall throughput. The max finding scheme in [3] incurs long latency. We further reduce the latency by offloading the max finding to the interleaved bit-serial max finder in Figure 2.5-❹. The interleaved bit-serial max finder is composed of k bit-serial max finders and the width k equals to the SA's bit width. This is to match the data rate of SA. The classic bit-serial max finder receives 2-bit input in parallel. However, only 1-bit data of multiple data points in the same vector can be read from CM through SA due to CM's bit-serial data organization. Hence, we add a latch and MUX before the input of bit-serial MAX finder to make it support the comparison of bit-serial data.

2.5.2 Data Flow with Four-level Data Parallelism

To fully exploit the acceleration opportunities and increase throughput, RAPIDx achieves four-level parallelism, namely tile level, sequence level, wavefront level, and alignment matrix level, as illustrated in Figure 2.6. On the host side, query reads are seeded and filtered in a batched processing manner. Then the resulted kt batches of reference and query pairs are sent to RAPIDx, where k denotes the number of memory segments in Figure 2.6-(b) and t denotes the number of tiles. The kt batches of reference and query data are evenly distributed to each

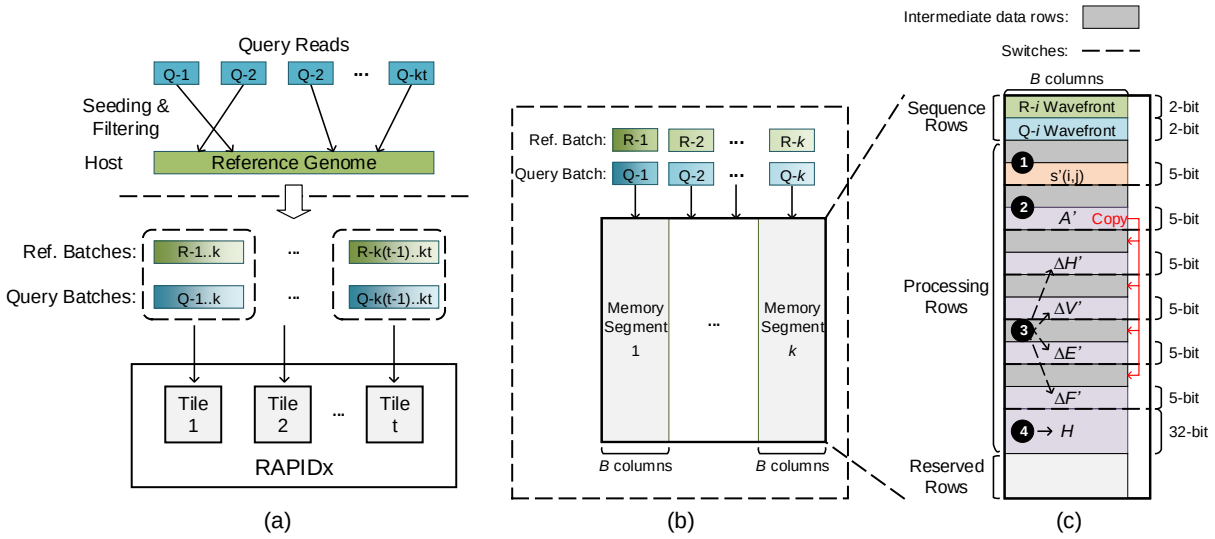


Figure 2.6: Four-level data parallelism and in-memory alignment in RAPIDx: (a) Tile-level parallelism. (b) Batched alignment in CM using sequence-level parallelism, (c) PIM-based in-situ banded parallelized alignment in each memory segment of CM.

tile. The tile-level parallelism enables different RAPIDx tiles to process and align k independent sequences in parallel, allowing the performance of RAPIDx to scale almost linearly with the number of implemented tiles. The CM subarray with size 1024×1024 used in this chapter introduces long latency due to the slow PIM operations [53]. The genome sequences in each CM are processed in batch to amortize the long latency of PIM. As illustrated in Figure 2.6-(b), each CM processes a reference and a query batch with batch size k. The CM is horizontally divided into k memory segments to compute the k pairs of reference and query sequences in parallel. The column width of each memory segment equals the bandwidth B of banded alignment. Hence, there are at most $\lfloor \frac{1024}{B} \rfloor$ memory segments.

RAPIDx achieves wavefront-level and alignment matrix-level parallelism in the memory segments of CM. The wavefront-level parallelism is based on the fact that the cells over anti-diagonal have no data dependency since they only depend on the cells in the previous diagonal. The row-parallel operations of ReRAM subarray compute and update the B wavefront cells over the anti-diagonal in Figure 2.4-(c) simultaneously. Meanwhile, the relaxed data dependency of parallelized alignment in Eq. (2.4) provides the alignment matrix-level parallelism, where $\Delta H'_{i,j}$,

$\Delta V'_{i,j}$, $\Delta E'_{i,j}$, and $\Delta F'_{i,j}$ can be computed in parallel.

2.5.3 In-memory Alignment

Forward DP Updating

As shown in Figure 2.6-(c), the data in ReRAM subarray are organized in the bit-serial manner, where each b-bit data lies vertically in b consecutive rows over the bit line. The rows of each memory segment are vertically divided into two regions, including sequence rows and processing rows. The sequence rows are used for storing genome bases of reference and query. Before starting the wavefront cells updating, the genome bases related to B wavefront cells are fetched from the sequence buffer and written to the sequence rows. Since each genome base, $\mathsf{A}, \mathsf{G}, \mathsf{C}, \mathsf{T}$, is encoded with 2-bit data, the sequence rows occupy 4 memory rows. The rest of memory rows work as processing rows and reserved rows, which are responsible for updating wavefront cells of $A'_{i,j}, \Delta H'_{i,j}, \Delta V'_{i,j}, \Delta E'_{i,j}, \Delta F'_{i,j}$ in Eq. (2.4) using PIM operations [53]. The processing rows are partitioned into five partitions by switches and $A'_{i,j}, \Delta H'_{i,j}, \Delta V'_{i,j}, \Delta E'_{i,j}, \Delta F'_{i,j}$ are stored and processed in each partition. Intermediate data rows are inserted into the processing rows to store constant values and intermediate results during computation. The constants used for comparison and subtraction when updating the DP alignment include $2o+2e$ and o. These pre-defined values are replicated and pre-stored in the reserved rows. PIM operations can directly access these values whenever needed. Specifically, the forward DP updating is computed in the following orders:

1. First, the 5-bit data $s'(i,j)$ are computed by comparing reference and query wavefront sequences (see ❶ of Figure 2.6-(c)). $s'(i,j)$ requires one comparison and addition to generate the match or mismatch score. The comparison between genome bases is done using 2-bit XOR operations.

2. Second, $A'_{i,j}$ is obtained from the maximum value of $s'(i,j)$, $\Delta E'_{i-1,j}$, and $\Delta F_{i,j-1}$ as shown in ❷ of Figure 2.6-(c). Two max operations are needed in this step.

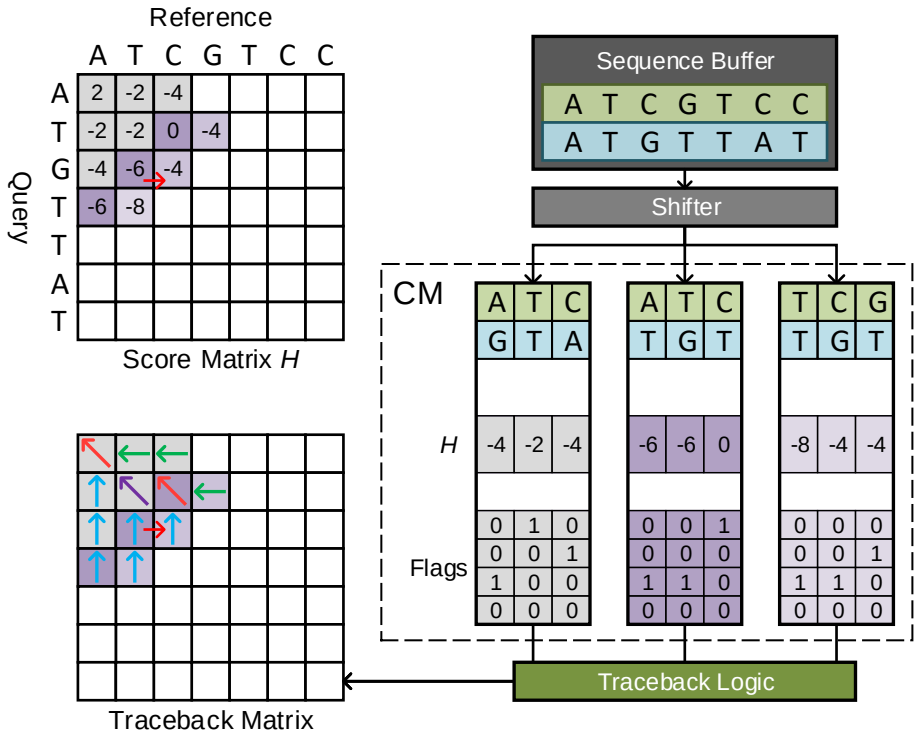


Figure 2.7: Illustration of adaptive wavefront direction and traceback process using peripheral circuits.

3. Third, four copies of $A'_{i,j}$ are written to the intermediate data rows with respect to $\Delta H'_{i,j}, \Delta V'_{i,j}, \Delta E'_{i,j}$, and $\Delta F'_{i,j}$ as ❸ of Figure 2.6-(c).

4. Third, $\Delta H'_{i,j}$ and $\Delta V'_{i,j}$ are updated in parallel using copied $A_{i,j}$ and previous $\Delta V'_{i-1,j}, \Delta H'_{i,j-1}$. Meanwhile, $\Delta E'_{i,j}$ and $\Delta F'_{i,j}$ are updated in parallel based on copied $A'_{i,j}$ and $\Delta E'_{i-1,j}$, $\Delta F'_{i,j-1}$, $\Delta H'_{i,j-1}$, $\Delta V'_{i-1,j}$ of the previous iteration. This step needs four subtractions, two additions, and two max operations.

5. Finally, the alignment score matrix $H_{i,j}$ need to be retrieved using the function $H_{i,j} = H_{i-1,j} + \Delta H_{i,j} = \Delta H'_{i,j} - (o+e) + H_{i-1,j}$, which requires one 5-bit subtraction and one 32-bit addition.

Adaptive Wavefront Direction

After wavefront cells are computed, the band will move either downwards or rightwards by one cell. Figure 2.7 illustrates how the wavefront with bandwidth $B = 3$ is moved using peripheral circuits, where the wavefront direction is controlled by the shifter and sequence buffer. The max finder first compares the leftmost and rightmost cells in score matrix H, determining the next direction for wavefront. Then, the shifter receives the direction signal and reads the corresponding genome sequence from the sequence buffer. If the wavefront is moving rightwards, the shifter fetches reference data. Otherwise, it fetches query data. After shifting to the position of wavefront cells, the new genome sequence is written to the sequence rows within CM. In this way, the majority of computation data stay stationary in CM using in-situ PIM-based alignment, reducing the data movement overhead.

Traceback Process

Each iteration of DP alignment is followed by updating traceback matrix. Eq. (2.1) can easily compute the traceback matrix through comparing the corresponding values of three alignment matrices I, D, and H. However, the difference-based DP alignment in Eq. (2.2) and Eq. (2.4) only store the difference values and do not explicitly give the score matrix H. Therefore, we modify the formula of generating traceback information of the original DP to calculate the traceback matrix TB as the following equation:

$$TB_{i-1,j-1} = \begin{cases} 00, \text{if } s'_{i,j} == (A+o+e) \text{ or } (-B+o+e) \\ 01, \text{if } \Delta H'_{i,j} == \Delta E'_{i-1,j} - \Delta V'_{i-1,j} \\ 10, \text{if } \Delta H'_{i,j} == \Delta F'_{i,j-1} - \Delta H'_{i,j-1} \\ 11, \text{if others} \end{cases} \tag{2.5}$$

where two subtractions and four comparisons are needed. 00, 01, and 10 denote the cases of match or mismatch, deletion, and insertion, respectively.

As shown in Figure 2.7, to efficiently implement Eq. (2.5) in memory, the traceback logic in ❹ of Figure 2.5 reads out the 4-bit flags that indicate the traceback information from CM in a bit-serial order. Then the traceback logic converts the 4-bit flags into 2-bit traceback data and stores them into TBM. Since there will be only one "1" in the 4-bit flags. The conversion from 4-bit flags to 2-bit data is accomplished by implementing one hot encoders within the traceback logic.

2.5.4 Reconfigurable Design with Dynamic Precision

The sequence alignment and edit distance calculation follow the same data flow of forward cell updating. The difference between alignment and edit distance calculation is the used scoring function. The scoring function of edit distance computation normally requires lower data bit width than alignment workloads. RAPIDx is reconfigurable to support these two workloads by adopting two types of PIM precisions. Moreover, we leverage the precision difference to further improve the performance of edit distance calculation.

Alignment Computation

For different alignment tools and target genomes to be aligned, various scoring functions with affine gap penalties may be applied. For example, BWA-MEM [39] uses a matching score $A = 1$, mismatch penalty $B = 4$, gap open penalty $o = 6$, and gap extension penalty $e = 1$. The other popular alignment tool, Minimap2 [8], uses a default scoring function with $A = 2, B = 4, o = 4, e = 2$. According to Section 2.4, the minimum data width should satisfy $\lceil \log_2(M + 2o + 2e + 1) \rceil$. For most scoring functions with affine gap penalties, a 5-bit PIM precision is able to realize accurate alignment without overflow.

Edit Distance Calculation

Edit distance (or Levenshtein distance) is a metric to measure the minimum number of deletion, insertion, and substitution required to transform one string to the other one. Edit distance calculation can be regarded as a simplified version of sequence alignment,

where the matching score is 0 while mismatch/gap opening/gap extension penalties are all 1. $\lceil \log_2(M+2o+2e+1) \rceil$ = 3-bit data width provides sufficient precision for edit distance calculation. Therefore, RAPIDx decreases the arithmetic precision from 5-bit to 3-bit when computing edit distance. This is beneficial to further improve throughput and reduce energy dissipation.

RAPIDx realizes the switching between the mentioned two types of PIM precisions through issuing different sets of commands to CMs. The commands for 3-bit and 5-bit precisions differ in they activate and access different ReRAM rows in CM to realize different computing precisions. So the overhead of PIM precision switching is negligible.

2.6 Evaluation

2.6.1 Experimental Setup

Methodology: We use VTEAM [69] with $R_{OFF} = 300k$ and $R_{ON} = 10k$ to model ReRAM cell. The other parameters are same with [66] that align with the practical ReRAM device [74]. The energy consumption and latency of PIM operations in RAPIDx are measured based on 10,000 Monte Carlo simulations in SPICE. The operation voltage of PIM is V_0=1V, and the worst-case switching latency is 2ns. The hardware parameters of ReRAM subarray are obtained from NVSim [75]. Its peripheral circuits, including shifter, interleaved bit-serial max finder, and traceback logic, are implemented using *Verilog* and synthesized by *Synopsys Design Compiler* on 45nm process node [76]. The area and energy consumption of sequence buffer are estimated using CACTI [77]. RAPIDx's frequency is set to 500MHz, matching the switching time of ReRAM device. We also develop a in-house simulator to estimate the genome alignment performance and energy consumption.

RAPIDx Configurations: Total 64 tiles are implemented in RAPIDx and each RAPIDx tile has 2MB size, containing one CM and 15 TBMs. Each ReRAM subarray consists of 1024×1024 cells and the width of column MUX output is 128-bit. The parameter selection is

Table 2.2: Error rates of generated datasets

Type	Substitution	Insertion	Deletion	Total
PacBio	1.5%	9.0%	4.5%	15%
ONT_2D	16.5%	5.0%	8.5%	30%
Illumina	3%	1%	1%	5%

Table 2.3: Hardware specifications of CPU and GPU baselines

CPU	Intel Xeon E5-2680	**GPU**	**Geforce GTX 1080 Ti**
	12 cores / 24 threads / 2.5GHz		
Cache	L1/L2/L3: 32KB/256KB/30MB	**Frequency**	1582 MHz
Memory	256GB / DDR4-2133MHz	**Memory**	11GB GDDR5X
TDP	120W	**TDP**	250 W

discussed in Section 2.6.3. The arithmetic precision is set to 5-bit for sequence alignment and 3-bit for edit distance calculation, which avoids overflow and maximizes the performance.

Datasets: We test RAPIDx's performance on both short and long reads. The sequence length of short reads ranges from 100bp to 500bp while the long reads vary from 2kbp to 10kbp. We use the homologous chromosomes, GRCh38 [78], from the National Center for Biotechnology Information (NCBI). The chromosomes, including 1 to 22, X, and Y, are used and the unmapped contigs are removed. These chromosomes contain 3 billion bp in total. The available memory space in RAPIDx is not able to store the entire genome. We assume RAPIDx fetches the query and reference sequences from the host memory for alignment.

As in Table 2.2, we generate the long-read data (PacBio and ONT datasets) using the sequence read simulator PBSIM [79]. PacBio and ONT have 15% and 30% error rate, respectively. PBSIM's default error profile and continuous long read (CLR) mode are used. The short-read Illumina datasets are produced by Mason [80] with 5% error rate. Both RAPIDx and other baselines are tested using at least 100,000 reads for each length.

Baselines: We compare the alignment performance of RAPIDx with state-of-the-art CPU, GPU, PIM, and ASIC accelerators. The CPU baselines include two libraries developed using C++, Minimap2 [8] and Edlib [11]. Minimap2 utilizes banded DP algorithms with affine gap penalties and adopts SIMD and multithreading to maximize the performance. Edlib is a C++

Table 2.4: Specifications of ASIC baselines

Design	ABSW [10]	GenASM [10]
Specifications	40nm with 480MHz frequency	28nm with 1GHz frequency
	Area: 5.51mm^2, Power: 1.2W	Area: 10.69mm^2, Power: 3.2W

program that makes use of edit distance and Myers's bit-vector algorithm [55] to parallelize the alignment and distance computation. The GPU baseline, GASAL2 [7], is optimized for GPU and delivers high throughput on various alignment workloads. We compile and run the programs on a server with hardware specifications in Table 2.3. The other parameters are the same as the original papers [7, 8, 11] without explicit specifications. We compare RAPIDx with four PIM designs, including RAPID [3], AlignS [4], AligneR [5], and PIM-Aligner [6]. We also compare RAPIDx with two ASIC accelerators, ABSW [10] and GenASM [9]. ABSW adopts the adaptive banded DP alignment algorithm with affine gap penalties based on 12-bit integers. Instead of using the DP-based alignment algorithms, GenASM exploits a modified approximate string matching algorithm to increase the parallelism and reduce memory footprint. Table 2.4 summarizes the area, frequency, and power consumption of ABSW and GenASM.

2.6.2 Algorithm Validation

The bandwidth of adaptive banded DP alignment is key to the alignment accuracy and efficiency. The base bandwidth w in the bandwidth calculation function $B = \min(w + 0.01 \times L, 100)$ determines the resulted bandwidth for sequence length L. Large w guarantees high alignment accuracy but increases the required computation and memory complexity.

We perform Monte Carlo simulations to validate the accuracy of adaptive banded parallelized DP alignment using different parameters. The alignment results of original DP with affine gap penalty in Eq (2.1) are regarded as the ground truth. Both of the tested algorithm adopt the identical scoring function $A = 2, B = 4, o = 4, e = 2$ with Minimap2 [8]. We randomly sample 1,000,000 short and long sequence reads from the read simulator. Illumina and ONT_2D in Table 2.2 are adopted as the reading scheme for short reads and long reads, respectively.

Table 2.5: Alignment accuracy of banded DP algorithms

Read Type	Adaptive Wavefront	Base bandwidth w				
		10	20	30	40	50
Short Read (Illumina)	No	100.0%	100.0%	100.0%	100.0%	100.0%
	Yes	100.0%	100.0%	100.0%	100.0%	100.0%
Long Read (ONT_2D)	No	6.51%	39.69%	31.33%	61.44%	71.13%
	Yes	99.23%	99.64%	99.85%	99.85%	99.95%

Table 2.5 gives the alignment accuracy, where the base bandwidth w is ranging from 10 to 50 and the bandwidth is calculated by $B = \min(w + 0.01 \times L, 100)$. We also add another dimension that enables or disables the adaptive wavefront direction. The results show that the accuracy for short read is all 100% even without adaptive wavefront direction. This is because Illumina only incurs 5% error. For long reads, the algorithm without adaptive wavefront direction yields unsatisfactory accuracy. Increasing w to 50 only yields 71.13% accuracy. This is because ONT_2D has lower reading quality, making the optimal alignment path more likely to be away from the diagonal. The fixed wavefront direction is unable to track and cover the optimal path. After enabling adaptive wavefront direction, a base bandwidth w of 10 achieves 99.23% accuracy. It is observed that the optimal w varies for reading schemes and sequence lengths. To balance alignment efficiency and accuracy, we choose $w = 10$ for short reads and $w = 30$ for long reads, which incurs 0.15% accuracy degradation.

2.6.3 Design Space Exploration

ReRAM Subarray Size

The ReRAM subarray size determines the memory density. The parasitic wire resistance is a major factor limiting the ReRAM size [66]. To study the impact of non-ideal wire resistance, we use the same model in [66] and assume the unit wire resistance between row or column is $R_w = 10\Omega$. The upper bound and lower bound of three critical voltages (operation voltage V_0, isolation voltages V_{HS} and V_{VS}) under different ReRAM array size are depicted in Figure 2.8. It shows the used $V_0 = 1.0$V falls in the allowed value range when array size is 1024×1024. The effective

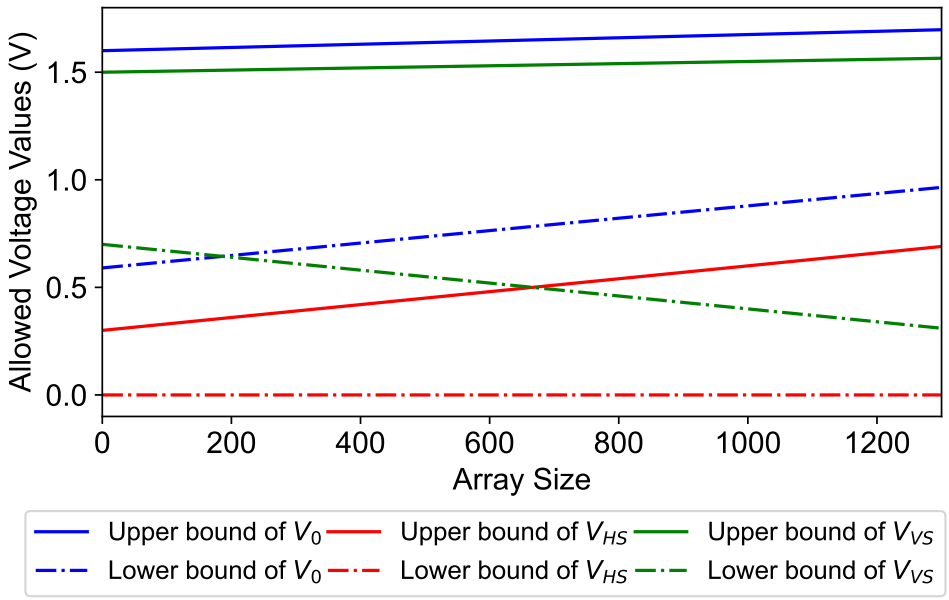


Figure 2.8: The lower bound and upper bound of voltages V_0, V_{HS}, V_{VS} under different ReRAM array sizes.

ranges for voltages V_{HS} and V_{VS} show we can set the isolation voltages to $V_{HS} = 0.2V, V_{VS} = 1.0V$ to satisfy the constraints for size 1024×1024. Given these results, the wire resistance does not affect the correct functionality of RAPIDx under ReRAM array size 1024×1024. This is because: 1. RAPIDx uses 2-input PIM operation to perform alignment, reducing the effects of wire resistance. 2. The 10kΩ R_{ON} is 10× larger than [66], making RAPIDx receive less impact from the wire resistance. Meanwhile, the chip-verified ReRAM [65] with 1024 dimension also demonstrates that the ReRAM subarray in RAPIDx is practical to manufacture.

Number of TBMs in Each Tile

The memory complexity of alignment is dominated by traceback data storage because the traceback data for a batch of sequences need to be stored until all DP alignment steps are finished. Therefore, each CM can access the memory space of t TBMs. The number of TBMs in each tile determines the supported maximum sequence length of RAPIDx. Each TBM is a 1024×1024 ReRAM subarray, thus each TBM can store $\frac{1024^2}{2}$ points of traceback data, where 2 denotes the 2-bit traceback information. Considering the sequence alignment or edit distance calculation has

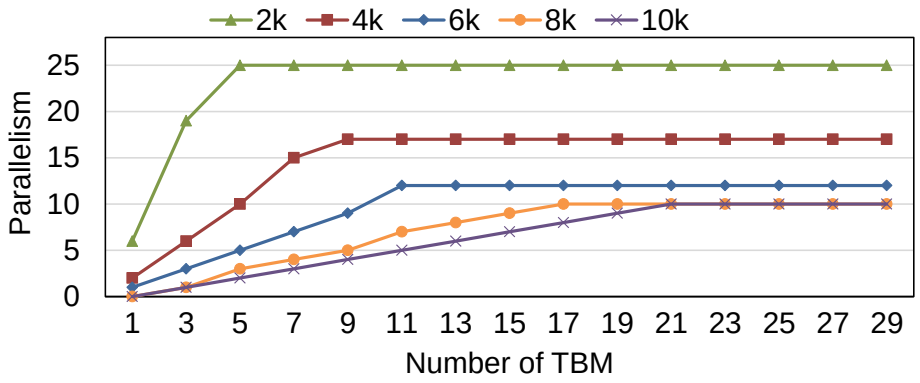


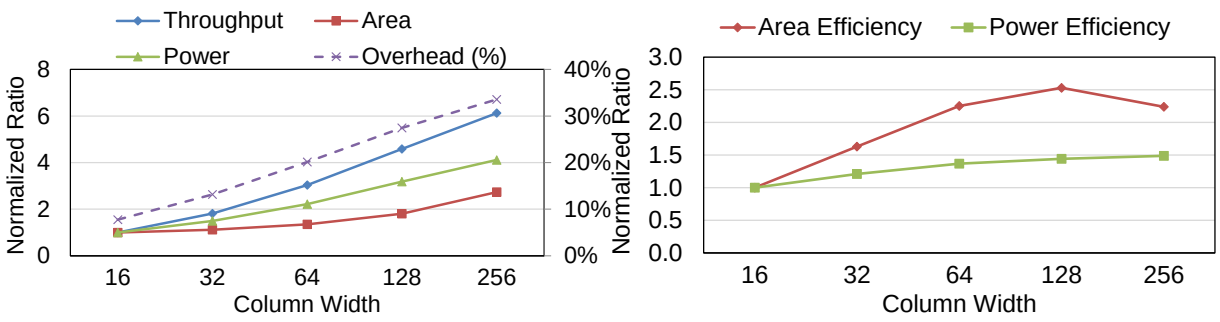
Figure 2.9: Relationship between maximum sequence-level parallelism and number of TBMs on long reads.

a bandwidth B and sequence length m, the number of TBMs t in each tile, satisfies $m \leq \frac{1024^2}{2B}t$. However, the memory requirement increases linearly by $k\times$ when each CM processes k sequences in parallel. In this case, the maximum sequence level parallelism (or the memory segment) becomes $k \leq \lfloor \frac{1024^2}{2m \cdot B}t \rfloor$. On the other hand, k will not exceed the maximum segment number in each ReRAM subarray $k \leq \lfloor \frac{1024}{B} \rfloor$. Therefore, the relationship between number of TBMs t, sequence-level parallelism k, and sequence length m is given by $k \leq \min(\lfloor \frac{1024}{B} \rfloor, \lfloor \frac{1024^2}{2m \cdot B}t \rfloor)$.

The sequence-level parallelism under various sequence lengths and TBM numbers is given in Figure 2.9. Shorter sequences require less TBMs to achieve the maximum parallelism. The $k_{\max}$ of sequences longer than 8kbp is limited by $\lfloor \frac{1024}{B} \rfloor$. As the maximum value of B is 100, $\lfloor \frac{1024}{B} \rfloor \leq 10$ for sequences over 8kbp. In this case, the number of TBMs t, making $\lfloor \frac{1024^2}{2m \cdot B}t \rfloor > 10$, can not further improve the performance. We implement $t = 15$ TBMs to ensure sufficient sequence-level parallelism for 10kbp while balancing area overhead. Thus, each RAPIDx tile consists of 16 ReRAM subarrays.

Column Width of Peripheral Circuits

The peripheral circuits of CM are connected to the column MUX of SA and have the same width as column MUX. The column width of peripheral circuits is a design parameter affecting the overall throughput, power, and area. Figure 2.10 shows the comparison of performance for different widths (from 16 to 256) of peripheral circuits. As shown in Figure 2.10-(a), wider



(a) Throughput, area, power, and overhead. **(b)** Area efficiency and power efficiency.

Figure 2.10: Performance comparison for different column widths of peripheral circuits.

column width leads to higher throughput and the increasing trend of throughput is slightly more significant than area and power when the width is between 16 and 128. The overhead here denotes the percentage of peripheral circuits area to single ReRAM subarray. We depict the area efficiency and power efficiency in Figure 2.10-(b) to understand the relationship between efficiency and column width. Area efficiency and power efficiency peak at width 128 and 256, respectively. However, wider width introduces larger area overhead to CM. We choose the column width of 128 to achieve good tradeoff between efficiency and overhead.

2.6.4 Area and Power Results

The area and power breakdown of RAPIDx is summarized in Table 2.6. The bit-serial max finder takes up 62.3% area and 61.6% power of the peripheral circuits, respectively. About 16% area of CM is consumed by peripheral circuits. Each RAPIDx tile is composed of 1 CM and 15 TBMs, consuming 0.637mm^2 area and 0.16W power. We measure the power dissipation of RAPIDx under sequence alignments for long sequence lengths (2kbp to 10kbp) with enabling the traceback procedure. As a result, the area and power of RAPIDx with 64 tiles in total are 40.8mm^2 and 10.3W, respectively.

2.6.5 Performance Evaluation

We measure the performance of RAPIDx on various sequence lengths and compare with state-of-the-art acceleration solutions for genome sequence analysis. The sequences are

Table 2.6: Area and power breakdown of RAPIDx

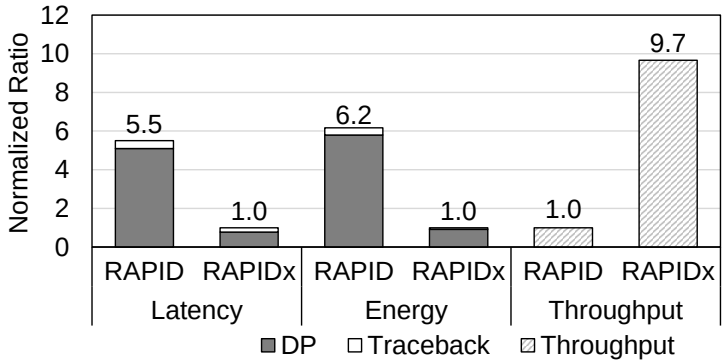
Peripheral Circuits	Area (um^2)	Power (mW)
Shifter	542.6	0.03
Max Finder	4, 520.8	2.05
Traceback Logic	1, 872,4	1.21
Others	325.2	0.03
Total	7, 260.9	3.32
Seq. Buffer	8, 492.6	1.5
ReRAM Subarray	38, 395.0	9.76

RAPIDx	Area	Power
Per tile	637,334.4um^2	0.16W
Total	40.8mm^2	10.3W

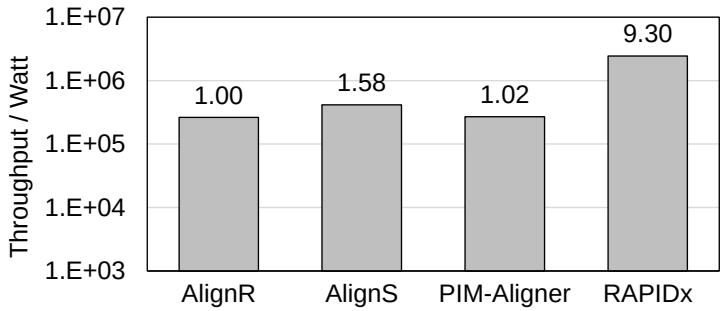
divided into short reads (<1kbp) and long reads (>1kbp). Two types of workloads are considered, including sequence alignment and edit distance calculations. RAPIDx uses 5-bit integer for alignment and 3-bit integer for edit distance calculation.

Comparison with PIM Designs

Our previous work, RAPID [3], is also a ReRAM-based PIM design for sequence alignment. First, we evaluate the reduction of processing latency and energy by adopting the parallelized DP alignment. The comparison of latency and energy with the original DP alignment for a single step of cells updating is shown in Figure 2.11-(a). RAPID uses the unoptimized DP alignment with 32-bit precision. The used PIM operations are the same as RAPIDx. As a result, the parallelized DP alignment based on difference presentation yields 5.5× latency reduction and 6.2× energy reduction over the original DP alignment. The latency and energy consumed by forward DP computation are reduced by 82% and 84% over the previous RAPID, respectively. The gain comes from the reduced arithmetic precision from 32-bit to 5-bit as well as the parallelized computation. On the other hand, the reduction of latency and energy for traceback is less significant. Although the parallelized DP alignment requires less bit width, its traceback is more complicated and involves more computations than the original DP algorithm. The longest sequence support by RAPIDx is 10kbp so we test the throughput of RAPID and RAPIDx on



(a) Latency, energy, and throughput comparison.



(b) Energy efficiency comparison (in log scale).

Figure 2.11: Performance comparison for PIM designs, including RAPIDx, RAPID [3], AlignS [4], AligneR [5], and PIM-Aligner [6].

this length in Figure 2.11-(a). RAPIDx yields 9.7× throughput improvement over RAPID due to the low complexity and high data parallelism provided by adaptive banded parallelized DP alignment.

In Figure 2.11-(b), we compare the energy efficiency with the other three PIM designs for short-read alignment, including AlignS [4], AligneR [5], and PIM-Aligner [6]. The read length is 100bp and the alignment efficiency is measured by the alignment throughput (reads per second) divided by the power dissipation. RAPIDx delivers 5.9× to 9.3× alignment efficiency compared to other PIM designs. It should be also noted that the area of mentioned PIM designs is: RAPIDx (40.8mm^2), AlignR (36.1mm^2), AlignS (62.5mm^2), and PIM-Aligner (59.3mm^2). This shows that RAPIDx achieves 8.4× to 13.3× throughput/W/mm^2 efficiency compared to other designs. This is because the optimized adaptive banded parallelized DP alignment in RAPIDx

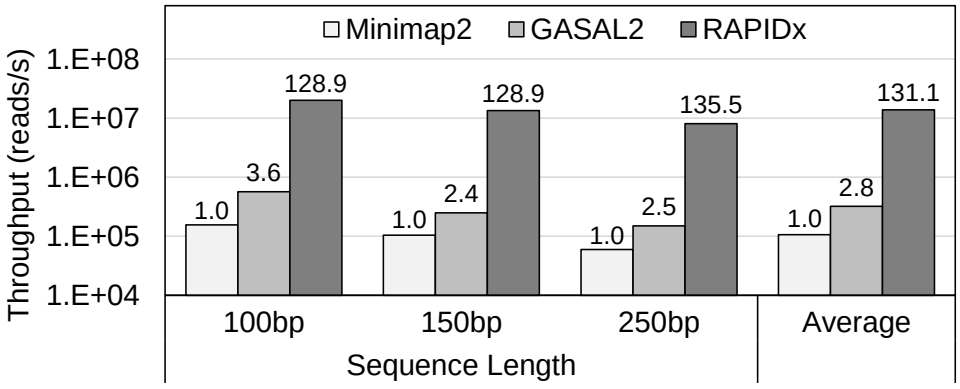


Figure 2.12: Alignment throughput comparison of RAPIDx, GASAL2 [7], and Minimap2 [8] for short reads.

significantly reduces computational complexity over the original full DP algorithm and allows to fully exploit the internal data parallelism of ReRAM. In comparison, AlignS, AligneR, and PIM-Aligner realize alignment based on FM-index algorithm, which requires multiple steps of computation and incurs data dependency [81]. AlignS, AligneR, and PIM-Aligner only support fixed read length while RAPIDx supports both short reads and long reads, making RAPIDx more scalable and reconfigurable.

Performance Comparison on Short-read Alignment

For alignment tasks on short reads, the length ranges from 100bp to 250bp and we use Minimap2 [8] as the CPU baseline and GASAL2 [7] as the GPU baseline. Figure 2.12 depicts the alignment throughput of RAPIDx, Minimap2, and GASAL2 for short reads in log scale. The alignment throughputs for three tested accelerators slightly decrease as the sequence length grows. RAPIDx on average delivers 131.1× and 46.8× throughput over Minimap2 and GASAL2, respectively. The processing latency of RAPIDx is longer than the other two counterparts due to the fact that a single PIM operation of RAPIDx requires longer latency than CPU and GPU. However, the row-parallel PIM operations provide higher computation parallelism. The proposed multi-level parallelism scheme ensures multiple reference and query sequences can be aligned in parallel, significantly increasing the data parallelism and PIM utilization. As a result, RAPIDx

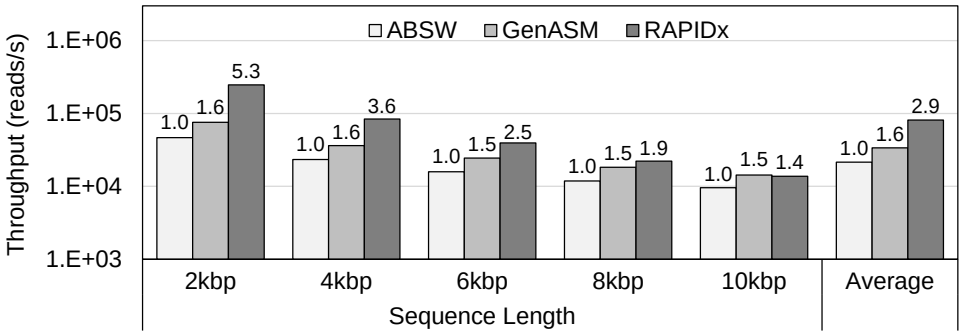


Figure 2.13: Alignment throughput comparison of GenASM [9], ABSW [10], and RAPIDx for long reads.

achieves an average throughput of 13.9M reads/s for short-read alignment.

DP alignment is computation-intensive and the bottleneck of CPU is the limited computing cores. Even though GPU has much more computing capabilities than CPU, we observe that GASAL2 only yields 2.4× to 3.6× speedup over Minimap2 because Minimap2 uses a banded DP algorithm and multi-threading to reduce the complexity, thus improving the overall throughput. In comparison, GASAL2 requires more computing resources since it does not finely optimize the original DP alignment. RAPIDx is an algorithm and hardware co-optimization that addresses the deficits of Minimap2 and GASAL2.

Performance Comparison on Long-read Alignment

For long reads from 2kbp to 10kbp, ABSW [10] and GenASM [9], are adopted as the two ASIC baselines. The throughput comparison with ASIC for long-read alignment is shown in Figure 2.13, where the performance of ASIC baselines is scaled to 45nm process for the fair comparison. RAPIDx achieves the highest throughput with an average speedup of 2.9× and 1.8× over ABSW and GenASM, respectively. Due to the limited on-chip memory space, both ABSW and GenASM are not able to store the entire traceback matrix for long reads. They rely on large off-chip memory to store the intermediate data. To realize alignment for long sequences, they use the overlapping scheme [32] to divide the long sequence into short chunks and the neighbor chunks are overlapped. ABSW and GenASM need to consecutively process the short chunks. As

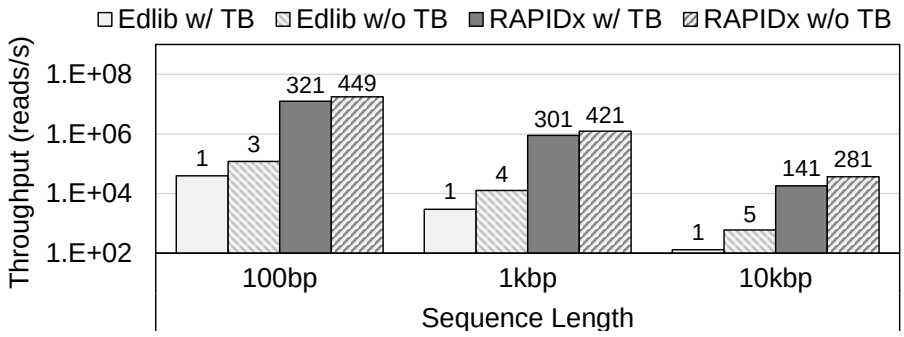


Figure 2.14: Throughput and latency comparison of RAPIDx and Edlib [11] for edit distance computation.

a result, the overlapping area incurs additional computational complexity, which degrades the performance.

ABSW and RAPIDx are based on banded DP algorithms. The difference is that RAPIDx adopts the optimized 5-bit parallelized DP alignment based on difference representations. ABSW uses 12-bit precision to ensure arithmetic precision for DP alignment. RAPIDx's lower bit width reduces both the complexity and the memory footprint of DP alignment compared to ABSW. The other limitation of ABSW is it can only process a fixed bandwidth of 128 since a total of 128 processing elements (PEs) are implemented and dedicated to updating the wavefront of banded alignment. This means ABSW is only able to align one sequence at a time. In contrast, RAPIDx accepts a batch of sequences and distributes them into different tiles to perform alignment in parallel.

Performance Comparison on Edit Distance Computation

To evaluate the performance of edit distance calculation, we compare RAPIDx with Edlib [11] on three lengths (100bp, 1kbp, and 10kbp). Figure 2.14 shows the throughput of RAPIDx and Edlib with or without traceback process. Knowing the edit distance of two sequences is enough for some scenarios, without the need for traceback process. So we test the cases with or without traceback. The throughput of RAPIDx with traceback is 141× to 321× over Edlib. After disabling the traceback, the speedup of RAPIDx is less significant. 56× to 149×

improvements of RAPIDx are observed compared to Edlib. Although Edlib adopts optimized Myers's bit-vector algorithm [55] with banded alignment to increase computation efficient, it is a single-thread program only able to access limited computing resources of CPU. Hence, the performance dramatically decreases after enabling traceback.

2.6.6 Discussions

Host-RAPIDx System Design

RAPIDx is a PIM-based domain-specific accelerator and works as the domain-specific co-processor for speeding up computation-intensive genome sequence alignments. We consider a system that transfers data between RAPIDx and the host. The sequencing and configuration data are sent from the host to RAPIDx. We estimate the memory bandwidth required by RAPIDx and the results show that required memory bandwidth decreases when sequence length grows. The required peak memory bandwidth is 1.41GB/s at 100bp. For the host side, the popular DDR4 Dual-Inline Memory Module (DIMM) that provides over 12.8GB/s data rate can easily satisfy the bandwidth requirement. The other consideration is the processing latency. As pointed out in Section 2.6.5, RAPIDx requires longer latency than CPU. Considering that genome sequence alignment is not a latency-sensitive task, the long latency will not become a major factor that limits system performance. Hence, RAPIDx can be integrated into existing computer machines with negligible hardware modifications.

Flexible Scoring Functions

The affine gap penalty of DP alignment will be changed according to different application scenarios. RAPIDx is able to flexibly support various scoring functions. When the gap open penalty o equals the gap extension penalty e, the affine gap penalty becomes a linear gap penalty scoring. If $e = 0$, RAPIDx implements a constant gap penalty where only opening a gap leads to a penalty, discouraging the number of gaps but tends to result in long gaps. Whereas, if $o \neq e$ and both of o and e are non-zero values, we have affine gap penalty, which is the widely used gap

penalty model for genome alignment. The affine gap penalty tries to align the given sequences with fewer and smaller gaps as compared to the constant gap penalty. No architectural and data flow modifications need to be made to RAPIDx if we want to switch between different scoring functions. The support for flexible scoring is realized by setting associated constant values into the intermediate data rows of CM before alignment.

ReRAM's Write Endurance

ReRAM cell has limited write endurance, so RAPIDx will fail after exceeding the endurance limit. As shown in Figure 2.6-(c), the wavefront alignment at each iteration needs to write the rows in the computing region once. Figure 2.4 shows the required number of iterations equals to the sum of reference and query sequences' lengths. We can apply wear leveling techniques to reduce the imbalance effect, thus extending the write endurance of ReRAM. The wear leveling is realized via moving the computing region over the row dimension. Specifically, this can be done through changing the writing address without additional overhead. Moreover, we observe some ReRAM devices [82] provide 10^{12} write endurance. In this case, RAPIDx can align over 10^{14} sequences with length 150bp. We notice that one of the most advanced next-generation sequencing (NGS) platforms from Illumina, *NextSeq 1000 & 2000*, generates a maximum 1.2 billion reads (each has a length of 150bp) in 11 to 48 hours [24]. Therefore, each RAPIDx is able to support the alignment task of each NGS sequencer for at least 100 years.

2.7 Conclusion

Chapter 2 presents a novel PIM accelerator, RAPIDx, for sequence alignment. We leverage the parallelized DP algorithm using difference representation to reduce the required data width from 32-bit to 5-bit integers. Based on this, we propose adaptive banded parallelized DP alignment to adaptively adjust the bandwidth and wavefront direction, reducing the quadratic complexity to near-linear complexity while only incurring 0.15% accuracy degradation. Then we present the PIM architecture on ReRAM that exploits four-level data parallelism to efficiently

implement the proposed algorithm. We develop peripheral circuits and row-parallel PIM data flow to support in-situ alignment with low latency. The evaluation results demonstrate that RAPIDx provides 131.1× and 46.8× better short-read alignment throughput compared to CPU and GPU baselines, respectively. For long-read alignment, RAPIDx delivers up to 2.9× and 9.3× throughput improvements compared to state-of-the-art ASIC and PIM accelerators.

Genome alignment demonstrates high accuracy across various genome analysis tasks. However, its substantial memory consumption makes it unsuitable for latency-sensitive scenarios or resource-constrained hardware. To address these challenges, the next chapter introduces genome sketching, a more lightweight and memory-efficient approach that enables rapid similarity estimation without the need for alignment.

This chapter contains material from "RAPIDx: High-performance ReRAM Processing in-Memory Accelerator for DNA Alignment", by Weihong Xu, Saransh Gupta, Niema Moshiri, and Tajana Rosing, which appears in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2023. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Memory-efficient Sketching for Genomics

3.1 Introduction

Calculating the Average Nucleotide Identity (ANI) similarity of genome files is the key step for various downstream workloads in genome analysis, such as large-scale database search [83], clustering [84], and taxonomy analysis [85]. The ReRAM-based PIM hardware design, RAPIDx, presented in the previous chapter, can be integrated into traditional BLAST-based methods [86, 87] that rely on base-level alignment to perform accurate ANI calculations. However, the alignment process is computationally expensive and requires hours to calculate ANIs. The slow speed of alignment-based approaches has become a major bottleneck for large-scale genome analysis.

Several state-of-the-art works have tried to speed up large-scale genome analysis by approximating the genome similarity using more efficient data structures. These works can be categorized into two types: mapping-based and sketch-based approaches as follows. FastANI [88] and Skani [89] are two representative mapping-based algorithms that leverage k-mer-based alignment for ANI estimation. FastANI is built upon the Mashmap sequence mapping algorithm [90] and achieves a significant speedup compared to the alignment-based baseline [86]. Skani uses the sparse chaining to increase the sensitivity of the mapping, further improving accuracy and efficiency of ANI estimation. However, both FastANI and Skani suffer from high memory consumption. For example, Skani needs to store indexing files with a storage size comparable

to the original dataset. FastANI encounters out-of-memory issues on large datasets as reported in [89].

This chapter presents a more lightweight method, called *genome sketching*, to address the aforementioned challenges because it significantly reduces storage size while providing satisfactory accuracy of estimation [85]. Unlike alignment-based or mapping-based tools [86–89] that require expensive computation or large memory space, sketch-based approaches [33,34,36,91] only preserve the most essential features of the genome (called the "sketch"). The sketch's compact representation enables rapid and efficient ANI approximation for genome files. Mash [33] and Sourmash [36] represent groundbreaking efforts to use MinHash [92] and FracMinHash [93,94] to estimate genomic similarity, respectively. Bindash [35] improves the accuracy of ANI estimation over Mash by adopting the one-permutation rolling MinHash with optimal densification [95]. Dashing 2 [34] utilizes the SetSketch data structure [96] and incorporates multiplicities to produce memory-efficient genome sketches and accurate estimation of ANI.

3.1.1 Motivation

By transforming raw genome data into more compact data structures, genome sketching represents a paradigm shift in bioinformatics, paving the way for more scalable and rapid genomic analyses in the era of big data. Recent studies on hyperdimensional computing (HDC) have demonstrated the effectiveness of using HDC to accelerate bioinformatics workloads, such as pattern matching [97–100] and spectral clustering [101].

Limitations of existing HDC/SimHash-related search algorithms.

Table 3.1 summarizes the key features of state-of-the-art tools that utilize HDC or SimHash algorithms. GenieHD [99], BioHD [97], and Demeter [100] are three representative HDC-based tools. Due to the limitation of N-gram binding-based encoding, existing HDC tools for genome search only supports short genomes sequences with length ≤ 200. However, they require very large sketch HV dimension (10k to 100k) to achieve good accuracy, which

Table 3.1: Comparison for related work for genome search and seed matching

Algorithm / Tool	**GenieHD** [99]	**BioHD** [97]	**Demeter** [100]	**BLEND** [102]	**HyperGen** (Proposed)
Encoding method	N-gram HDC binding	N-gram HDC binding	N-gram HDC binding	SimHash	**FracMinHash HDC**
Supported sequence length	≤200	≤200	≈150	150-20k	**Arbitrary**
Sketch dimension	100k	10k-40k	40k	30-50*	**2k-8k**
Support ANI estimation?	No	No	No	No	**Yes**
Supported Application	Containment search	Containment search	Containment search	Seed matching	**ANI-based search and clustering**

* Sketch dimension for each seed.

degradates the overall efficiency. The N-gram binding-based encoding shows high computational complexity. In comparison, HyperGen adopts a more efficiency encoding method that combines FracMinHash and HDC aggregation.

Meanwhile, existing HDC-based tools do not support ANI estimation and ANI-based search. They can only check the containment of given query. These drawbacks limit their downstream applications. The other related work is BLEND [102] that uses SimHash to encode genome seeds. The difference includes: 1. HyperGen and BLEND are for different tasks. BLEND is used for seed matching while HyperGen is for more general-purpose ANI estimation and database search, 2. Compared to HyperGen, BLEND uses much smaller sketch dimension for each seed.

Opportunities and Limitations of DotHash

Recent DotHash [103] shows superior space and computational efficiency for the Jaccard similarity estimation. DotHash leverages the HDC-based random indexing [104, 105] and is originally designed for fast set intersection estimation. The main difference between DotHash and MinHash lies in the format of generated sketch: MinHash represents a sketch as a hash set with discrete values, while DotHash represents a sketch with a nonbinary vector of high dimension. DotHash's vector representation of the sketch achieves faster processing speed since it can fully exploit the low-level hardware parallelism (such as CPU's Single Instruction Multiple Data (SIMD) and GPU) optimized for vector processing.

However, DotHash still suffers from two major limitations that hinder its application

to genome sketching. First, DotHash is only applicable to non-genome data since it lacks an effective k-mer sampling strategy to generate genomic sketches. Second, DotHash uses high-precision floating point numbers to represent random vectors, exhibiting large runtime overhead and slow speed. Our goal is using HDC [103, 106] to achieve better tradeoffs between ANI estimation accuracy, runtime performance, and memory efficiency over previous sketch-based tools [33, 34, 36].

3.1.2 Contributions

In this chapter, we propose HyperGen, a novel tool for efficient genome sketching and ANI estimation. HyperGen exploits the emerging HDC (similar to DotHash [103]) to boost genomic ANI calculation. Specifically, we optimize DotHash's efficiency by converting the sketch generation process into a low bit-width integer domain. This allows us to represent the genome sketch using the high-dimensional vector (HV) at the cost of negligible runtime overhead. Based on the HV sketch, we propose an approach to estimate the Jaccard similarity using vector matrix multiplication. We also introduce a lossless compression scheme using bit-packing to further reduce the sketch size.

We benchmark HyperGen against several state-of-the-art tools [33, 34, 86, 88]. For ANI estimation, HyperGen demonstrates comparable or lower ANI estimation errors compared to other baselines across different datasets. For generated sketch size, HyperGen achieves 1.8× to 2.7× sketch size reduction as compared to Mash [33] and Dashing 2 [34], respectively. HyperGen also enjoys the benefits of the modern hardware architecture optimized for vector processing. HyperGen shows about 1.7× sketch generation speedup over Mash and up to 4.3× search speedup over Dashing 2. To the best of our knowledge, HyperGen offers the optimal trade-off between speed, accuracy, and memory efficiency for ANI estimation.

3.2 Preliminaries

Fast computation of Average Nucleotide Identity (ANI) is pivotal in genomic data analysis (microbial genomics to delineate species), as ANI serves as a standardized and genome-wide measure of similarity that helps facilitate genomic data analysis. Popular approaches to calculate ANI include: alignment [86, 87], mapping [88, 89], and sketch [33, 34, 36, 91]. However, base-level alignment-based and k-mer-level mapping-based methods involve either time-consuming pairwise alignments or memory-intensive mappings. In the following sections, we focus on the sketch-based ANI estimation with significantly better efficiency.

3.2.1 MinHash and Jaccard Similarity

Existing sketh-based approaches [33, 34, 36, 91] do not directly compute ANI. Instead, they compute the Jaccard similarity [33], which is used to measure the similarity of two given k-mer sets. Then the Jaccard similarity is converted to ANI as shown in Eq. (3.8). The conversion between Jaccard similarity and ANI is computationally trivial, so most efforts in previous works [33, 34, 36, 91] are to find more efficient and accurate ways to estimate Jaccard similarity.

Without loss of generality, we denote k-mer as consecutive substrings with length k of the nucleotide alphabet, *e.g.* $\sum^k = \{A, G, C, T\}^k$. $\mathcal{S}_k(X)$ denotes the set of k-mers sampled from genome sequence X based on a given condition. HyperGen uses k-mer's hash to represent $\mathcal{S}_k(X)$ for better efficiency. Therefore, the Jaccard similarity for two sequences, A and B, can be computed as follows:

$$J_k(A, B) = \frac{|\mathcal{S}_k(A) \cap \mathcal{S}_k(B)|}{|\mathcal{S}_k(A) \cup \mathcal{S}_k(B)|}, \tag{3.1}$$

where $J_k(A, B) \in [0, 1]$ is the Jaccard similarity indicating the overlap between k-mer sets of two sequences. Note that HyperGen uses canonical k-mers by default.

A straightforward idea to sample k-mer sets in Eq. (3.1) is to keep all k-mers. However, this incurs prohibitive complexity since all unique k-mers need to be stored. The resulting

complexity is $O(L)$ for a sequence of length L. To alleviate the complexity issue, Mash [33] and its variants [90, 107] use MinHash [92] to approximate the Jaccard similarity by only preserving a tiny subset of k-mers. In particular, Mash keeps N k-mers that have the smallest hash values $h(\cdot)$. In this case, the Jaccard similarity is estimated as:

$$J(A,B) = \mathcal{P}(\min_{a\in A} h(a) = \min_{b\in B} h(b)). \tag{3.2}$$

Here, using MinHash helps to reduce the sketch complexity from $O(L)$ to a constant $O(N)$. The sampled k-mer set $\mathcal{S}_k(X)$ that stores N smallest k-mer hash values is regarded as the genome file sketch required for ANI estimation.

3.2.2 Jaccard Similarity using DotHash

A recent work [103] demonstrates that the speed and memory efficiency of Jaccard similarity approximation can be improved by using the DotHash based on Random Indexing [104]. The key step to compute Jaccard similarity in Eq. (3.1) is computing the cardinality of set intersection $|A \cap B|$ while the cardinality of set union can be calculated through $|A \cup B| = |A| + |B| - |A \cap B|$.

In DotHash, each element of the set is mapped to a unique D-dimensional vector in real number using the mapping function $\phi(x)$. Each set is expressed as an aggregation vector $\mathbf{a} \in \mathbb{R}^D$ such that

$$\mathbf{a} = \sum_{a\in A} \phi(a), \tag{3.3}$$

where the aggregation vector sums all the elements' vectors generated by the mapping function $\phi(x)$. One necessary constraint for function $\phi(x)$ is: the generated vectors should satisfy the quasi-orthogonal properties:

$$\phi(a)\cdot\phi(b) = \begin{cases} 0, \text{if } a \neq b, \\ 1, \text{if } a == b. \end{cases} \tag{3.4}$$

DotHash [103] uses a pseudo random number generator (RNG) as the mapping function $\phi(x)$ because the RNG can generate uniform and quasi-orthogonal vectors in an efficient manner.

Using the quasi-orthogonal properties, the cardinality approximation for set intersection is transformed into the dot product of two aggregation vectors:

$$\begin{aligned} |A \cap B| &= \mathbb{E}[\mathbf{a} \cdot \mathbf{b}] \\ &= \mathbb{E}\left[\sum_{a \in A}\sum_{b \in B} \phi(a) \cdot \phi(b)\right] \\ &= \sum_{a \in A}\sum_{b \in B} \mathbb{1}(a == b) \\ &= \sum_{x \in A \cap B} 1, \end{aligned} \tag{3.5}$$

where those vectors not in the set intersection ($a \neq b$) have no contribution to the inner product due to their quasi-orthogonality as in Eq. (3.4). DotHash effectively aggregates all elements in a set to form an aggregation vector with D dimension. The space and computational complexity of set cardinality estimation is $O(D)$. Moreover, the computation process of DotHash is highly vectorized and can be easily boosted by existing hardware architecture optimized for general matrix multiply (GEMM).

3.3 HyperGen: Memory-efficient Genome Sketching Tool

The aforementioned DotHash provides both good accuracy and runtime performance [103]. However, we observe two major limitations of DotHash: 1. Although DotHash can be used to calculate the cardinality of set intersection, it cannot be applied to genomic sketching because DotHash lacks a k-mer sampling module that identifies the useful k-mers; 2. The computation and space efficiency can be further optimized because the previous DotHash manages and processes all vectors in floating-point (FP) numbers. The mapping function $\phi(x)$ incurs significantly overhead.

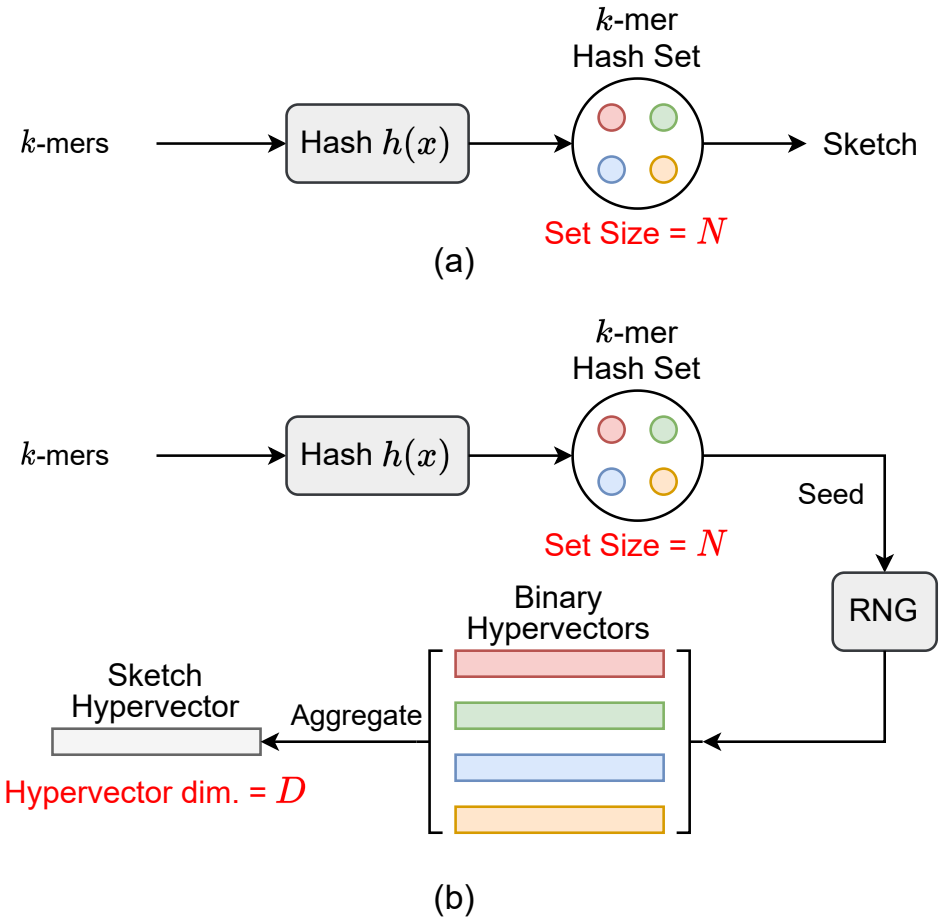


Figure 3.1: Algorithmic overview for (a) Mash-like sketching, and (b) HyperGen sketching for genome sequences. Mash stores the genome sketch in a k-mer hash set with $O(N)$ complexity while HyperGen aggregates N k-mer hashes into a D-dimensional sketch HV with $O(D)$ complexity.

We present HyperGen for genomic sketching applications that addresses the limitations of DotHash. Figure 3.1 shows the algorithmic overview for (a) Mash-like sketching and (b) HyperGen sketching schemes. The first step of HyperGen is similar to Mash, where both Mash and HyperGen extract k-mers by sliding a window through given genome sequences. The extracted k-mers are uniformly hashed into the corresponding numerical values by a hash function $h(x)$. To ensure low memory complexity, most k-mer hashes are filtered and only a small portion of them are preserved in the k-mer hash set to work as the sketch (or signature)

Algorithm 1: Generation of sketch hypervector in HyperGen

Input: Genome sequence X, Scaled factor S, Maximum hash value M, HV dimension D, Pseudo random number generator RNG

Output: Sketch HV $\mathbf{H}$ for sequence X

```
/* Sampling k-mers using FracMinHash                                  */
S_k ← {}
for k-mer x ∈ X do
    if h(x) < M/S then
        S_k ← h(x) ∪ S_k
/* Hyperdimensional encoding for k-mer hash                           */
H ← 0
for seed ∈ S_k do
    // Binary HV encoding for k-mer hash
    hv ← 0
    for i ← 1 to D/64 do
        rnd ← RNG(seed)
        seed ← rnd
        hv_{i*64...(i+1)*64} ← rnd
    // Binary HV aggregation
    for i ← 1 to D do
        H_i ← H_i + (hv^i × 2 − 1)
```

of the associative genome sequence. The key difference is that HyperGen adds a key step, called *Hyperdimensional Encoding for k-mer Hash*, to convert k-mer hash values into binary hypervectors (HVs) and aggregate to form the D-dimensional sketch HV. To distinguish itself from DotHash, the random vector in HyperGen is named HV. Algorithm 1 summarizes the flow of generating sketch hypervector in HyperGen. In the following sections, we explain the details of HyperGen.

3.3.1 Step 1: *k*-mer Hashing and Sampling

Mash uses MinHash that keeps the smallest N hash values as the genome sketch. In comparison, HyperGen adopts a different k-mer hashing and sampling scheme. Specifically, HyperGen performs a sparse k-mer sampling using FracMinHash [93, 94] (instead of MinHash in Mash). Given a hash function $h : \sum^k \mapsto [0, M]$ that maps k-mers into the corresponding

nonnegative integer, the sampled k-mer hash set is expressed as Line 2-4 in Algorithm 1:

$$\mathcal{S}_k(A) = \{h(x) \mid \forall x \in A : h(x) \leq \frac{M}{S}\}, \tag{3.6}$$

where M is the maximum hash value while S denotes the scaled factor that determines the density of sampled k-mers in the set. FracMinHash has been widely adopted in other tools, such as Sourmash [36] and Skani [89], due to its excellent performance. The advantage of using FracMinHash over MinHash [92] is that it ensures an unbiased estimation of the Jaccard similarity of k-mer sets with very dissimilar sizes [93], providing better approximation quality than MinHash and its variants [33, 90]. However, FracMinHash usually produces a larger hash set compared to Mash [93], requiring more memory space. Step 2 in HyperGen alleviates the increased memory issue.

3.3.2 Step 2: Hyperdimensional Encoding for k-mer Hash

In Figure 3.1-(a), after the k-mer hashing and sampling process, Mash-like sketching algorithms (such as Mash [33], Sourmash [36], and Mash Screen [108]) directly use the sampled k-mer hash set as the sketch to compute the Jaccard similarity for given sequences.

In Figure 3.1-(b), HyperGen adds an additional step, called *Hyperdimensional Encoding for k-mer Hash* (Line 5-13 in Algorithm 1), before the sketch is generated. This step essentially converts the discrete and numerical hashes in the k-mer hash set to a D-dimensional and nonbinary vector, called *sketch hypervector*. The hypervector dimension D is normally large (1024 to 8192) to ensure good accuracy. In particular, each hash value in the k-mer hash set is uniquely mapped to the associated binary HV hv as Line 6-11 of Algorithm 1. HyperGen relied on recursive random bit generation to produce binary HVs of arbitrary length: the k-mer hash value is set as the initial seed of the pseudo RNG$(seed) \mapsto rnd$ function. For each iterative step, a 64b random integer rnd is generated using $seed$. The generated integer rnd is not only assigned to the corresponding bits in hv, but is also set as the next $seed$.

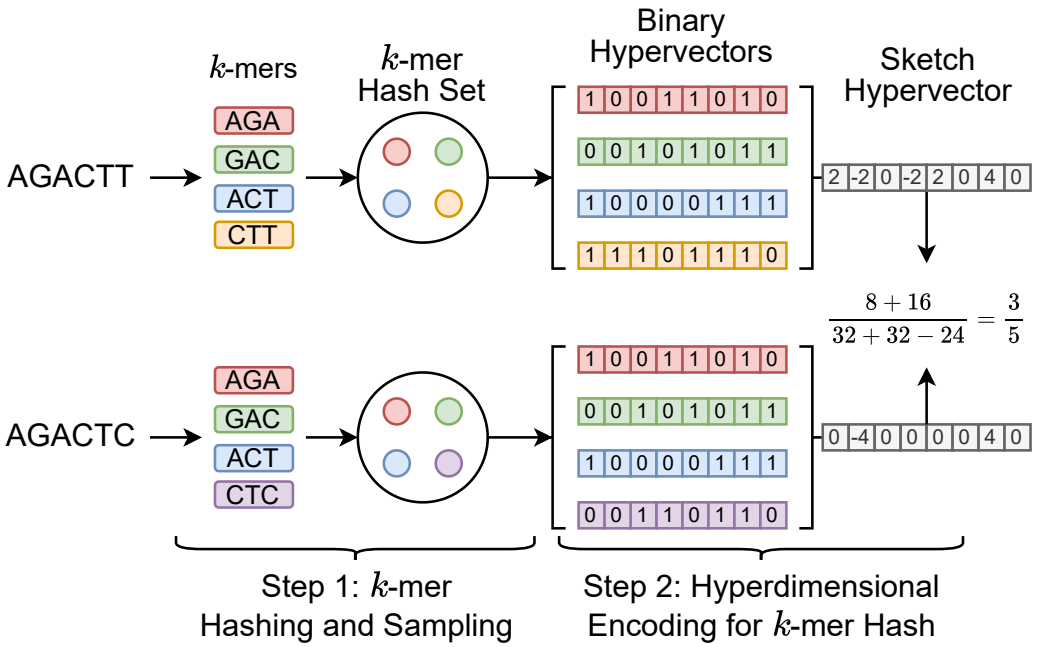


Figure 3.2: Sketch hypervector generation and set intersection computation in HyperGen. Each k-mer with size $k = 3$ first passes through a hash function $h(x)$. The k-mers ($A = AGACTT$ and $B = AGACTC$) are hashed to hash set. Then each k-mer hash value is converted into the associated orthogonal binary HV. The set intersection between two k-mer hash sets is computed using Eq. (3.11).

The hash function RNG$(\cdot)$ that maps the k-mer hash value to the binary HV hv is the key component of HyperGen because it determines the speed and quality of genome sketch generation. The following factors should be considered when selecting a good RNG$(\cdot)$ function: 1. The function needs to be fast enough to reduce the additional overhead for sketch generation. 2. The generated random binary HVs need to be able to provide enough randomness (*i.e.*, the binary HVs are as orthogonal as possible). This is because binary HVs are essentially random binary bit streams that need to be nearly orthogonal to each other to satisfy the quasi-orthogonal requirements. 3. The sketches results should be reproducible (*i.e.*, the identical bit streams can be generated using the same seed). We adopt a fast and high-quality pseudo RNG[1] in Rust language [109], which passes two randomness tests: TestU01 and Practrand [110]. In this case, we can use the pseudo RNG to stably generate high-quality and reproducible binary HVs.

[1]https://github.com/wangyi-fudan/wyhash

Figure 3.2 shows an example of generating the sketch HVs with dimension $D = 8$ for two genome sequences based on k-mer size $k = 3$ and k-mer hash set size $N = 4$. Each sampled k-mer hash value in the hash set is converted to the corresponding binary HV $hv \in \{0,1\}^D$ using the function $\mathrm{RNG}(x)$. Then, all N binary HVs are aggregated into a single sketch HV $\mathbf{H} \in \mathbb{Z}^D$ based on the following point-wise vector addition:

$$\mathbf{H} = \sum_{i=1}^{N} hv^i \times 2 - 1, \tag{3.7}$$

where the binary HV $hv \in \{0,1\}^D$ is first converted to $\{-1,+1\}^D$. hv^i denotes the i-th binary HV in the set. Then all binary HVs in the set are aggregated together to create the corresponding sketch HV. Compared to Mash-liked sketching approaches [33, 36, 93], HyperGen is more memory efficient because the sketch HV format is more compact with $O(D)$ space complexity, which is independent of the k-mer hash set size N. Meanwhile, HyperGen's hyperdimensional encoding step helps to achieve better ANI similarity estimation quality (see Section 3.4).

3.3.3 Step 3: ANI Estimation using Sketch Hypervector

The generated sketch hypervector can be used to efficiently estimate the ANI similarity. HyperGen estimates ANI value using the same approach in [33]. The ANI under the Poisson distribution is estimated as:

$$\mathrm{ANI}(A,B) = \left(1 + \frac{1}{k} \cdot \log \frac{2 \cdot J_k(A,B)}{1 + J_k(A,B)}\right) \times 100, \tag{3.8}$$

where $J_k(A,B)$ denotes the Jaccard similarity between genome sequence A and sequence B while k is the k-mer size.

Therefore, ANI estimation in HyperGen becomes calculating Jaccard similarity based on sketch HVs. Eq. (3.1) shows that the intersection size and the set size of two k-mer hash sets are the keys to calculating the Jaccard similarity. For $hv^i \in \{-1,+1\}^D$, the cardinality of a set $\mathcal{S}_k(A)$

is computed as follows:

$$|\mathcal{S}_k(A)| = \frac{\|\mathbf{H}_A\|_2^2}{D} = \frac{\sum_{i=1}^{N}\|hv^i\|_2^2}{D} = \frac{N \cdot D}{D} = N, \tag{3.9}$$

which shows the set cardinality can be computed based on the L^2 norm of sketch HV. The computation of set intersection in HyperGen is similar to DotHash [103]'s Eq. (3.5) because HVs in HyperGen share the same quasi-orthogonal properties as DotHash. Then, Eq. (3.5) becomes:

$$\begin{aligned}
|\mathcal{S}_k(A) \cap \mathcal{S}_k(B)| &= \frac{\mathbf{H}_A \cdot \mathbf{H}_B^T}{D} \\
&= \frac{\sum_i (hv^i \times 2 - 1) \cdot \sum_j (hv^j \times 2 - 1)^T}{D} \\
&= \frac{\sum_i \sum_j D \cdot \mathbb{1}(hv^i == hv^j)}{D} \\
&= \sum_i \sum_j \mathbb{1}(hv^i == hv^j) \\
&= \sum_{x \in \mathcal{S}_k(A) \cap \mathcal{S}_k(B)} 1.
\end{aligned} \tag{3.10}$$

With Eq. (3.9) and Eq. (3.10), HyperGen first estimates the following Jaccard similarity using the derived sketch HVs:

$$\begin{aligned}
J_k(A,B) &= \frac{|\mathcal{S}_k(A) \cap \mathcal{S}_k(B)|}{|\mathcal{S}_k(A)| + |\mathcal{S}_k(B)| - |\mathcal{S}_k(A) \cap \mathcal{S}_k(B)|} \\
&= \frac{\mathbf{H}_A \cdot \mathbf{H}_B^T}{\|\mathbf{H}_A\|_2^2 + \|\mathbf{H}_B\|_2^2 - \mathbf{H}_A \cdot \mathbf{H}_B^T}.
\end{aligned} \tag{3.11}$$

Then ANI in Eq. (3.8) can be easily calculated.

3.3.4 Software Implementation and Optimization

HyperGen is developed using the Rust language, and the code is available at https://github.com/wh-xu/Hyper-Gen. We present the following optimizations to improve the speed and efficiency of HyperGen.

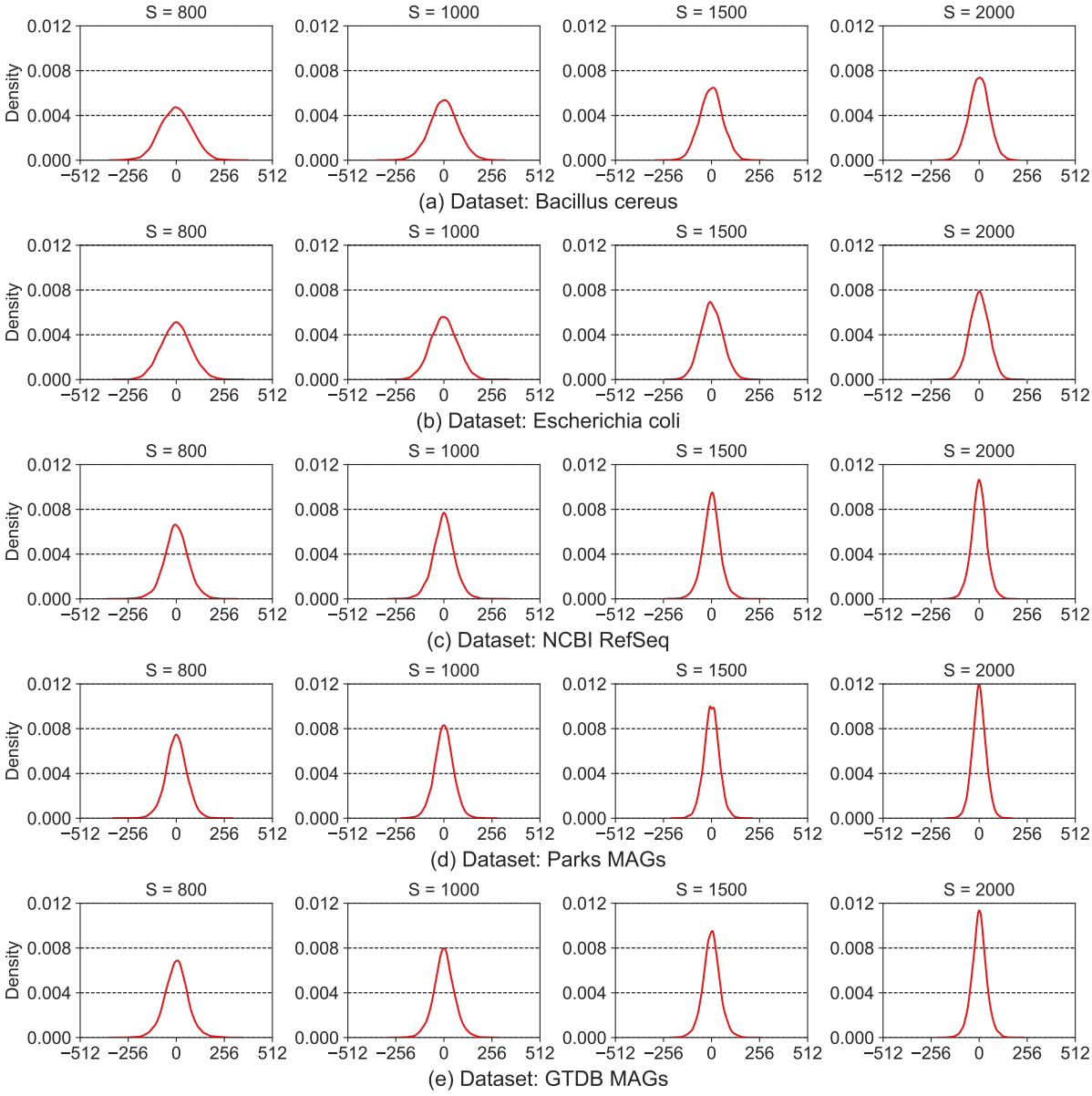


Figure 3.3: The value distribution of sketch hypervectors (HVs) generated by HyperGen when using various scaled factor $S = 800$ to 2000.

Sketch Quantization and Compression

Although the sketch HV has a compact data format with high memory efficiency, there still exists data redundancy in sketch HVs that can be utilized for further sketch compression. Our experimental observation is that the value range of sketch HVs is distributed within a bell curve as shown in Figure 3.3. HV values exhibit a bell curve distribution, where the majority of

values locate within the range -300 to 300. Sketch HVs can be effectively quantized to about 10 bits without precision loss. Rather than store the full-precision sketch hypervector (*e.g.*, INT32), we perform lossless compression by quantizing the HV to a lower bit width. The quantized bits are concatenated together using bit-packing.

Fast HV Aggregation using SIMD

The inner loop of binary HV aggregation step in Algorithm 1 incurs significant runtime overhead when a large HV dimension D is applied. We develop a parallelized HV aggregation using single instruction, multiple data (SIMD) instruction to reduce the impact of increased HV aggregation time.

Parallel Sketching

HyperGen provides two sketching modes: 1. *normal mode* and 2. *fast mode*. The *normal mode* sketches genome files on CPU with multithreading. The *fast mode* offloads genome sketching to GPU with better computing capabilities. The *fast mode* can be widely supported by commodity GPUs. Our measurement results in Figure 3.7 show that HyperGen's *fast mode* further improves the sketching speed by 1.8× to 2.7× over *normal mode*. The results produced by these two modes are identical.

Pre-computation for HV Sketch Norm

The L^2 norm of each sketch hypervector, $\|\mathbf{H}\|_2$, is precomputed during sketch generation phase. The L^2 norm value is stored along with the sketch hypervector to reduce redundant computations for the ANI calculation phase.

Table 3.2: Specifications for the evaluated genome datasets

Dataset Name	Description	Size	Query Genome	Source
Bacillus cereus	Draft genome assemblies of *Bacillus cereus s.l.* from the prokaryote section of the NCBI Genome database.	3.1GB	*Bacillus anthracis* (NZ_CM002395)	Dataset 2 at http://enve-omics.ce.gatech.edu/data/fastani
Escherichia coli	Draft genome assemblies of *Escherichia coli* from the prokaryote section of the NCBI Genome database.	22GB	*Escherichia coli* (GCA_000303255)	Dataset 3 at http://enve-omics.ce.gatech.edu/data/fastani
NCBI RefSeq	Prokaryotic genomes downloaded from RefSeq database.	5.6GB	*Escherichia coli* K12 W3110 (NC_007779)	Dataset 1 at http://enve-omics.ce.gatech.edu/data/fastani
Parks MAGs	A large collection of metagenome-assembled genomes.	20GB	Pseudomonas stutzeri (Parks GCA_002292085_1)	Dataset 5 at http://enve-omics.ce.gatech.edu/data/fastani
GTDB MAGs	A phylogenetically consistent and rank normalized genome-based taxonomy for prokaryotic genomes sourced from the NCBI Assembly database.	203GB	*Escherichia coli* K12 W3110 (NC_007779)	Release r207 at https://gtdb.ecogenomic.org/

3.4 Evaluation and Results

3.4.1 Evaluation Methodology

Genome Dataset and Hardware Setting

The evaluation is conducted on a machine with a 16-core Intel i7-11700K CPU with up to 5.0GHz frequency, 2TB NVMe PCIe 4.0 storage, and 64GB of DDR4 memory. Unless otherwise specified, all programs are allowed to use 16 threads with their default parameters. Five genome datasets in Table 3.2 are adopted for benchmarking. The datasets include: *Bacillus cereus*, *Escherichia coli*, NCBI RefSeq [88], Parks MAGs [111], and GTDB MAGs [112]. These datasets vary in terms of number of genomes, lengths, and sizes.

Benchmarking Tools

We compare HyperGen to five state-of-the-art tools, including Mash [33], Bindash [35], Sourmash [36], Dashing 2 [34], FastANI [88], Skani [89], and ANIm [86]. Mash, Bindash, Sourmash, and Dashing 2 are sketch-based tools for ANI estimation. In comparison, FastANI and Skani use mapping-based methods while ANIm adopts the most accurate base-level alignment-

Table 3.3: Names, versions, and commands of benchmarked genome tools for ANI calculation. The sketch-based tools include: Mash, Dashing 2, and HyperGen. The mapping-based tool is FastANI. The alignment-based tool is ANIm.

Tool	Version	Commands and arguments
HyperGen	v0.2.2	hyper-gen sketch -D cpu -t 16 -k 21 -s 1500 -d 4096 -p (fna_path) -o (file_out) hyper-gen sketch -D gpu -t 16 -k 21 -s 1500 -d 4096 -p (fna_path) -o (file_out) hyper-gen dist -t 16 -r (ref_sketch) -q (query_sketch) -o (dist_file)
Mash	v2.3	mash sketch (data set) -o (sketches) -p 16 mash dist (query genome) (sketches) -p 16
Bindash	v1.0	bindash sketch –nthreads=16 –listfname=(genome_list) –outfname=(sketch) bindash dist (query_sketch) (ref_sketch) –nthreads=16 –outfname=(dist_out)
Sourmash	v4.5	sourmash sketch dna –output-dir (sketches) (data set) sourmash compare (sketches)/*.sig -k 21 –max-containment –ani
Dashing 2	v2.1.19	dashing2 sketch –bagminhash -k 21 -S (sketch_size) -p 16 -F (file_list) dashing2 sketch –bagminhash –cache -k 21 -S (sketch_size) -p 16 -F (ref_file) -Q (query_file) dashing2 sketch –set –cache -k 21 -p 16 -F (file_list) –cmpout (file_out)
Skani	v0.2.1	skani sketch -t 16 -c 70 -m 1000 -l (genome_list) -o (sketches) skani dist -t 16 -q (query_sketches) -r (ref_sketches) -o (file_out)
FastANI	v1.33	fastANI –rl (genome_list) -q (query_genome) -t 16
ANIm (nucmer)	v0.2.12	average_nucleotide_identity.py -m ANIm –workers 16 -i (genomes) -o (output_folder)

based method to calculate the ANIs. ANIm results are regarded as the ground truth. Specifically, we use NUCleotide MUMmer [86] to generate the alignment results and then convert the alignment data into the corresponding ground-truth ANIs. Dashing 2 uses its weighted *bagminhash* mode. HyperGen (similar to Mash, Bindash, Sourmash, and Dashing 2) is an ANI approximation tool for the high ANI regime. We follow the previous work [33] and only preserve ANI values > 85. The versions and commands used are summarized in Table 3.3. HyperGen uses k-mer size $k = 21$, scaled factor $S = 1500$ as suggested in previous works [36, 89, 93]. Our analysis in Section 3.4.2 shows that the HV dimension $D = 4096$ achieves a good balance between ANI estimation error and sketching complexity. So we set it as the default parameter. HyperGen also supports the *fast mode* which accelerates the sketching process on GPU.

Evaluation Metrics

ANI Precision. One of the critical metrics for evaluating the effectiveness of a genome sketching tool is the precision of ANI estimation. We use three metrics to evaluate the ANI approximation errors: 1. mean absolute error (MAE), 2. root mean squared error (RMSE), and 3. mean percentage absolute error (MPAE). We also adopt the Pearson correlation coefficient to assess the linearity of the ANI estimate with respect to ground truth.

Computation and Memory Efficiency. An ideal genome sketching scheme should be able to generate compact sketch files at the cost of short runtime, especially for large-scale genomic analysis. To compare the computation and memory efficiency of evaluated tools, we measure and report the wall-clock runtime and sketch sizes during database search.

3.4.2 ANI Estimation Quality

In this section, we study the quality of ANI estimation by performing the following pairwise ANI experiment. First, the largest 100 genome files are collected from each dataset. Then, each batch of 100 genome files is used to calculate the pairwise and symmetric 100×100 ANI matrix.

HyperGen ANI Quality using Different Parameters

We first evaluate the impact of HyperGen's two algorithmic parameters: scaled factor S and HV dimension D on the final ANI estimation errors and linearity. The experimental results are depicted in Figure 3.4, where the scaled factor S and the HV dimension D vary from 800 to 2000 and from 256 to 16384, respectively. It shows that: for all scaled factors, the ANI approximation errors decrease significantly as D increases from 256 to 4096. This is because a larger HV dimension can produce better orthogonality, which is helpful to reduce the approximation error of the set intersection according to the theory in [103]. But increasing the HV dimension larger than $D = 4096$ does not yield a significant error reduction or linearity improvement.

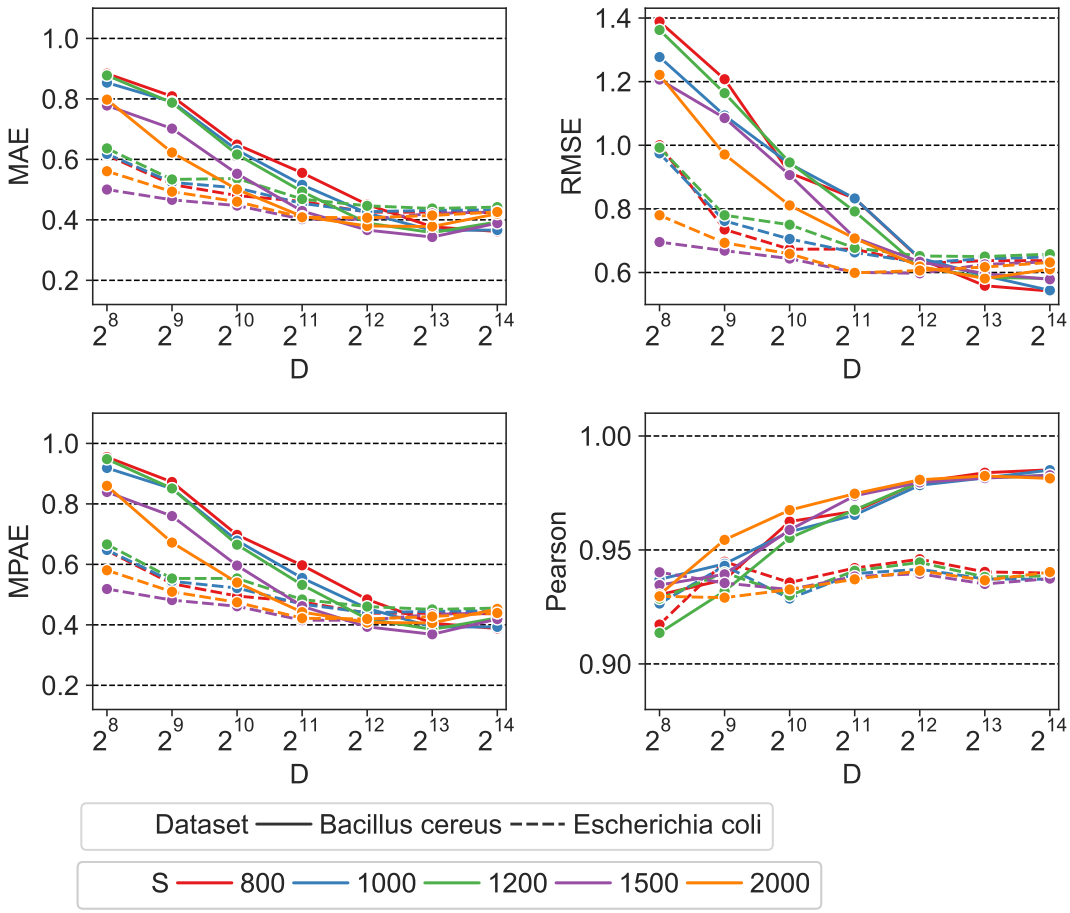


Figure 3.4: Error metrics (MAE, RMSE, MPAE) and ANI linearity (Pearson coefficient) as a function of scaled factor S and HV dimension D.

It is also observed that a smaller scaled factor S generally leads to a worse ANI approximation error when using the same HV dimension D. The reason behind this is: a smaller S that produces a larger hash threshold value as in Eq. (3.2), will generate a denser sampling of k-mers. This increases the size of sampled k-mer hash set. As a result, more binary HVs need to be aggregated to the sketch HV. The excessive number of binary HVs degrades the orthogonality between binary HVs, reducing the approximation accuracy for set cardinality. To balance between the quality and complexity of the ANI approximation, we choose $S = 1500$ and $D = 4096$ as the default scaled factor and HV dimension, respectively.

Table 3.4: Error metrics for the 100×100 pairwise Jaccard estimation. HyperGen-2048 and HyperGen-4096 use $D = 2048$ and $D = 4096$, respectively. Other tools use their default parameters. The ground truth values of Jaccard index are calculated using Dashing 2's exact mode. The command is given in Table 3.3 (The 3rd line of Dashing 2's commands).

Dataset: *Bacillus cereus*			
Tool	MAE ↓	RMSE ↓	MPAE ↓
Mash	0.008	0.011	3.860
Bindash	0.007	0.010	3.748
Dashing 2	0.011	0.016	6.458
Sourmash	0.004	0.005	3.123
HyperGen-2048	0.010	0.013	5.815
HyperGen-4096	0.007	0.009	4.274
Dataset: *Escherichia coli*			
Tool	MAE ↓	RMSE ↓	MPAE ↓
Mash	0.009	0.011	3.500
Bindash	0.007	0.009	2.498
Dashing 2	0.032	0.051	8.709
Sourmash	0.013	0.014	4.394
HyperGen-2048	0.010	0.013	3.267
HyperGen-4096	0.009	0.011	2.641

Comparison with Other Sketching Tools

We also compare the quality of the ANI estimation for various tools, including Mash, Bindash, Dashing 2, Sourmash, FastANI and Skani. For fair comparison, the sketch-based tools (HyperGen, Mash, Bindash, Sourmash, and Dashing 2) use the same sketch size. Other parameters are the same as their default parameters. Specifically, HyperGen uses $D = 4096$, while Mash and Dashing 2 use a sketch size of 1024.

HyperGen can be used to estimate the Jaccard index. First we perform Jaccard estimation experiment and compare HyperGen to Mash, Bindash, Dashing 2, and Sourmash. Table 3.4 shows the error metrics with respect to the true Jaccard results. The 100×100 Jaccard matrix for *Bacillus cereus* and *Escherichia coli* datasets is computed. HyperGen achieves competitive Jaccard estimation accuracy with other baseline tools.

Table 3.5: Error and linearity metrics for pairwise ANI estimation. (Underline: the best among sketch-based algorithms. **Bold**: the best among all algorithms.)

Dataset: *Bacillus cereus*						
Tool	k	Sketch Size	MAE ↓	RMSE ↓	MPAE ↓	Pearson ↑
FastANI	16	-	**0.312**	**0.368**	**0.334**	**0.999**
Skani	-	198MB (850×)	0.354	0.422	0.377	0.996
Mash	21	830KB (3.6×)	0.399	0.591	0.430	0.981
Bindash	21	351KB (1.5×)	0.360	0.530	0.385	0.986
Dashing 2	21	1.2MB (5.2×)	0.500	0.650	0.537	0.981
Sourmash	21	11MB (47×)	0.415	0.558	0.449	0.986
HyperGen-2048	21	233KB (1.0×)	0.411	0.707	0.442	0.975
HyperGen-4096	21	459KB (2.0×)	0.372	0.522	0.400	0.986
Dataset: *Escherichia coli*						
Tool	k	Sketch Size	MAE ↓	RMSE ↓	MPAE ↓	Pearson ↑
FastANI	16	-	0.680	1.152	0.705	0.899
Skani	-	200MB (855×)	0.403	0.572	0.419	**0.956**
Mash	21	831KB (3.6×)	0.456	0.686	0.470	0.930
Bindash	21	351KB (1.5×)	0.442	0.658	0.456	0.936
Dashing 2	21	1.2MB (5.1×)	0.464	0.704	0.479	0.930
Sourmash	21	9.6MB (41×)	0.381	**0.565**	0.393	0.944
HyperGen-2048	21	234KB (1.0×)	0.449	0.644	0.464	0.942
HyperGen-4096	21	460KB (2.0×)	**0.368**	**0.565**	**0.381**	0.952

Table 3.5 summarizes the ANI error and linearity metrics with respect to the ground truth values on *Bacillus cereus* and *Escherichia coli* datasets. For the *Bacillus cereus* dataset, HyperGen is slightly inferior to Bindash, FastANI and Skani, which yields a comparable Pearson correlation coefficient compared to the other sketch-based tools (Mash and Dashing 2). In the *Escherichia coli* dataset, HyperGen consistently surpasses all other sketch-based tools, providing both lower ANI approximation errors and better linearity. Meanwhile, HyperGen's sketch size is over 800× smaller than Skani. These experiments demonstrate that HyperGen is capable of delivering a high quality of ANI estimation.

3.4.3 Genome Database Search

One critical workload that genome sketching tools can accelerate is the genome database search. Meanwhile, the genome database search can be extended to multiple downstream

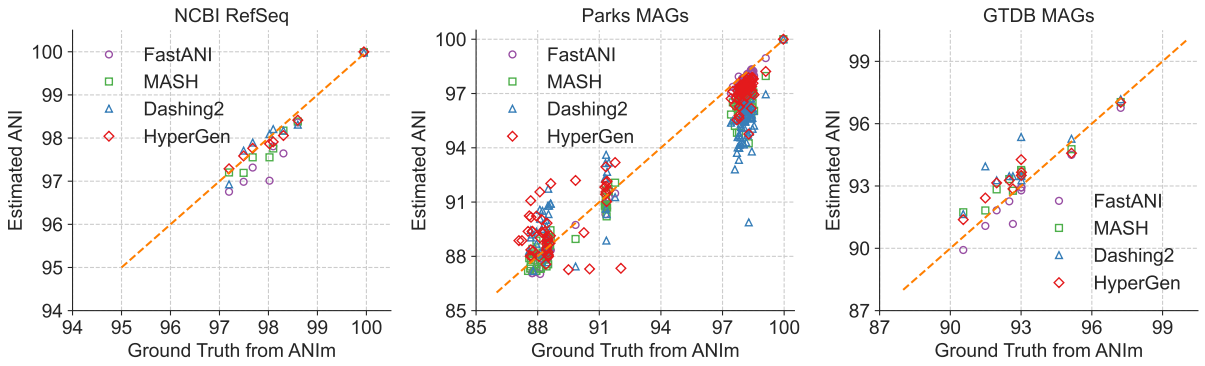


Figure 3.5: Database search ANI comparison for FastANI, Mash, Dashing 2, HyperGen, and ground-truth ANIm on NCBI RefSeq, Parks MAGs, and GTDB MAGs datasets.

applications.

ANI Linearity and Quality

We extensively consider the five evaluated datasets as reference databases. We run FastANI, Skani, Mash, Bindash, Dashing 2, and HyperGen using the commands and queries listed in Table 3.3. Sourmash is not considered because it does not support multi-thread execution. The execution consists of two steps: 1. All tools first generate reference sketches for the target database, 2. The second step is to search for the query genomes against the built reference sketches. Note that FastANI and Skani were unable to complete the database search on the Parks MAGs and GTDB datasets in one shot because it requires more memory than the available 64GB and experienced out of memory issues. We divided FastANI and Skani executions into smaller batches and measured the accumulative runtime.

The estimated ANI values generated in Table 3.6 by each tool in the NCBI RefSeq, Parks MAGs, and GTDB MAGs datasets are depicted in Figure 3.5 with their corresponding ground truth values from ANIm. Data points with ANI< 85 are filtered. It shows that HyperGen produces good ANI linearity compared to the ground truth results.

Quantitative results in terms of numerical error and linearity metrics are summarized in Table 3.6. The ANI error distribution for each tool can be seen in Figure 3.6. In datasets *Bacillus cereus*, *Escherichia coli*, and NCBI RefSeq, HyperGen achieves the lowest ANI errors among all

Table 3.6: Sketch size, error, and linearity metrics for database search. (Underline: the best among sketch-based algorithms. **bold**: the best among all algorithms.)

Dataset: *Bacillus cereus*						
Tool	k	Sketch Size	MAE ↓	RMSE ↓	MPAE ↓	Pearson ↑
FastANI	16	-	**0.218**	**0.296**	**0.235**	**0.999**
Skani	15	1.0GB (714×)	0.299	0.378	0.320	0.998
Mash	21	4.7MB (3.4×)	0.542	0.678	0.586	0.996
Bindash	21	2.0MB (1.4×)	0.467	0.579	0.502	0.994
Dashing 2	21	6.7MB (4.8×)	0.576	0.715	0.622	0.993
HyperGen-2048	21	1.4MB (1.0×)	0.355	0.480	0.382	0.994
HyperGen-4096	21	2.6MB (1.9×)	0.318	0.424	0.342	0.996
Dataset: *Escherichia coli*						
Tool	k	Sketch Size	MAE ↓	RMSE ↓	MPAE ↓	Pearson ↑
FastANI	16	-	0.215	0.391	0.221	0.950
Skani	15	6.9GB (697×)	0.198	**0.277**	0.203	**0.983**
Mash	21	36MB (3.6×)	0.226	0.529	0.231	0.877
Bindash	21	16MB (1.6×)	0.206	0.514	0.210	0.870
Dashing 2	21	51MB (2.6×)	0.234	0.536	0.239	0.873
HyperGen-2048	21	9.9MB (1.0×)	0.178	0.502	0.182	0.833
HyperGen-4096	21	20MB (2.0×)	**0.153**	0.491	**0.156**	0.851
Dataset: NCBI RefSeq						
Tool	k	Sketch Size	MAE ↓	RMSE ↓	MPAE ↓	Pearson ↑
FastANI	16	-	0.443	0.522	0.452	0.968
Skani	15	1.8GB (486×)	0.266	0.292	0.272	**0.997**
Mash	21	14MB (3.8×)	0.204	0.251	0.208	0.983
Bindash	21	**5.9MB** (1.6×)	0.238	0.269	0.243	0.988
Dashing 2	21	20MB (5.4×)	0.167	0.189	0.171	0.972
HyperGen-2048	21	3.7MB (1.0×)	0.216	0.304	0.234	0.991
HyperGen-4096	21	7.4MB (2.0×)	**0.135**	**0.164**	**0.138**	0.991
Dataset: Parks MAGs						
Tool	k	Sketch Size	MAE ↓	RMSE ↓	MPAE ↓	Pearson ↑
FastANI	16	-	0.457	0.551	0.490	**0.998**
Skani	15	6.6GB (367×)	**0.310**	**0.456**	**0.335**	0.997
Mash	21	65MB (1.9×)	1.090	1.298	1.137	0.990
Bindash	21	29MB (1.6×)	1.096	1.308	1.140	0.991
Dashing 2	21	93MB (5.2×)	2.163	2.466	2.251	0.921
HyperGen-2048	21	18MB (1.0×)	1.291	1.448	1.374	0.975
HyperGen-4096	21	35MB (1.9×)	1.146	1.297	1.211	0.983
Dataset: GTDB MAGs						
Tool	k	Sketch Size	MAE ↓	RMSE ↓	MPAE ↓	Pearson ↑
FastANI	16	-	**0.436**	**0.592**	**0.469**	0.976
Skani	15	66GB (458×)	0.466	0.630	0.500	0.969
Mash	21	533MB (3.7×)	0.584	0.668	0.632	**0.980**
Bindash	21	231MB (1.6×)	0.772	0.837	0.835	0.971
Dashing 2	21	770MB (5.3×)	0.994	1.283	1.078	0.892
HyperGen-2048	21	144MB (1.0×)	1.098	1.409	1.250	0.904
HyperGen-4096	21	287MB (2.0×)	0.982	1.138	1.094	0.974

sketch-based tools, delivering more accurate ANI estimations as compared to Mash, Bindash, and Dashing 2. HyperGen still shows competitive accuracy over mapping-based FastANI and

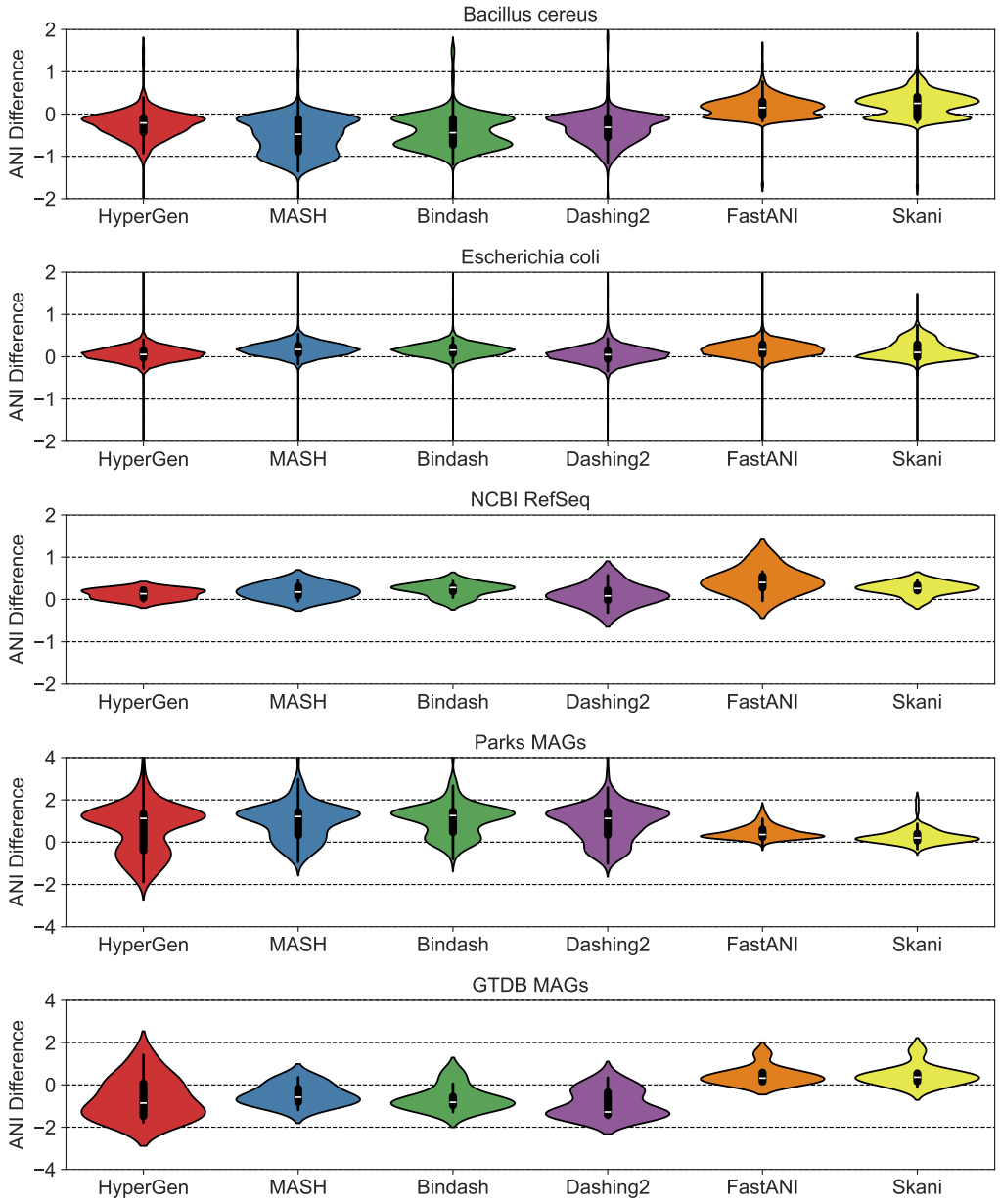
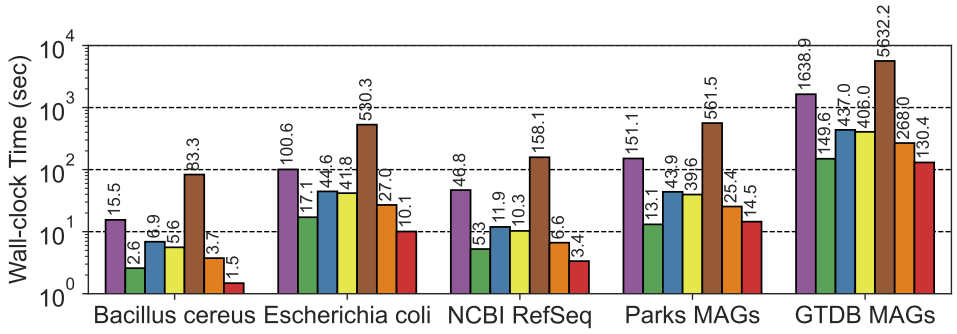
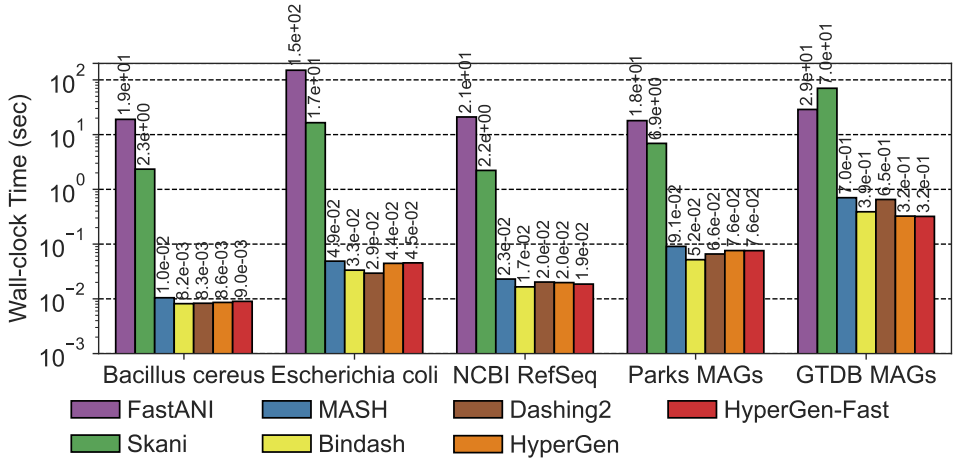


Figure 3.6: The ANI estimation error distribution of database search for all benchmarking tools (HyperGen, Mash, Bindash, Dashing 2, FastANI, and Skani). Data points with ANI > 85 are considered here.

Skani. In *Escherichia coli* and NCBI RefSeq, HyperGen outperforms FastANI and Skani in terms of most error metrics and produces comparable Pearson coefficients. HyperGen is capable of achieving state-of-the-art error and linearity for large-scale genome search. Meanwhile, the required sketch size is two orders of magnitude smaller than Skani.



(a) Sketching Time



(b) Search Time

Figure 3.7: Runtime performance comparison for genome search in Table 3.6. (a) Reference sketching time and (b) Query search time.

Runtime Performance

The wall-clock time spent on two major steps during database search: reference sketch generation and query search, is illustrated in Figure 3.7. HyperGen-Fast means using the *fast sketching mode* on GPU. The reference sketching step is mainly bounded by the sketch generation process, while the search step is bounded by the sketch file loading and ANI calculation. HyperGen without *fast mode* achieves the 2nd fastest sketching speed, slightly slower than Skani. After enabling *fast mode*, HyperGen is the fastest sketching tool for most evaluated datasets. The sketching speed of HyperGen is 2.7× to 4.1× faster than Bindash. HyperGen is significantly faster (10× to 13×) than the mapping-based FastANI.

Table 3.7: Benchmarking peak memory consumption and runtime for single-query search on GTDB MAGs dataset. OOM: out of memory.

Tool	Sketch Phase		Search Phase	
	Peak Memory	Runtime	Peak Memory	Runtime
FastANI	OOM	1,638.9s	OOM	28.8s
Skani	5.3GB	149.6s	OOM	70.2s
Mash	1.9GB	437.0s	1.0GB	0.7s
Bindash	0.3GB	406.0s	0.2GB	0.4s
Dashing 2	8.9GB	5,632.2s	0.6GB	0.7s
HyperGen	1.0GB	130.4s	0.9GB	0.3s

For query search, HyperGen is also one of the fastest tools. The search speedup of HyperGen over FastANI and Skani is 100× to > 3000× because FastANI and Skani require slow sequence mapping and large index file loading processes. Moreover, the speedup of HyperGen is more significant for larger datasets. Dashing 2 sketch size is about 2.6× of HyperGen so it takes more time to load sketch files. The reduced sketch size helps to save sketch loading time. Meanwhile, the HV sketch format of HyperGen allows us to adopt highly vectorized programs to compute ANI with a short processing latency.

Memory Efficiency

The file sizes of the reference sketches generated by Mash, Dashing 2, and HyperGen, are listed in Table 3.6. We apply the *Sketch Quantization and Compression* technique to HyperGen. As a result, HyperGen consumes the smallest memory space among the three sketch-based tools. The sketch sizes produced by Mash and Dashing 2 are 1.8× to 2.6× of HyperGen's sketch sizes. This suggests that HyperGen is the most space-efficient sketching algorithm. Compared to original datasets with GB sizes, a compression ratio of $600-1200\times$ can be achieved by only processing the sketch files. This enables the large-scale genome search on portable devices with memory constraints. HyperGen's memory efficiency comes from two factors. First, the *Hyperdimensional Encoding for k-mer Hash* step converts discrete hash values into continuous high-dimensional sketch HVs, which are more compact than hash values. Second, HyperGen's

Sketch Quantization and Compression provides additional 1.3× compression through further removing redundant information in sketch HVs.

Table 3.7 summarizes performance metrics in terms of peak memory consumption and runtime for the GTDB MAG dataset search. HyperGen achieve both the fastest sketching and search speed due to the efficient HDC algorithm as well as software optimizations. FastANI and Skani experience OOM (out of memory) issues because they require a large memory space to store intermediate data for sequence mapping. In comparison, HyperGen consumes about 1GB of memory for the sketching or searching phase, significantly lower than FastANI and Skani. This indicates that HyperGen is friendly to run on memory-limited device, such as laptop.

3.4.4 Discussion

Future directions of HyperGen include the following aspects:

Further Compression and Faster Large-scale Search: The vector representation of sketch HVs allows us to apply more optimizations on the top of HyperGen. For instance, we can employ lossy vector compression techniques, such as product quantization [113, 114] and residual quantization [115], to reduce sketch size and memory footprint. This is advantageous for achieving rapid genome database search on embedded or mobile devices.

On the other hand, the search step in HyperGen requires intensive GEMM operations to obtain ANI values between genomes. The large-scale database search can be further accelerated using advanced hardware architectures with high data parallelism and optimized interfaces. Previous work [101] demonstrates that deploying HDC-based bioinformatics analysis on GPU exhibits at least one order of magnitude speedup over CPU.

More genome workloads: HyperGen can be extended to support a wider range of genomic applications. For example, in metagenome analysis, we can utilize HyperGen to perform the containment analysis for genome files such as [108]. To realize this, the sketch HVs generated by HyperGen can be used to calculate the max-containment index instead of ANI. The ANI estimation error and memory requirements of HyperGen can be reduced by considering the more

accurate ANI estimation based on multi-resolution k-mers [107].

3.5 Conclusion

In this chapter, we present HyperGen, a genome sketching tool based on hyperdimensional computing (HDC) [103,106] that improves accuracy, runtime performance, and memory efficiency for large-scale genomic analysis. HyperGen avoids the expensive alignment process and realizes fast and accurate estimation for ANI, a standardized measure of genome file similarity. HyperGen inherits the advantages of both FracMinHash-based sketching [93, 94] and DotHash [103]. HyperGen first samples the k-mer set using FracMinHash. Then, the discrete k-mer hash set is encoded into the corresponding sketch HV in hyperdimensional space. This allows the genome sketch to be presented in compact vectors without sacrificing accuracy. HyperGen software implemented in Rust language deploys vectorized routines for both sketch and search steps. The evaluation results show that HyperGen offers superior ANI estimation quality over state-of-the-art sketch-based tools [33, 34]. HyperGen delivers not only the fastest sketch and search speed, but also the best memory efficiency in terms of the sketch file size.

HDC is a versatile and efficient computing paradigm that significantly enhances a wide range of bioinformatics workloads beyond genomics. In the next chapter, we apply the HDC approach to develop a high-performance clustering tool for mass spectrometry. Our work demonstrates that HDC offers a unified computing framework, serving as a foundational technique to accelerate genomics and proteomics.

This chapter contains material from "HyperGen: Compact and Efficient Genome Sketching using Hyperdimensional Vectors", by Weihong Xu, Po-kai Hsu, Niema Moshiri, Shimeng Yu, and Tajana Rosing, which appears in Bioinformatics, 2024. The dissertation author was the primary investigator and author of this paper.

Chapter 4

High-performance Clustering for Mass Spectrometry

4.1 Introduction

The previous chapter shows the effectiveness of applying hyperdimensional computing (HDC) to genome sketching. In this chapter, we show that HDC is a versatile computing approach that can be also used to develop high-performance tool for spectral clustering. Spectral clustering is an effective approach to shortening the spectral identification runtime by reducing the search space [12–16, 116]. Prior to peptide identification, highly similar MS/MS spectra are first clustered together, and each cluster is represented by a consensus spectrum. The benefits of this approach are threefold. First, clustering minimizes data redundancy by grouping repeated MS/MS spectra and representing them as a single consensus spectrum. Second, rather than having to analyze all raw spectra, downstream tools only need to process a smaller number of consensus spectra. For example, the authors in [13] report using spectral clustering to reduce the runtime of subsequent peptide identification by over 50%. Third, the downstream analysis can achieve better results by operating on high-quality consensus spectra with a higher signal-to-noise ratio compared to the raw spectra [117].

Previous spectral clustering tools have focused on optimizing clustering quality and clustering speed. For example, MS-Cluster [16] and spectra-cluster [15] use an iterative greedy approach to efficiently merge similar spectra. spectra-cluster has been utilized for large-scale

clustering of public MS data in the PRoteomics IDEntifications (PRIDE) database [118] to build the PRIDE-Cluster spectral libraries. MaRaCluster [14] proposed an optimized similarity metric that relies on the rarity of fragment peaks to compare MS/MS spectra. Based on the intuition that peaks shared by only a few spectra offer more evidence than peaks shared by a large number of spectra, relative to a background frequency of fragment peaks with specific *m/z* values, matches of frequent fragment peaks contribute less to the spectrum similarity than matches of rare peaks. Next, MaRaCluster performs hierarchical clustering with complete linkage to group similar spectra in clusters. However, the clustering speed of MS-Cluster, spectra-cluster, and MaRaCluster can be extremely slow on large datasets as they run on CPU hardware only, which lacks massive parallelism. For example, [12] report that these tools took 8 to 30 hours to cluster a draft human proteome dataset consisting of 25 million MS/MS spectra [119]. Unfortunately, such long clustering runtime negates any potential benefits of shortened runtime from downstream applications. Additionally, as current MS data repositories contain orders of magnitude more data, with several billions of MS/MS spectra currently available, performing spectral clustering at the repository scale becomes increasingly challenging and computationally infeasible.

Several other spectral clustering tools have tried to address this issue by focusing on processing speed. msCRUSH [13] utilizes locality-sensitive hashing (LSH) to achieve fast clustering speeds by projecting similar spectra into shared LSH buckets to avoid unnecessary pairwise spectrum comparisons. Within each bucket, msCRUSH then uses a greedy spectrum merging strategy similar to MS-Cluster and spectra-cluster to cluster the spectra. falcon [12] first converts spectra to low-dimensional vectors using a hashing strategy. It uses approximate nearest neighbor searching [120] to construct a sparse pairwise distance matrix, which helps to shorten the required runtime. ClusterSheep [116] further optimizes the spectral clustering runtime by offloading computations to a graphics processing unit (GPU). Compared to falcon, ClusterSheep implements function kernels on GPU to speed up further. Unfortunately, however, despite their efficient runtimes, falcon and ClusterSheep exhibit some reduction in clustering quality compared to MaRaCluster and msCRUSH. Consequently, existing spectral clustering

tools still lack the ability to yield high clustering quality within a short runtime when processing large-scale datasets.

In this chapter, we present HyperSpec, a GPU-accelerated spectral clustering library using brain-inspired HDC [106]. Unlike previous hashing-based methods that project spectra into a low-dimensional space, HDC instead encodes spectra into binary high-dimensional vectors, called *hypervectors* (HVs). Compared to the low-dimensional embeddings used by msCRUSH [13] and falcon [12], HVs are superior in the sense that spectra can be encoded as compact, binary vector representations with a minimal loss of information. Additionally, binary HDC operation in HyperSpec offers high data parallelism for low-level hardware architecture, which we leveraged by developing fast Python kernels tailored for exploiting GPU resources. By operating on spectra represented as HVs, HyperSpec achieves state-of-the-art clustering quality and clustering speed. Our experiments show that HyperSpec is scalable to different dataset sizes and significantly accelerates spectral clustering up to 15× compared to alternative clustering tools. As an example, clustering of a large draft human proteome dataset [119] was reduced from over 4 hours (using MaRaCluster) to only 24 minutes. Meanwhile, the peptide identification quality using clustered consensuses generated by HyperSpec is comparable with state-of-the-art tools. HyperSpec is freely available as open source on GitHub at https://github.com/wh-xu/Hyper-Spec under the BSD license.

4.2 HyperSpec: Fast Clustering Software for Mass Spectrometry

4.2.1 Overall Flow

HyperSpec is a Python library for spectral clustering, that optimally makes use of both CPU and GPU hardware resources (Figure 4.1-(a)). The overall data processing flow of HyperSpec consists of five main steps, including spectrum preprocessing, bucket division, hyperdimensional (HD) encoding, HD distance computation, and clustering. The first two steps, namely spectrum

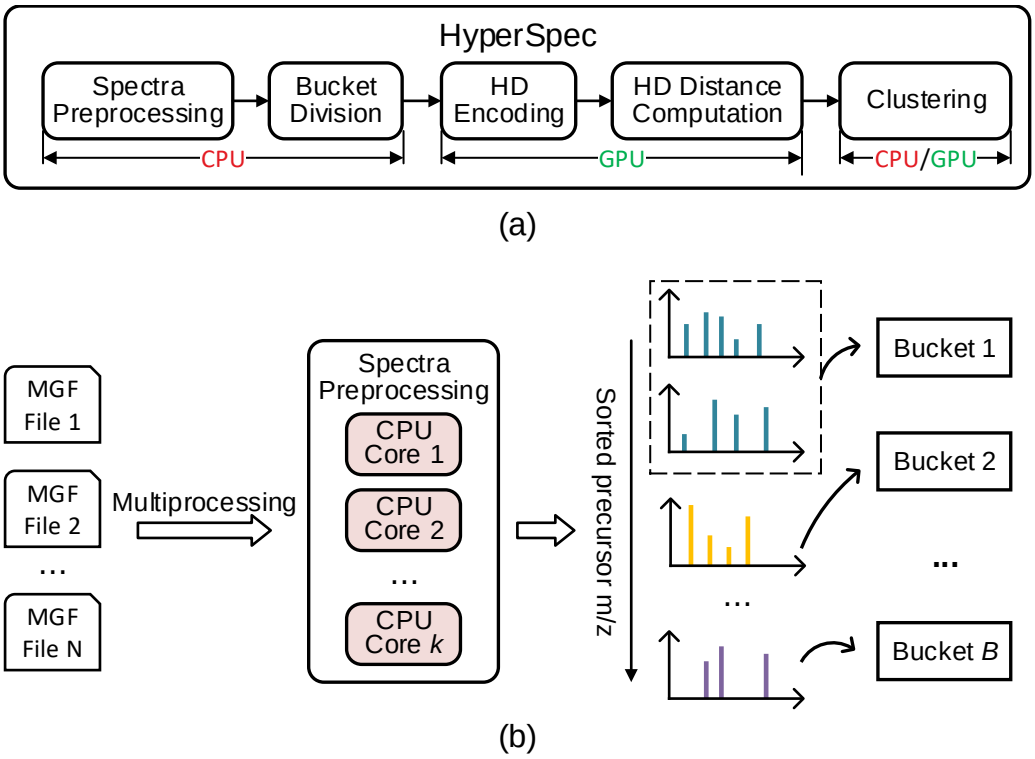


Figure 4.1: (a) Overall diagram of HyperSpec. (b) HyperSpec's spectrum preprocessing and bucket division flow. HyperSpec's spectra preprocessing and bucket division are optimized using multiprocessing on CPU. HD encoding and distance computation are offloaded to highly parallel GPU.

preprocessing and bucket division, are executed on CPU, while the HD encoding and distance computation are accelerated using GPUs. This allows HyperSpec to fully utilize both CPU and GPU computing resources for optimized preprocessing and clustering speed.

4.2.2 Efficient Spectrum Preprocessing

Prior to spectral clustering, the raw spectra need to be loaded and preprocessed. This is one of the bottlenecks during spectral clustering, contributing 20% to 90% of the overall runtime for several state-of-the-art spectral clustering tools (Figure 4.2). There are several reasons contributing to the slow preprocessing step: 1. During preprocessing, raw spectra are loaded and parsed from the storage device, after which the parsed spectra are processed to remove noise. The parsing phase is bounded by the speed of parsing spectral data into a numerical format, since

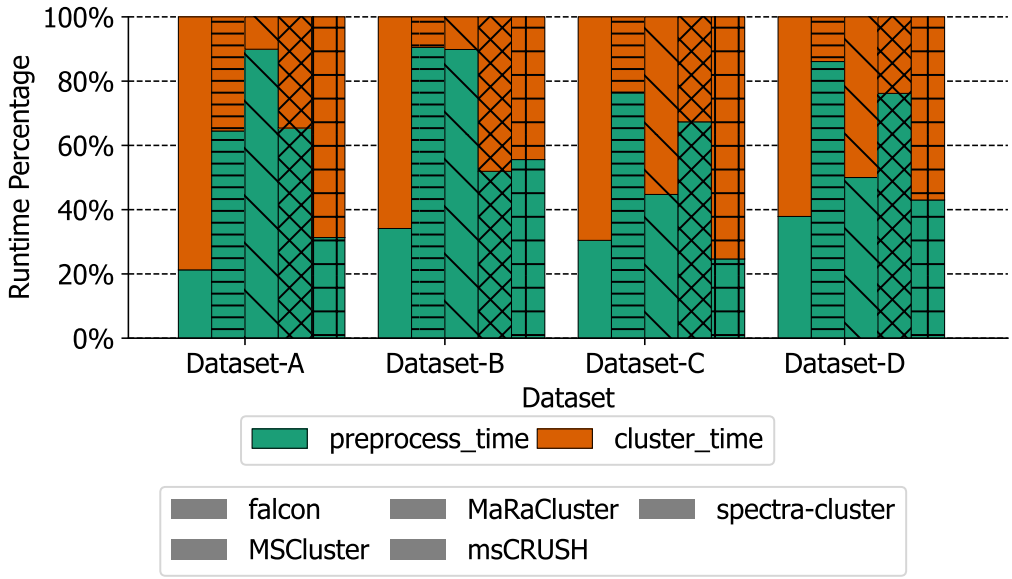


Figure 4.2: Runtime profiling for five clustering tools (falcon [12], msCRUSH [13], MaRaCluster [14], spectra-cluster [15], and MS-Cluster [16]). The runtimes were evaluated in terms of the time required for spectrum preprocessing and the time required for spectral clustering.

the peak information takes up over 95% of the data volume in raw files. 2. Processing the parsed spectra is computation-bounded, because it requires sorting, computing, and data manipulation for high-dimensional peak vectors. 3. Another crucial factor limiting the preprocessing speed of existing clustering tools is the underutilization of storage I/O bandwidth. Specifically, modern solid-state drives (SSD) provide GB/s sequential access speeds, but most clustering tools cannot provide sufficient preprocessing speed to match the I/O bandwidth. To this end, HyperSpec optimizes spectrum preprocessing as follows:

Multiprocessing. HyperSpec uses the commonly used Mascot Generic Format (MGF) as an input. HyperSpec utilizes multiprocessing to read each file in parallel and distribute the computation to k independent CPU cores (Figure 4.1-(b)). Spectrum preprocessing is composed of two phases: spectrum parsing and preprocessing. We implemented an optimized `spectrum data parser` and a `parallelized preprocessor`, which are executed independently on k CPU cores to increase data parallelism.

Spectrum data parser. Rather than using standalone C++ or Python parsing [12, 13, 16],

HyperSpec uses a hybrid C++-Python program, which balances Python's convenience of code extension and C++'s performance. The low-level `spectrum data parser` is built using the `Spirit X3` parser in Boost C++ [121] to convert MGF data to numerical arrays. After being compiled, the `spectrum data parser` is invoked by the high-level Python interface using multiprocessing.

Parallelized preprocessor. HyperSpec's `parallelized preprocessor` reduces the preprocessing time by vectorizing the computation. Specifically, the preprocessing operations are parallelized to multiple CPU cores using just-in-time (JIT) compilation by Numba [122]. The JIT compilation requires negligible human intervention while providing great portability for code extension and modification.

These modules are used to preprocess the spectra as follows. First, peaks related to the precursor ion or with $< 1\%$ intensity than the base peak intensity are removed. Second, spectra with < 5 valid peaks or with a < 250 Da mass range between their minimum and maximum peak are removed. Third, at most 50 peaks with the highest intensities are retained, and the peak intensities are normalized to $[0, 1]$ using their L2 norm.

4.2.3 Bucket Division

An important challenge while clustering large datasets, with n spectra, is that performing all pairwise spectrum comparisons results in a dense pairwise distance matrix with quadratic $O(n^2)$ complexity, which is prohibitive for large n. To reduce this requirement, we follow a simple and effective strategy by dividing spectra into buckets [12, 116]. After all MGF files have been processed by the `spectrum preprocessing` module, the spectra are sorted and organized by ascending precursor *m/z* order (Figure 4.1-(b)). Instead of having to cluster an entire dataset, the spectra are divided into smaller buckets as follows:

$$bucket_i = \lfloor \frac{(m/z_i - 1.00794) \times C_i}{1.0005079} \rfloor, \tag{4.1}$$

where C_i is the precursor charge and m/z_i is the precursor m/z associated with the ith spectrum, 1.00794 is the mass of the charge, and 1.0005079 corresponds to the distance between the centers of two adjacent clusters of physically possible peptide masses [123]. Each bucket is represented using an integer.

This bucket division scheme significantly reduces the memory usage and runtime by only comparing spectra within the same bucket to compute distance matrices for each bucket, instead of having to perform all pairwise spectrum comparisons for the full dataset.

4.2.4 GPU-accelerated Spectral Clustering in Hyperdimensional Space

HyperSpec exploits emerging HDC techniques [106, 124] to convert the preprocessed spectra into hyperdimensional space, where data are expressed as high-dimensional vectors with binary values. An important advantage of such HDC encoding is that the transformed data preserve features of the original space while exhibiting opportunities for data parallelism that can be leveraged by parallel GPU architectures [124]. Due to this reason, the final three steps of HyperSpec (HD encoding, HD distance computation, and clustering) can be significantly accelerated using GPU or CPU. HyperSpec clusters spectra by bucket granularity, meaning that one bucket is encoded, computed, and clustered at a time.

HD Encoding for Spectra

Whereas previous works [13, 14] directly compute spectrum similarities and perform clustering using the peak vectors, HyperSpec first uses HD encoding to project spectra to binary hypervectors (HVs) in the hyperdimensional space before performing the distance calculations (Figure 4.3). The HD encoding models the locality of the peak m/z and intensity values using two sets of encoding HVs (ID HVs $\mathbf{I}$ and level HVs $\mathbf{L}$), respectively. The ID HVs $\mathbf{I} \in \{\mathbf{I}_1, \mathbf{I}_2, \ldots, \mathbf{I}_f\}$ reflect the spatial locality of m/z values, while the level HVs $\mathbf{L} \in \{\mathbf{L}_1, \mathbf{L}_2, \ldots, \mathbf{L}_Q\}$ reflect the intensity of peaks, where f and Q are the maximum peak index range and intensity quantization levels, respectively. Both $\mathbf{I}_i$ and $\mathbf{L}_j$ are D-dimensional binary HVs, such that $\mathbf{I}_i, \mathbf{L}_j \in \{0, 1\}^D$.

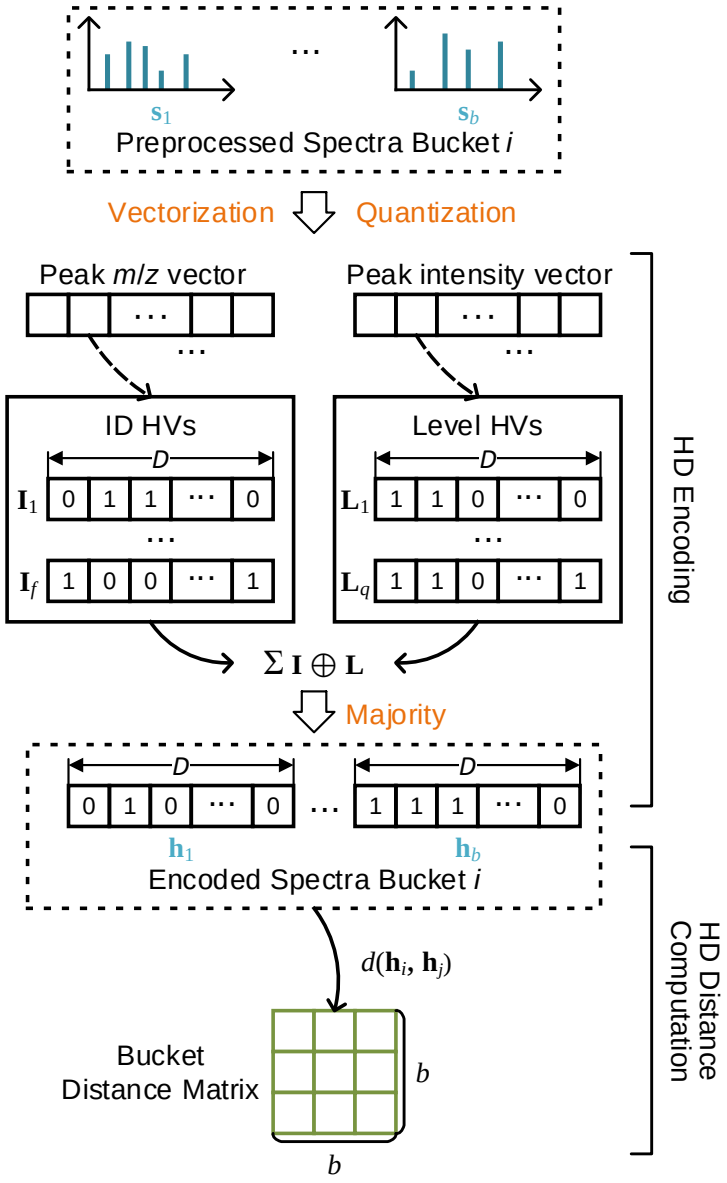


Figure 4.3: HD encoding and distance computation on GPU. Each preprocessed spectrum's *m/z*and intensity after vectorization and quantization are encoded into single hypervector (HV). Then the bucket distance matrix is computed.

The two sets of encoding HVs, $\mathbf{I}$ and $\mathbf{L}$, are iteratively generated in a stochastic manner. For ID HVs $\mathbf{I}$, first a random HV is generated and regarded as $\mathbf{I}_1$. Next, the ith HV $\mathbf{I}_i$ is generated by randomly flipping a specific number of bits from its preceding HV $\mathbf{I}_{i-1}$. The default number of flipped bits is $\frac{D}{2}$. For level HVs $\mathbf{L}$, the generation process of first HV $\mathbf{L}_1$ is identical to $\mathbf{I}$. The difference is that level HVs generate the ith HV $\mathbf{L}_i$ by flipping $\frac{D}{Q}$ bits compared to the preceding

HV $\mathbf{L}_{i-1}$. The impact of the generation parameters on the clustering quality is discussed in Section 4.3.

In the HD spectrum encoding process, the spectra in each bucket are first converted and quantized to two sparse vectors: peak *m/z* and peak intensity vectors. Based on the *m/z* and intensity pairs (i, j) in the peak *m/z* and peak intensity vectors, HyperSpec's HD encoder finds the associated ID HV I_i and level HV L_j in the encoding HV sets. The fetched ID HV I_i and level HV L_j are then point-wise XORed by $I_i \oplus L_j$. After all $(i, j) \in \mathbb{P}$, where $\mathbb{P}$ denotes the set of peak *m/z* and intensity vectors, are computed, the HD encoded spectrum is generated as follows:

$$\mathbf{h} = \text{Majority}\left(\sum_{(i,j)\in\mathbb{P}} \mathbf{I}_i \oplus \mathbf{L}_j\right), \tag{4.2}$$

where Majority$(\cdot)$ denotes the point-wise majority function that generates the binarized spectrum HV $\mathbf{h} \in \{0, 1\}^D$.

The HD dimension D needs to be large (normally > 1000) to guarantee representation capability [124]. However, because such a large dimension incurs an expensive encoding cost, we have made two optimizations to reduce the encoding overhead:

Bit-packing. By default, existing CPU or GPU architectures have a byte-level data granularity. However, storing a binary HV as a byte array needs 8× larger space than the theoretical number of bits D. To increase the memory efficiency of HyperSpec, HVs are stored in a bit-packed data structure, where every 32 bits of a HV $\mathbf{h}$ are packed into a 32-bit integer and each HV is stored in an integer array with length $\frac{D}{32}$, which reduces the memory requirements to store a HV 8-fold.

Batched GPU parallel encoding. The HD encoding is a bit-parallel algorithm, such that each bit of $\mathbf{h}$ can be computed independently. We have implemented the HD encoding modules using the CUDA platform [125] and the HDC-specialized GPU memory optimization scheme [126] to exploit this parallelism on GPUs. Before starting the HD encoding process, ID HVs $\mathbf{I}$ and level HVs $\mathbf{L}$ are transferred to the GPU memory. We found that data transfer of the

HVs incurs a large overhead, since the size of the ID HVs $\mathbf{I}$ is much larger than the size of a single encoded spectrum. To reduce this overhead, the GPU parallel encoding in HyperSpec is performed in a batch-wise manner, where the HV data are transferred while a batch of spectra are processed.

HD Distance Computation

The clustering step of HyperSpec operates on the pairwise distance matrix of each bucket (also called bucket distance matrix). We use a normalized Hamming distance to measure the similarity between spectrum HVs. For two binary encoded spectra $\mathbf{h}_i$ and $\mathbf{h}_j$, the Hamming distance is first computed by counting the set bits of their XOR result $\mathbf{h}_i \oplus \mathbf{h}_j$. Then the Hamming distance is normalized to $[0, 1]$ by dividing D. Consequently, the pairwise distance $d(\mathbf{h}_i, \mathbf{h}_j)$ is computed as:

$$d(\mathbf{h}_i, \mathbf{h}_j) = \frac{\text{popcount}\left(\mathbf{h}_i \oplus \mathbf{h}_j\right)}{D}, \tag{4.3}$$

where popcount$(\cdot)$ denotes the operation that obtains the number of set bits in a binary vector.

The HDC-based distance computation is lightweight because the computation of Eq. (4.3) only needs XOR and ones-counting operations. Our efficient implementation of these distance calculations leverages two CUDA integer intrinsics, `XOR` and `popc`. Additionally, by operating on bit-packed HVs, the time complexity to calculate each value in the pairwise distance matrix is reduced from D to $\frac{D}{32}$.

Clustering Algorithms

HyperSpec supports two popular clustering algorithms—DBSCAN [127] and hierarchical clustering [128] — to cluster each spectra bucket based on the bucket distance matrix. HyperSpec implements these two clustering algorithms due to their three common benefits:

1. DBSCAN and hierarchical clustering have been previously demonstrated effective to generate satisfactory quality for spectral clustering [12, 37, 129]. The analysis in Section 4.3.2

shows DBSCAN and hierarchical clustering yield various trade-offs between runtime and clustering quality. Supporting both of them allows the users to have more flexible choices.

2. DBSCAN and hierarchical clustering require minimal efforts to tune the algorithmic hyperparameters, as the number of clusters does not need to be specified explicitly.

3. From the perspective of runtime performance, the off-the-shelf fast DBSCAN and hierarchical clustering implementations [128, 130] are available and the clustering speed scales well to million or even billion-scale scenarios.

4.2.5 Software Development and Code Availability

HyperSpec was implemented in Python 3.8. The MGF loading and parsing functions were written in C++ and compiled to Cython interfaces [131] that can be invoked by Python. The spectrum preprocessing functionality was parallelized using the JIT compilation library Numba (version 1.20.2) [122]. The HD encoding and distance computation functions on GPU were implemented using Numba and CuPy. We used the DBSCAN available in the cuML library (version 22.04) [130] of the RAPIDS [132] framework and fast hierarchical clustering with complete linkage in fastcluster [128] to perform clustering on GPU and CPU, respectively. HyperSpec is publicly available as open source at https://github.com/wh-xu/Hyper-Spec under the BSD license.

4.3 Evaluation

4.3.1 Evaluation Methodology

Clustering Quality Metrics

We used the following metrics to evaluate the clustering quality and runtime performance:

- **Clustered spectra ratio.** The clustered spectra ratio equals the number of clustered spectra divided by the total number of spectra. This metric evaluates the clustering capability of the corresponding clustering tool.

- **Incorrect clustering ratio.** Incorrectly clustered spectra are those spectra whose peptide labels deviate from the most frequent peptide label in their clusters. The incorrect clustering ratio is the number of incorrectly clustered spectra divided by the total number of clustered and identified spectra.

- **Completeness.** Completeness measures the fragmentation of spectra corresponding to the same peptide across multiple clusters and is based on the notion of entropy in information theory. A clustering result that perfectly satisfies the completeness criterium (value "1") assigns all PSMs with an identical peptide label to a single cluster. Completeness is computed as one minus the conditional entropy of the cluster distribution given the peptide assignments divided by the maximum reduction in entropy the peptide assignments could provide [133].

- **Runtime.** The runtime is defined as the wall clock time between the start of spectrum preprocessing and the finish of the clustering procedure. We use the Linux system command to measure the wall clock time. The time for generating cluster consensus spectra was excluded since the overhead hereof is small.

Hardware Configurations

The runtime performance of all clustering libraries was measured on a server with a 12-core CPU, 128GB DDR4 memory, and a 2TB NVMe solid-state drive (SSD). The equipped GPU card was an NVIDIA GeForce RTX 3090 GPU with 24GB RAM. All tools were allowed to use all available processor cores and threads.

Benchmarks

We compared HyperSpec to six state-of-the-art spectral clustering libraries, including GLEAMS [37], falcon [12], msCRUSH [13], MaRaCluster [14], spectra-cluster [15], and MS-Cluster [16]. The clustering quality was controlled by varying the spectrum similarity threshold values, while the other configurations were set to the default values without explicit

Table 4.1: Properties of the evaluated MS datasets

Dataset	Sample Type	PRIDE ID	# of Spectra	Size
Dataset-A	Kidney cell [134]	PXD001468	1.1M	5.6GB
Dataset-B	Kidney cell [135]	PXD001197	1.1M	25GB
Dataset-C	HeLa proteins [137]	PXD003258	4.1M	54GB
Dataset-D	HEK293 cell [136]	PXD001511	4.2M	87GB
Dataset-E	Human proteome draft [119]	PXD000561	21.1M	131GB

specifications. The distance threshold during clustering in HyperSpec was from 0.2 to 0.45. GLEAMS' distance threshold for agglomerative clustering with complete linkage was 0.2 to 0.7. The cosine distance threshold of falcon was 0.05 to 0.25. msCRUSH's cosine similarity threshold was varied from 0.3 to 0.8. MaRaCluster's *P*-value was -30 to -3. The clustering threshold for spectra-cluster was 0.8 to 0.99999. MS-Cluster's mixture probability was from 0.00001 to 0.1.

Dataset

We used five MS datasets at different scales for evaluation (Table 4.1). These datasets consist of various human proteomics data, such as the HEK293 cell line [134–136], HeLa [137], and a draft map of the human proteome [119]. For all datasets, raw files were downloaded from PRIDE [138] and converted to MGF files using ThermoRawFileParser [139].

For each dataset, spectra with precursor charge 2 and precursor charge 3 were considered. The largest dataset, PXD000561 [119], was used for runtime and clustering quality evaluation. The corresponding spectrum identifications were downloaded from MassIVE reanalysis RMSV000000091.3. These identifications were obtained via automatic reanalysis of public data on MassIVE using MS-GF+ [140]. Spectra were searched against the UniProtKB/Swiss-Prot human reference proteome (downloaded 2016/05/23) [141] augmented with common contaminants. Search settings included a 50 ppm precursor mass tolerance, trypsin cleavage with maximum one non-enzymatic peptide terminus, and cysteine carbamidomethylation as a static modification. Methionine oxidation, formation of pyroglutamate from N-terminal glutamine, N-terminal carbamylation, N-terminal acetylation, and deamidation of asparagine and glutamine

Table 4.2: Clustering quality on Dataset-E for different clustering algorithms and values of HD dimension D

Algorithm	HD Dimension D	Quantization Level Q	Clustered Spectra	Incorrect Clustered Spectra	Completeness
DBSCAN	$D = 128$	$Q = 16$	8 001 982 (37.89%)	2.17%	0.8979
DBSCAN	$D = 256$	$Q = 16$	7 107 355 (33.66%)	1.62%	0.8829
DBSCAN	$D = 512$	$Q = 16$	6 365 562 (30.14%)	1.40%	0.8693
DBSCAN	$D = 1024$	$Q = 16$	5 985 007 (28.34%)	1.34%	0.8656
DBSCAN	$D = 2048$	$Q = 16$	5 958 478 (28.20%)	1.30%	0.8638
DBSCAN	$D = 4096$	$Q = 16$	5 847 470 (27.67%)	1.28%	0.8615
Hierarchical	$D = 128$	$Q = 16$	14 809 404 (70.09%)	2.16%	0.8071
Hierarchical	$D = 256$	$Q = 16$	13 643 733 (64.57%)	1.67%	0.8154
Hierarchical	$D = 512$	$Q = 16$	13 266 246 (62.78%)	1.57%	0.8242
Hierarchical	$D = 1024$	$Q = 16$	13 301 599 (62.95%)	1.58%	0.8331
Hierarchical	$D = 2048$	$Q = 16$	13 289 980 (62.90%)	1.59%	0.8377
Hierarchical	$D = 4096$	$Q = 16$	13 292 059 (62.90%)	1.59%	0.8406

were specified as variable modifications, with maximum one modification per peptide. The remaining four datasets were used for runtime evaluation only.

4.3.2 Clustering Quality Comparison

HyperSpec Clustering Quality using Different Parameters

We studied the impact of HD parameters and clustering algorithms on HyperSpec's clustering quality to select the optimal parameter combination. For HD, the two hyperparameters that influence the capability to represent spectra as HVs, and thus impact the spectrum clustering quality, are the HV dimension D and quantization level Q. Using the draft human proteome Dataset-E, we examined the clustering quality using different combinations of clustering algorithms (DBSCAN or hierarchical clustering with complete linkage), HV dimension D (Table 4.2), and quantization level Q (Table 4.3), fixing the clustering distance threshold. DBSCAN and hierarchical clustering with complete linkage were used. The default distance threshold was $eps = 0.2$ for DBSCAN and $eps = 0.3$ for hierarchical clustering with complete linkage, respectively.

As in Table 4.2, the HV dimension D was varied between 128 and 8192 and three clustering quality metrics (clustered spectra ratio, incorrect clustering ratio, and completeness) were computed for each combination of clustering algorithm and D value. This evaluation showed that as the HV dimension D increases, the incorrect clustering ratio and the clustered spectra

Table 4.3: Clustering quality on Dataset-E for different clustering algorithms and values of HD quantization level Q

Algorithm	HD Dimension D	Quantization Level Q	Clustered Spectra	Incorrect Clustering Ratio	Completeness
DBSCAN	$D = 2048$	$Q = 4$	6 278 284 (29.73%)	1.41%	0.8644
DBSCAN	$D = 2048$	$Q = 8$	6 092 636 (28.85%)	1.33%	0.8621
DBSCAN	$D = 2048$	$Q = 16$	5 958 478 (28.20%)	1.30%	0.8638
DBSCAN	$D = 2048$	$Q = 32$	5 931 716 (28.09%)	1.29%	0.8596
DBSCAN	$D = 2048$	$Q = 64$	5 930 616 (28.08%)	1.29%	0.8595
Hierarchical	$D = 2048$	$Q = 4$	13 543 166 (64.10%)	1.66%	0.8415
Hierarchical	$D = 2048$	$Q = 8$	13 362 120 (63.24%)	1.60%	0.8389
Hierarchical	$D = 2048$	$Q = 16$	13 289 980 (62.90%)	1.59%	0.8377
Hierarchical	$D = 2048$	$Q = 32$	13 301 802 (62.95%)	1.59%	0.8378
Hierarchical	$D = 2048$	$Q = 64$	13 302 179 (62.96%)	1.59%	0.8378

ratio for both two clustering algorithms decreased. However, the completeness of DBSCAN decreases from 0.8979 to 0.8615 while hierarchical clustering's completeness is improved from 0.8071 to 0.8406. This is because a larger D allows the HVs to more granularly represent the spectra after encoding, their corresponding similarities will more accurately reflect the true spectral similarities and avoid that spectra corresponding to different peptides are incorrectly grouped together. The clustering results become less complete for DBSCAN as the density-based DBSCAN is unable to form large clusters when the spectral similarities are more accurate. Larger D also increases the memory usage for HV encoding and storing. The HV dimension $D = 2048$ balances well between clustering quality and memory consumption. We used $D = 2048$ as the default value for HV dimension.

In Table 4.3, we fixed $D = 2048$ and then varied the quantization level Q from 4 to 64 and calculated the corresponding clustering quality metrics for each quantization level. Increasing quantization level Q reduced the clustered spectra ratio as well as completeness while slightly improving the incorrect clustering ratio for both two clustering algorithms. For DBSCAN, the incorrect clustering ratio dropped from 1.41% to 1.29% while completeness dropped from 0.8644 to 0.8595 as Q is increasing from 4 to 64. Overall, the clustering quality is less sensitive to the change of quantization level Q. We choose $Q = 16$ as the default value for quantization level.

We find hierarchical clustering with complete linkage achieves better clustering spectra ratio and lower incorrect clustering ratio as compared to DBSCAN. In the following sections, we

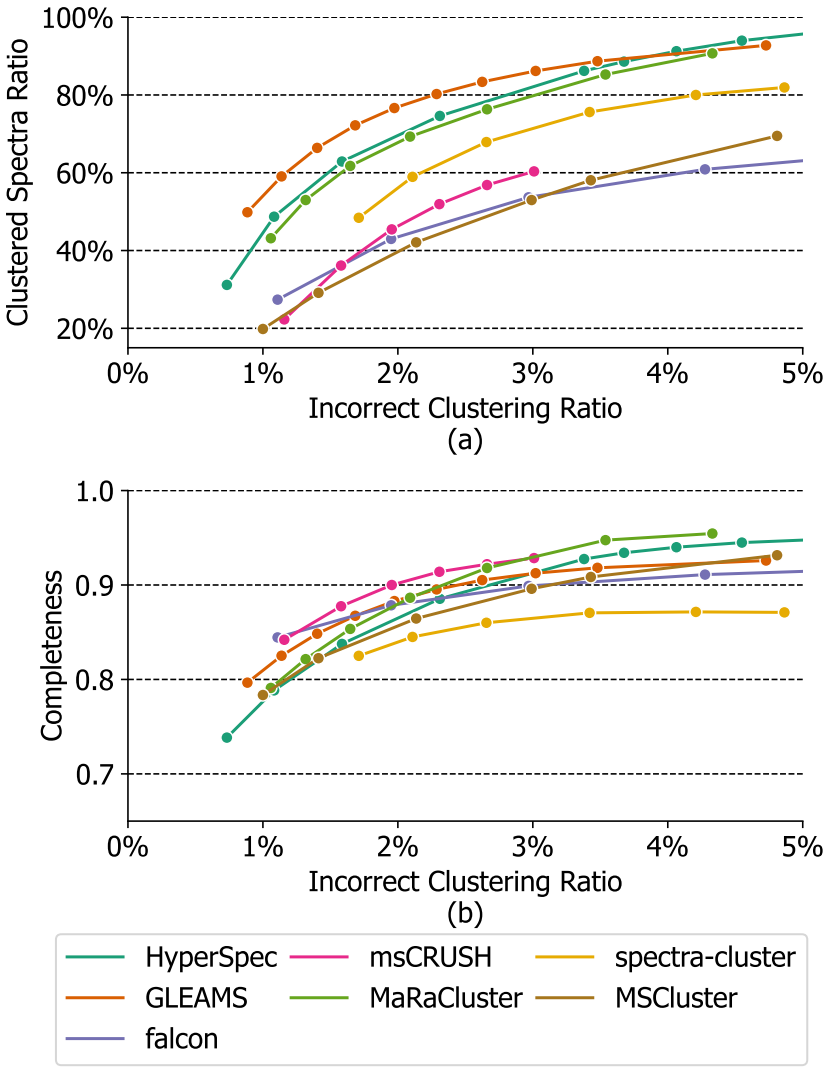


Figure 4.4: Clustering quality comparison for seven clustering tools: (a) clustered spectra ratio vs incorrect clustering ratio, (a) clustering completeness vs incorrect clustering ratio.

use hierarchical clustering as the default clustering algorithm without explicit specifications.

Comparison with Existing Tools

Using the draft human proteome Dataset-E, we also compared the clustering quality of HyperSpec to six alternative spectral clustering tools (Figure 4.4). As suggested previously [15, 129], a high clustered spectra ratio at a low incorrect clustering ratio indicates a better clustering capability for a specific tool. Additionally, completeness measures fragmentation of the same

peptide over multiple clusters, and an ideal clustering result should be as complete as possible to ensure that spectra originating from the same peptide are more likely to be grouped into a single cluster.

For clustered spectra ratio shown in Figure 4.4-(a), HyperSpec is significantly higher than falcon and MS-Cluster across different incorrect clustering ratio. Meanwhile, HyperSpec consistently clusters more spectra than MaRaCluster, and is slightly inferior to GLEAMS, achieving the second best result at the region with low incorrect clustering ratio.

In terms of completeness, HyperSpec outperforms spectra-cluster, MS-Cluster, and falcon, achieving top-3 completeness among the six clustering tools according to Figure 4.4-(b). In contrast to falcon and spectra-cluster, which reach a plateau in terms of completeness as the incorrect clustering ratio increases, HyperSpec is able to trade off a small amount of incorrect clustering ratio for more complete clustering results. HyperSpec also maintains high completeness values as the incorrect clustering ratio increases. For the region with incorrect clustering ratio $> 3\%$, HyperSpec surpasses other counterparts except for MaRaCluster, suggesting that the clusters produced by HyperSpec are generally less fragmented. This can be especially beneficial for downstream analysis tasks, since more complete clustering results can be represented by a smaller number of consensus spectra to optimally minimize data redundancy.

To intuitively understand the clustering results, we studied the distribution of cluster sizes for the most frequently identified peptide sequence `VATVSIPR` with precursor charge 2 for different spectral clustering tools (Figure 4.5). Here, HyperSpec used a threshold of $eps = 0.25$, HD dimension of $D = 2048$, and quantization level of $Q = 16$ to achieve a clustering with a ratio of incorrectly clustered spectra $< 1.2\%$. The other spectral clustering tools use configurations as listed in Table 4.4. We can see that HyperSpec mostly forms medium-size clusters with size 5 to 500 as compared to falcon and msCRUSH which tend to generate large clusters (size$>$ 500). The majority of clusters produced by MaRaCluster and spectra-cluster contain less than 100 spectra, which indicates that these two tools are more likely to group spectra corresponding to the same peptide into multiple small and fragmented clusters. In comparison, falcon and msCRUSH group

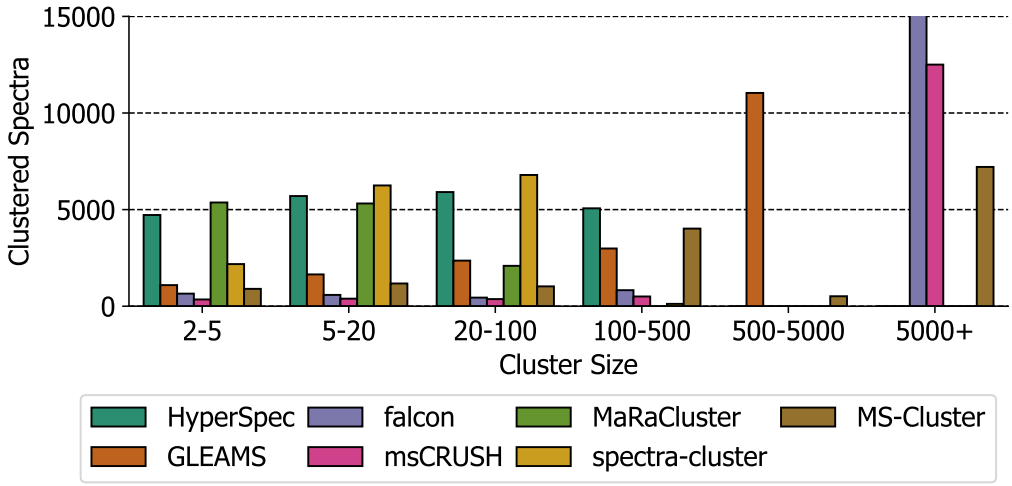


Figure 4.5: Distribution of cluster sizes for the most frequently identified peptide sequence `VATVSIPR` with precursor charge 2.

these spectra into a limited number of large clusters that contain at least 5000 spectra. We also add the six most frequent peptide sequences on Dataset-E with charge 2 and charge 3 as shown in Figure 4.6 to illustrate the distribution of the cluster sizes. The incorrect clustering ratios for all clustering results were controlled at around 1.0%. HyperSpec tends to form medium-size clusters from size 5 to 500.

4.3.3 Spectra Database Searching Comparison

The generated consensuses from spectra clustering tools can be used for the downstream spectra database search to identify peptide sequences. We compared the spectra searching performance on the Human proteome draft Dataset-E in Table 4.1 for three clustering tools, including HyperSpec, GLEAMS, and falcon. The clustering results generated by these tools were controlled to have clustered spectra ratio around 60%. We use the default parameters provided by the software except for the distance thresholds. Specifically, HyperSpec uses a distance threshold value=0.3 and produces 62.9% clustered ratio with 1.58% incorrect clustering ratio. GLEAMS uses a distance threshold value=0.25 and produces 59.1% clustered ratio with 1.14% incorrect clustering ratio. falcon uses a distance threshold value=0.2 and produces 61.1%

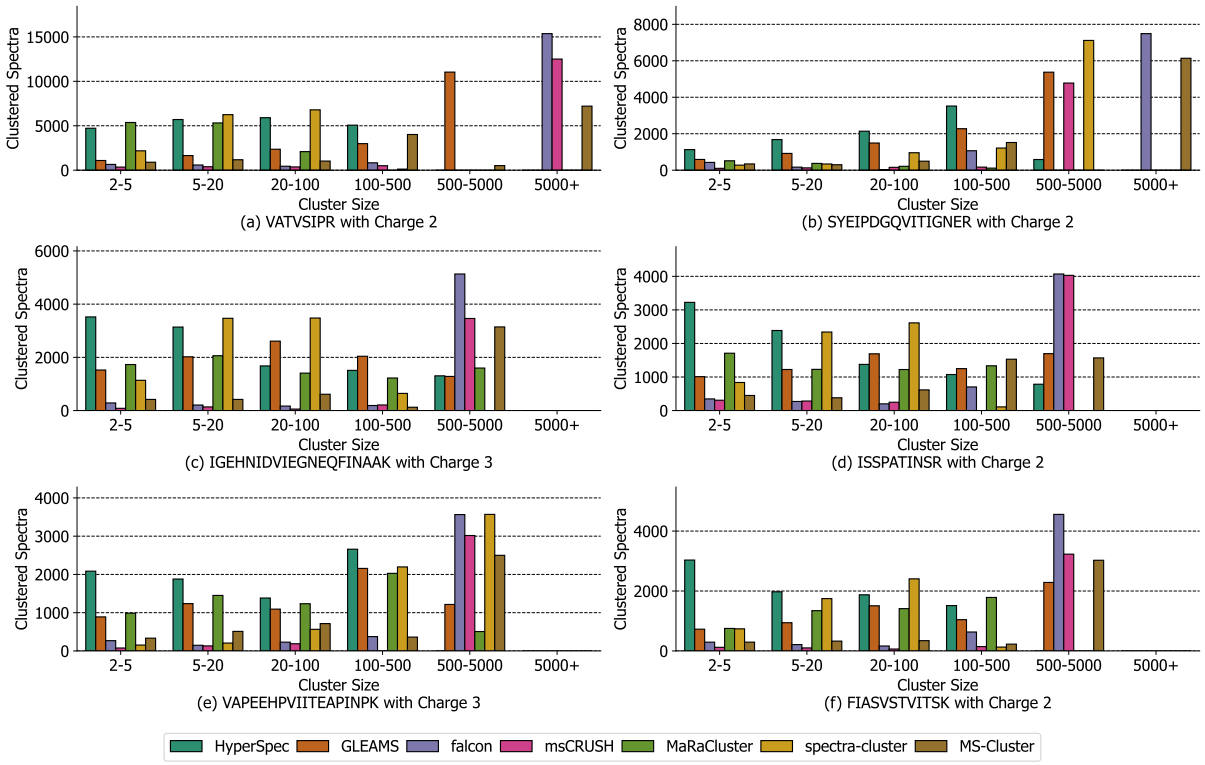


Figure 4.6: Distribution of cluster sizes for the six most frequently identified peptide sequences on Dataset-E with precursor charge 2 and charge 3.

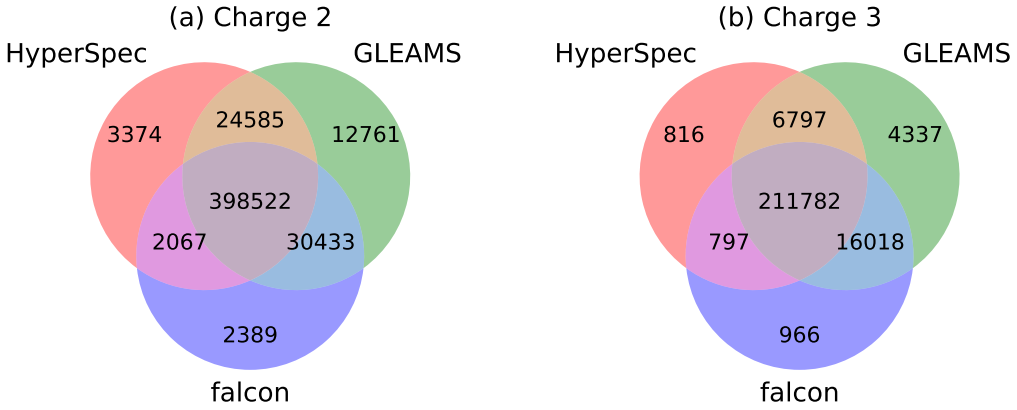


Figure 4.7: Venn diagrams that depict the overlap of identified unique peptides using consensus spectra generated by HyperSpec, GLEAMS, and falcon, respectively. The precursor charges include charge 2 in (a) and charge 3 in (b). Identified peptides from HyperSpec are highly overlapped with the results generated by GLEAMS and falcon.

clustered ratio with 4.27% incorrect clustering ratio. The clustering consensuses were searched using MSGF+ [140] with the same parameters given in Section 4.3.1.

Table 4.4: Key performance metrics of HyperSpec, GLEAMS, falcon, msCRUSH, and MaRaCluster on the draft human proteome Dataset-E.

Tool	Parameters	Runtime	Peak Memory	Clustered Spectra	Incorrect Clustering Ratio	Completeness
HyperSpec	$eps = 0.25$, $D = 2048$, $Q = 16$	24min	54GB	10 290 245 (48.70%)	1.08%	0.7885
GLEAMS	threshold=0.25	217min	34GB	12 392 427 (59.06%)	1.14%	0.8251
falcon	$eps = 0.05$	161min	87GB	5 675 468 (27.42 %)	1.11%	0.8438
msCRUSH	similarity=0.8	55min	91GB	4 397 921 (22.34 %)	1.16 %	0.8418
MaRaCluster	pvalThreshold = -30.0	251min	19GB	9 305 471 (43.20 %)	1.07%	0.7911

Figure 4.7 illustrates the Venn diagrams that depict the overlap relationship of identified unique peptides using consensus spectra clustered by HyperSpec, GLEAMS, and falcon. GLEAMS identifies the largest number of unique peptides. HyperSpec identifies 8.1% and 1.1% less unique peptides for charge 2 and identifies 7.8% and 4.1% less unique peptides for charge 3 as compared to GLEAMS and falcon, respectively. It should be noted that HyperSpec achieves much lower incorrect clustering ratio than falcon (1.58% v.s. 4.27%). Considering that HyperSpec is significantly faster than GLEAMS and falcon, we believe its slight degradation of spectra searching quality is acceptable. Furthermore, HyperSpec not only boosts the spectra clustering procedure but also reduces the search time of spectra database search. HyperSpec yields about 2.7× speedup over the spectra searching using raw spectra because the redundant searching process for those similar spectra are skipped.

4.3.4 Runtime Performance Comparison

Runtime is a crucial metric to evaluate the efficiency of spectral clustering tools. Especially to be able to perform spectral clustering at the repository scale, tools have to be fast to handle the ever-growing amount of MS data that is available in public data resources.

We first compared the total clustering time of HyperSpec using DBSCAN or hierarchical clustering with complete linkage on five datasets. Figure 4.8 shows that hierarchical clustering was ≈ 29% faster than DBSCAN on the small-size and medium-size Dataset-A to Dataset-D. However, HyperSpec using DBSCAN generated more complete results with 38% shorter runtime than hierarchical clustering on large-scale dataset Dataset-E. The shorter runtime on large-scale

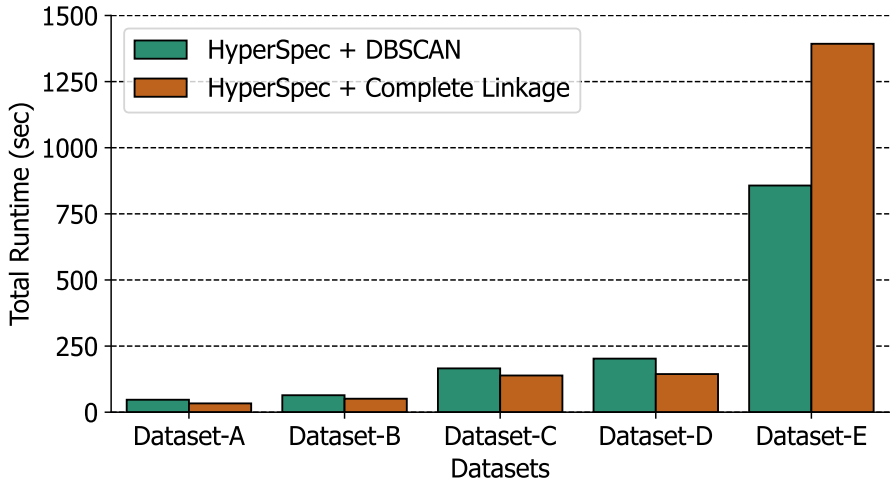


Figure 4.8: Runtime comparison for HyperSpec with DBSCAN and hierarchical clustering with complete linkage on five datasets.

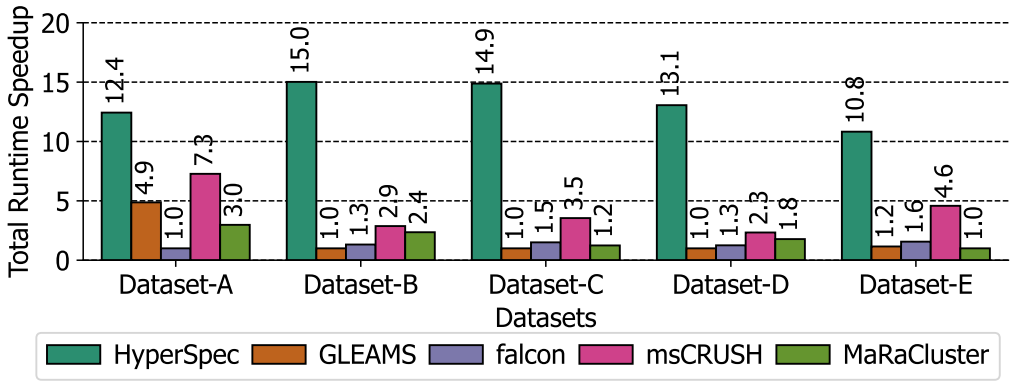


Figure 4.9: Total clustering runtime speedup of HyperSpec compared to alternative clustering tools. The tool with the slowest runtime on each dataset was normalized to 1.

datasets comes from the optimized DBSCAN routines on parallel GPU devices.

We extensively measured the runtime performance of HyperSpec hierarchical clustering with complete linkage compared to three fast spectral clustering tools on five datasets with a varying number of spectra in Table 4.1. falcon and GLEAMS are Python-based libraries that both use optimized JIT compilation and multiprocessing, while msCRUSH and MaRaCluster were written in high-performance C++ and optimized using multithreading. spectra-cluster and MS-Cluster were not considered here since they are significantly (> 5×) slower than other tools. Our evaluation results in Figure 4.9 indicate that HyperSpec consistently outperforms all

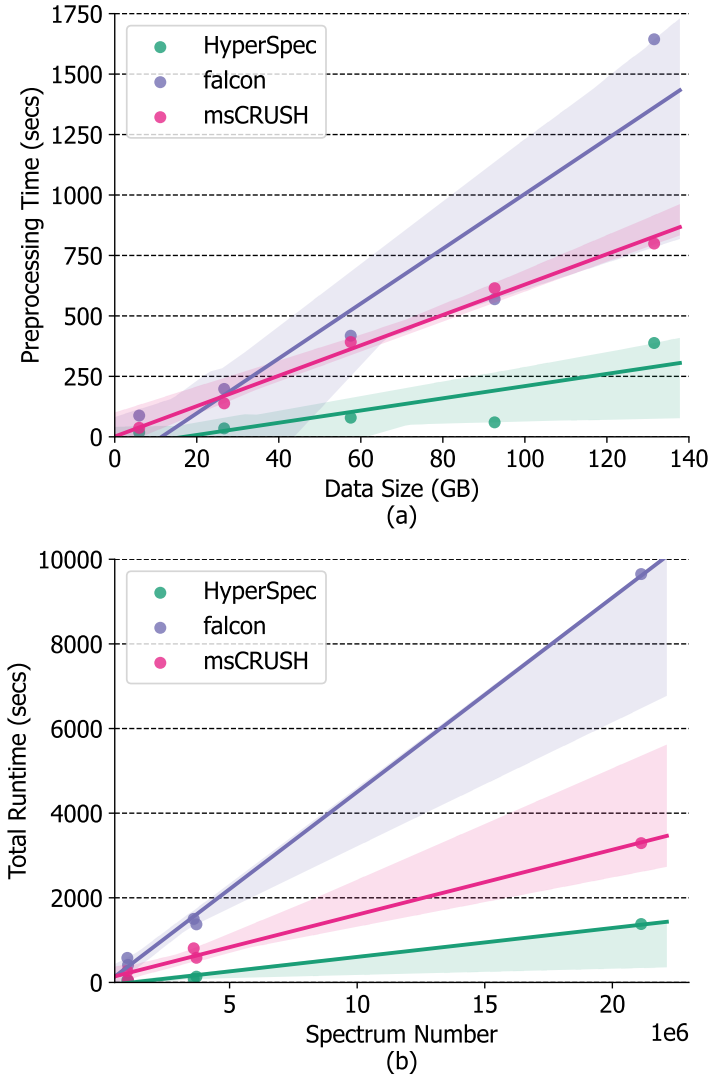


Figure 4.10: Runtime performance of msCRUSH [13], falcon [12], and HyperSpec when scaling to different dataset sizes and number of spectra.

other tools in terms of runtime. 10.8× to 15.0× speedup was observed across different datasets. HyperSpec's speed advantage for spectra preprocessing progressively grows for larger datasets (Figure 4.10-(a)). We further investigated the runtime scalability when processing an increasing number of spectra (Figure 4.10-(b)). Our analysis shows HyperSpec's excellent scalability and performance advantages over alternative tools for increasingly large MS datasets.

We also studied detailed performance metrics (runtime, peak memory consumption, and clustering quality) when running HyperSpec, GLEAMS, falcon, msCRUSH, and MaRaCluster on

the draft human proteome Dataset-E (Table 4.4). All tools were allowed to use all available CPU cores to obtain the fastest clustering speed and were configured to produce a clustering result with a ratio of incorrect clustered spectra around 1.0%. HyperSpec was able to process the full draft human proteome dataset, amounting to 131GB of MS data, in a mere 24 minutes, which is by far the fastest speed among the four spectral clustering tools considered. This runtime is 2.3× faster than the second-fastest tool, msCRUSH, while achieving a higher clustered spectra ratio and smaller peak memory usage. Although GLEAMS produced the highest ratio of correctly clustered spectra, it required 217 minutes of processing time, which is 9.0× slower than HyperSpec. This is because > 90% of GLEAMS' runtime is consumed by the spectra preprocessing and embedding steps. MaRaCluster obtained the lowest ratio of incorrectly clustered spectra among all tools and a comparable completeness value with HyperSpec. Finally, with a peak memory consumption of 54GB, HyperSpec was more memory-efficient than msCRUSH and falcon. To sum up, because HyperSpec achieves an optimal trade-off between clustering quality and runtime efficiency, it is an especially appealing option to process the quickly growing volumes of MS data.

4.3.5 Discussion

HyperSpec is extensible to plug in and support other MS workloads. For example, spectrum preprocessing is a common step during various MS data analysis tasks, such as sequence database searching [142] and spectral library searching [143, 144]. The spectrum preprocessing routines in HyperSpec are highly modularized, so that users can easily integrate these optimized routines into other workloads to take advantage of their efficient implementations.

Another potential application of HyperSpec is to utilize the compact binary HV representation to compress MS data. We have demonstrated that the original spectra in floating-point format can be encoded into binary HVs with D = 1024 to 4096 bits with minimal loss of information to maintain a high-quality clustering quality. In this case, the original spectrum with 50 to 100 peaks in 32-bit or 64-bit floating-point number can be compressed by a factor of 3.1× to 12.5×. Moreover, HV encoding could be convenient for the subsequent downstream

MS workloads, such as spectrum identification. Specifically, off-the-shelf HDC-based pattern matching algorithms [99, 145, 146] could be leveraged to match spectra against a peptide database.

There are still several opportunities to improve upon HyperSpec's clustering quality and runtime performance. Similar to MaRaCluster and spectra-cluster [14, 143], one possible approach could be to derive an optimized distance function to compare spectrum HVs and improve the clustering quality, since finding similar spectra is an essential task during spectral clustering. Another strategy could be to adopt a post-processing scheme after clustering to split up invalid clusters [116, 129]. To further shorten the clustering runtime, the HV distance computations and the clustering step can be parallelized over multiple GPU cards. Because the bucket division mechanism relaxes data dependencies between different buckets of spectra, and the clustering implementations in cuML [130] natively support multiple GPUs, a multi-GPU mode could be integrated in HyperSpec at minimal efforts to achieve a near-linear speedup. The other possible speedup opportunity is combining HyperSpec with the emerging near-storage spectrum processing hardware [147] that can generate higher energy efficiency for repository-scale data processing.

4.4 Conclusion

In this chapter, we present a HDC-based spectral clustering tool, HyperSpec, to achieve both excellent clustering quality and runtime. Instead of clustering raw spectra directly, HyperSpec leverages HDC [124] to convert spectra to hyperdimensional space. Specifically, the spectra are first encoded into binary HVs that have high dimensionality but simpler representation format. Our evaluations show that HyperSpec achieves a comparable clustering quality as state-of-the-art spectral clustering tools [12–14, 16, 143]. Furthermore, we profiled and analyzed the bottlenecks of existing clustering tools. We develop optimized spectra preprocessing routines and an efficient clustering flow by addressing bottleneck components. As a result, HyperSpec is orders of magnitude faster than alternative spectral clustering tools [12–14, 37].

Profiling analysis, as shown in Figure 4.2, identifies the preprocessing step as another significant bottleneck, contributing to considerable overhead in terms of runtime and energy consumption. Although this chapter introduces software optimizations that partially mitigate the issue, preprocessing remains limited by the bandwidth between CPU and solid-state drive (SSD) storage. To address this from a hardware perspective, the next chapter explores the use of near-storage processing (NSP) techniques. Our NSP design offloads the preprocessing step onto the SSD, reducing data redundancy and movement costs, ultimately delivering improved speed and energy efficiency.

This chapter contains material from "HyperSpec: Fast Mass Spectra Clustering in Hyperdimensional Space", by Weihong Xu, Jaeyoung Kang, Wout Bittremieux, Niema Moshiri, and Tajana Rosing, which appears in Journal of Proteome Research, 2023. The dissertation author was the primary investigator and author of this paper.

Chapter 5

Near-storage Acceleration for Preprocessing for Mass Spectrometry

5.1 Introduction

The previous chapter demonstrates that raw data preprocessing is a significant bottleneck in mass spectrometry clustering. To mitigate this overhead, various parallel processing techniques and programming optimizations were proposed. Accelerating the preprocessing step yields substantial speedups across the entire analysis pipeline, including database search [98] and clustering [101]. However, these software-level optimizations fall short of fully exploiting the low-level data parallelism and internal bandwidth available in solid-state drive (SSD) storage.

In this chapter, we introduce a near-storage accelerator designed to enhance data preprocessing in mass spectrometry analysis. A popular approach to obtain spectra data is shotgun proteomics [119], where mass spectrometers analyze samples and acquire millions of fragment spectra in hours. Processing spectra data is the most time-consuming part during experiments [17] since the same molecules may be scanned by mass spectrometer many times, creating many similar and redundant spectra fragments. The high data redundancy severely degrades the efficiency of the MS analysis pipeline, such as open search [134]. Various clustering algorithms and tools [13, 15–17] have been developed to reduce the data redundancy through clustering similar spectra and selecting representatives of each cluster for protein and peptide identification. The clustering step not only decreases the overall analysis time but also improves the identification

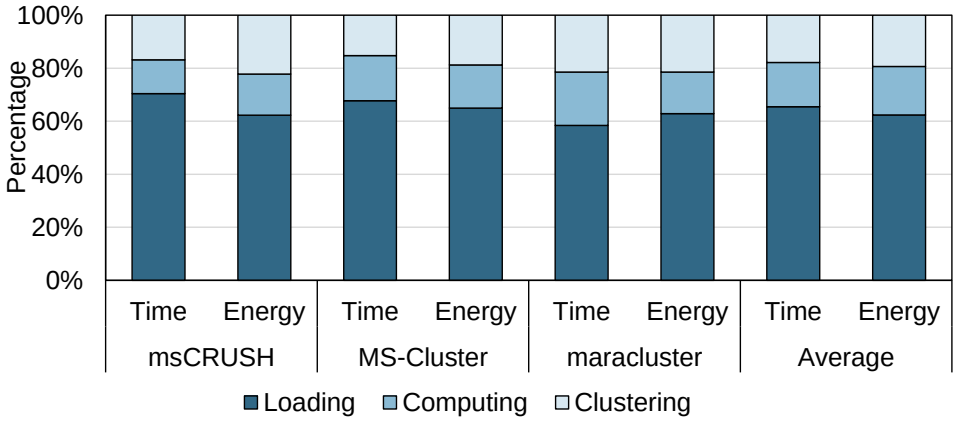


Figure 5.1: Execution time and energy breakdown for various mass spectrum clustering tools, msCRUSH [13], MS-Cluster [16], and *MaRaCluster* [17]. Preprocessing = Loading + Computing.

quality [13, 16].

However, state-of-the-art MS clustering tools are too slow to tackle the exponential growth of MS data. MS-Cluster [16] and spectra-cluster [15] take nearly 30 hours to cluster a dataset with 25M spectra [12], far behind the over-gigabyte hourly data generation speed of modern mass spectrometers [17]. The number of spectra data submission to one of the largest public mass spectra datasets, PRIDE [148], has increased over 10× during the past ten years. The MassIVE [28] database has stored 4×10^9 publicly accessible spectra with over 600 terabytes (TB). Due to the enormous size of these datasets, clustering is done only a few times a year, resulting in less accurate search results. To this end, new tools are essential for both clustering and search of mass spectrometry data to accelerate the identification of proteins, which are critical for the development of new medicines.

Our profiling in Figure 5.1 shows that the spectra preprocessing step (includes loading and computing) is the major bottlenecks that account on average 82% of execution time and energy of the clustering pipeline. The speed of spectra loading and computing is restricted by the costly data movement and limited bandwidth between host memory and storage. MS tools need to fetch the bulky spectra data from storage devices to the host memory before performing the computing step on the CPU. Even with PCIe and NVMe [149] techniques, the peak read speed of

commercial SSD storage [150] (3.2GB/s) is insufficient compared to spectra data over tens of GBs. The other issue is that only a tiny portion of spectra are preserved after preprocessing. The current pipeline still loads the entire unprocessed dataset into memory, leading to unnecessary time and data movement overhead. Unfortunately, existing DRAM-based accelerators, such as MEDAL [81] and RAPID [3], are not suitable for MS preprocessing because they are optimized for reducing data movement cost near DRAM. They are unsuitable to efficiently process spectra and remove the inherent redundancy before fetching data from the storage without dramatically increasing the DRAM capacity and cost. Thus, MS preprocessing should be accelerated in storage as it has the low cost and high capacity needed to prepare the MS data.

In this chapter, we propose a near-storage architecture, MSAS, to accelerate MS preprocessing. The key contributions can be summarized as follows:

- To the best of our knowledge, MSAS is the first near-storage design to boost MS spectra preprocessing. We exploit the internal bandwidth of SSD by conducting a design space exploration at the SSD level and channel level. The results indicate that the channel-level acceleration yields the best hardware and energy efficiency.

- We develop a fully pipelined accelerator with scalable performance for each storage level. We identify the top-k selector as the bottleneck of MSAS and develop an efficient top-k selector based on modified Bitonic algorithms to match the internal bandwidths.

- We extensively compare MSAS with state-of-the-art MS clustering tools on various datasets. We obtain up to 187× speedup on spectra preprocessing tasks. Furthermore, as compared to the state-of-the-art in-storage computing prototype [18], MSAS is up to 1.8× faster. After integrating MSAS into state-of-the-art clustering tools, 2.8× to 11.9× energy efficiency and 3.5× to 9.8× speedup are achieved on clustering workloads.

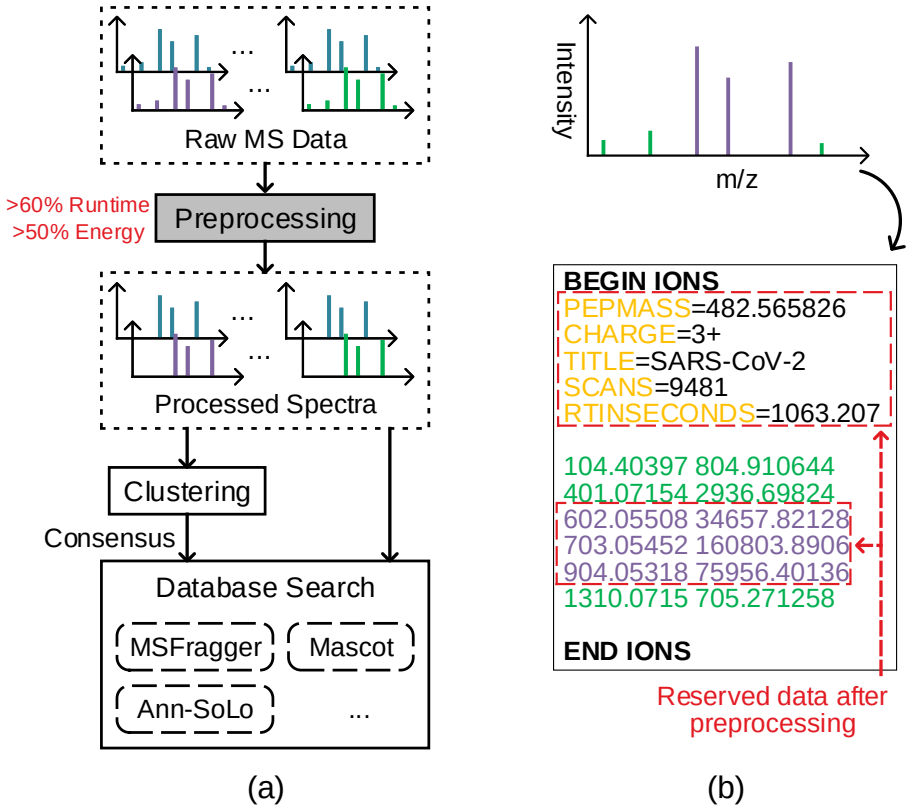


Figure 5.2: (a) Pipeline of data analysis for MS, (b) A spectrum example in MGF format.

5.2 Background

5.2.1 Mass Spectrometry

A wide variety of fields, such as analytical chemistry and large-scale proteomics, have adopted MS as a powerful tool to identify biological structures or chemical compounds. A mass spectrometer is used to generate spectra data containing the mass-to-charge ratio (*m/z*) and ion signal intensity of molecules. As depicted in Figure 5.2-(b), a mass spectrum can be considered as a plot of ion signal intensity and mass-to-charge ratio [151], where the ion strength is expressed as its intensity in y-axis against their *m/z* in the x-axis. Each peak in a spectrum stands for a component with unique *m/z* in the tested sample. Throughout this thesis, we adopt the commonly used Mascot generic format (MGF) [152] as the storage format of spectra. MGF consists of metadata and spectra data, such that metadata record the precursor *m/z*, peptide charge, query

title, and other information. In contrast, the spectra data we use consists of the peak intensity and *m/z* pairs. Figure 5.2-(b) shows a spectrum with total six intensity and *m/z* pairs.

Figure 5.2-(a) illustrates the data analysis pipeline for MS. Researchers can leverage the spectra data to identify molecules and analyze inherent properties through matching the discovered spectra against all the peptides in sequence databases using search engines, like Mascot [152]. Preprocessing step, including filtering, intensity selection, and data normalization, is essential for the subsequent processes since it improves the quality of the results. The filtering step filters the precursor-related peaks and peaks out of the target range, thus reducing noise interference. After the filtration, the intensity selection step finds and preserves the k most intensive peaks. It further reduces the impact of trivial peaks. The typical k value is from 30 to 50 [13, 16]. Data normalization is the final step to lessen the dominant effects of excessive values through additional transformations. The spectra preprocessing is critical for the final analysis quality as demonstrated in [153]. The above three steps are widely used in clustering and search tools [13, 15, 16]. Moreover, the preprocessing compresses spectra size and reduces data redundancy.

5.2.2 Modern SSD

The architecture of modern SSDs [150, 154, 155] is shown in Figure 5.3. The internal SSD is organized into multiple-level hierarchies, such as channels, chips, planes, to name a few. The most frequently used non-volatile storage elements in SSDs are NAND flash memory [156]. The NAND flash chips are organized into 4 to 32 channels, and each channel operates independently and simultaneously [155]. The flash memory controller (FMC) is implemented to perform dedicated data access and error correction for each channel. Several NAND flash chips share one channel, and these NAND flash chips can issue I/O requests independently. Normally 4 to 8 chips are connected to single-channel [154]. Each NAND chip consists of multiple dies, where each die contains multiple planes, blocks, and pages [155]. Page with 4 to 16KB size is the smallest unit that can be accessed and indexed in SSDs. The embedded cores are responsible for

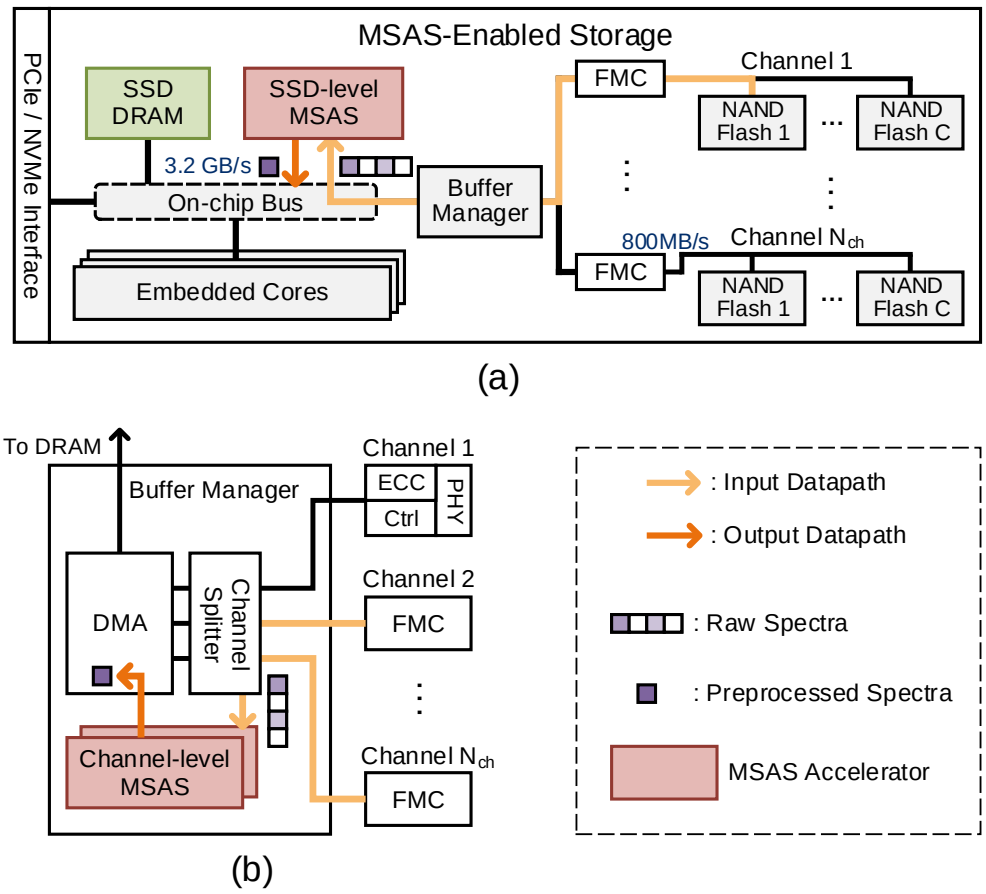


Figure 5.3: Overall diagram of MSAS accelerators embedded in regular SSD, including two types of designs in different storage levels: (a) SSD-level design, (b) Channel-level design in the buffer manager.

SSD control, including issuing I/O commands from the external interface and data scheduling. Modern SSDs normally adopt the high-speed NVMe [149] interface to ensure over GB/s external data rate. However, the internal multiple-level hierarchy provides significantly higher data parallelism [155, 157].

5.3 MSAS Near-storage Architecture

5.3.1 Overview

Figure 5.3 illustrates the overall architecture of MSAS in generic SSDs. MSAS-enabled SSD storage preserves the regular SSD datapath and can additionally boost MS data processing.

The limited external bandwidth of SSD may become the bottleneck for data-intensive workloads. However, the highly parallelized SSD internal architecture provides opportunities to alleviate the bandwidth bottleneck [155]. To this end, we create and evaluate two independent designs, namely SSD-level and channel-level MSAS accelerators, to exploit the internal bandwidth of SSD. Accelerators in different storage levels use scalable hardware configurations to provide sufficient throughput at the cost of reasonable overhead. For the two designs in Figure 5.3, they follow a similar execution flow: the raw spectra are fetched from NAND flash through the input datapath and then computed in MSAS accelerators. The preprocessed spectra results are temporarily cached in the buffers. When the buffers are full or ready, the output datapath transfers processed spectra in buffers through the regular SSD read datapath to host memory. The difference between the two designs is they are exposed to different internal bandwidths and physical address space.

SSD-level Design: Figure 5.3-(a) shows the topmost SSD-level MSAS accelerator that resides in the SSD. The SSD-level accelerator is implemented using CMOS technology on the same die of SSD's embedded cores. It is connected to the global on-chip bus and fetches data from the NAND flashes through the regular SSD datapath. Thus it can access the entire physical address space of back-end NAND flashes and enjoy a peak bandwidth that equals to the external SSD bandwidth (*e.g.* 3.2GB/s [150]). The SSD-level accelerator helps to save over 95% SSD's external bandwidth through the NVMe interface; over 95% of data after preprocessing step are discarded due to the redundancy of spectra data.

Channel-level Design: To improve the overall efficiency, the channel-level accelerator in MSAS aggregates the bandwidth of multiple channels as shown in Figure 5.3-(b). In contrast, DeepStore [154] extends the internal bandwidth by implementing accelerators in each channel. However, this approach has two defects. First, only 800MB/s rate can be exposed to the accelerator in each channel. Second, implementing accelerators in every channel incurs a large hardware overhead. Our channel-level design resides between FMC and DMA engine, where each channel has 800MB/s bandwidth [158]. In generic SSD architecture, the buffer manager is connected

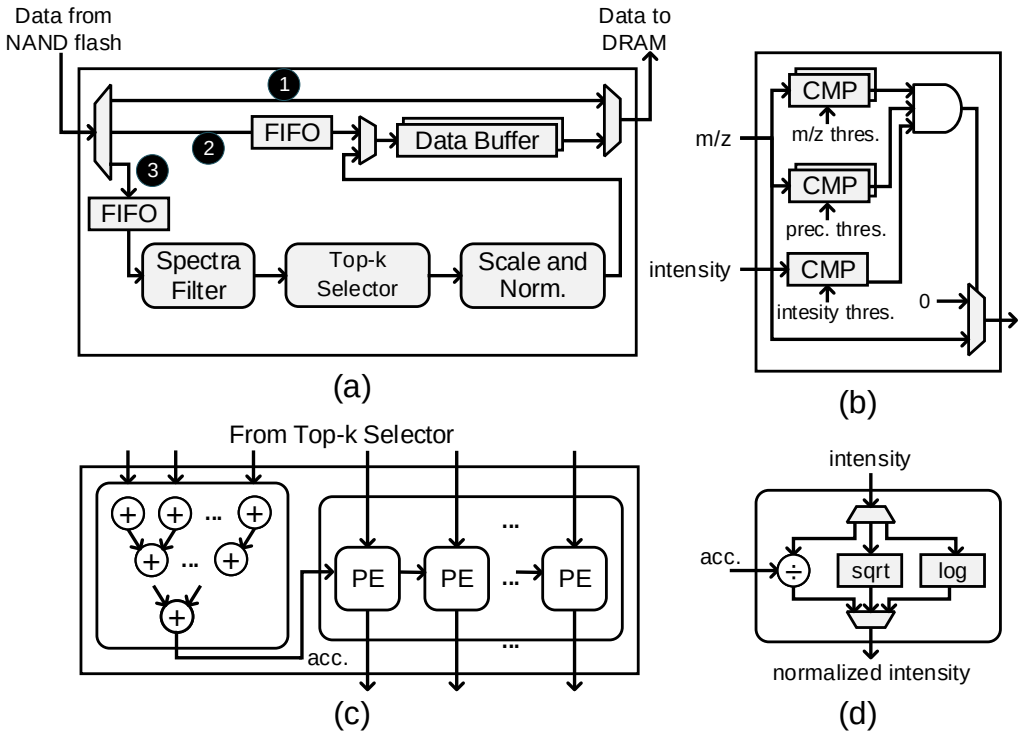


Figure 5.4: (a) Architectures of MSAS accelerator, ❶: regular SSD read datapath, ❷: metadata loading, ❸ datapath for *m/z* and intensity preprocessing, (b) Spectra filter, (c) Scale and normalization module, (d) Processing element (PE).

to all N_{ch} channels, and it is responsible for transporting data from channel bus to DRAM buffer [157]. We add a channel splitter [159] between FMC and DMA engine to multiplex the channel data bus. The channel-level MSAS accelerator is implemented within the buffer manager, and it receives data from the channel splitter and sends processed results to DMA. We can tune the internal bandwidth exposed to the channel-level design by choosing different splitting factors of the channel splitter. Assume each accelerator shares C_{share} channel buses, the highest available bandwidth to each accelerator is $(800 \cdot C_{\text{share}})$MB/s. In this case, total $(N_{\text{ch}}/C_{\text{share}})$ channel-level accelerators need to be implemented. Meanwhile, the physical address space of shared C_{share} channels is accessible for each accelerator. Section 5.4.1 gives the chosen parameter C_{share}.

5.3.2 MSAS Accelerator

Figure 5.4-(a) gives the architecture of MSAS accelerator for performing spectra loading and preprocessing. We use 32-bit single floating-point as the spectra data format processed by MSAS accelerators. The original data reading of SSD is preserved in datapath ❶. Datapath ❷ and datapath ❸ are used for spectra loading and preprocessing. When a new spectrum is coming, the metadata is first cached into the data buffer through datapath ❷. The processing for *m/z* and intensity are performed in datapath ❸ after the metadata loading is finished. Three processing modules, including spectra filter, top-k selector, and scale and normalization module, are implemented over ❸. These modules are executed in a fully pipelined manner to ensure high throughput.

Data Buffer: Multiple data buffers are implemented in each MSAS accelerator, and each data buffer has the same size as SSD's page (8KB). After the preprocessing steps, data buffers work as plane registers to transfer data to SSD DRAM. The page-size data buffers align with SSD's physical addresses since the page is the minimum data chunk that can be indexed. Moreover, we implement double data buffers and interleave the data access of data buffers to avoid clock stalls caused by pipelining.

Spectra Filter: *m/z* and intensity pairs are fed into the spectra filter in Figure 5.4-(b). The spectra filter requires only a single clock cycle to perform a total of five comparisons. The first two comparators determine whether the given *m/z* value is located within the range of *m/z* threshold. The third and fourth comparisons with the precursor threshold are performed to discard those spectra near the precursor peptide. The last comparison is for filtering the spectra with peak intensity less than the intensity threshold. Finally, the *m/z* and intensity pairs that satisfy all the comparison criteria are allowed to pass to the top-k selector.

Top-k Selector: The top-k selector finds the k most intensive peaks of the streaming *m/z* and intensity pairs from the spectra filter. Classic sorting algorithms like Bitonic sort and quick sort can solve the top-k problem by (1) sorting all the data and then (2) selecting the k

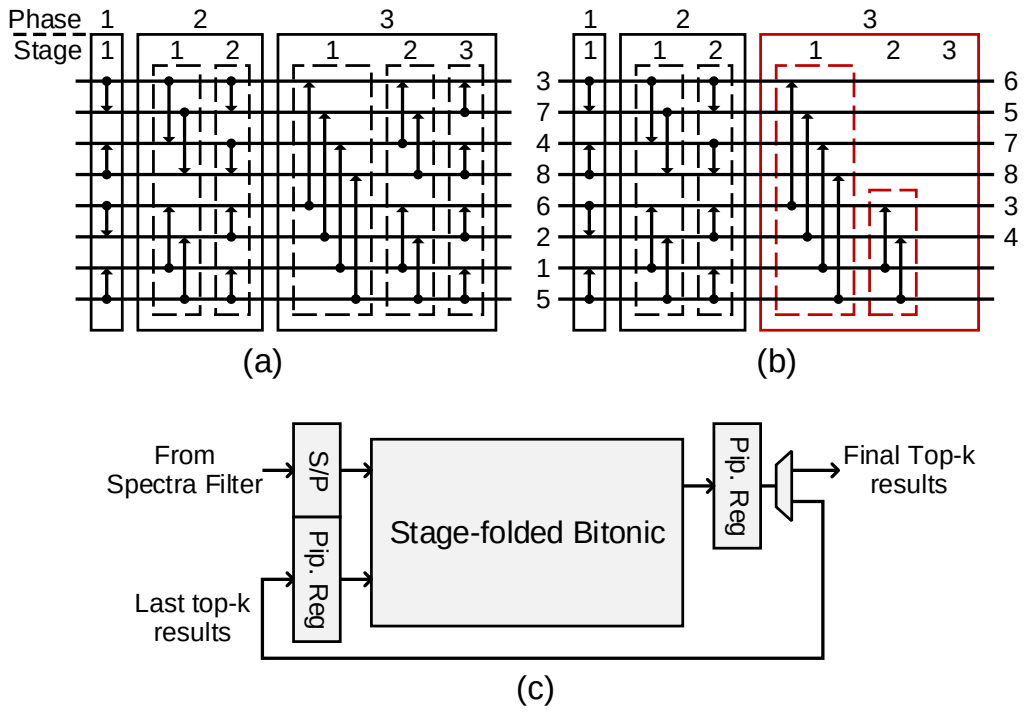


Figure 5.5: (a) Full Bitonic sorting network with $N = 8$, (b) Simplified Bitonic network for Top-k selection ($N = 8$ and $k = 6$), (c) Iterative Top-k selector for streaming data.

largest values. However, the straightforward Bitonic or quick sort is inefficient and superfluous in practical use since the top-k selector does not require strict sorting. Moreover, the number of streaming intensities could be over hundreds, thus finding the top-k values in such data volume would incur huge hardware overhead.

We simplify the original Bitonic algorithms for efficient top-k selection. Figure 5.5-(a) illustrates an example of 8-point Bitonic sorting network composed of $\log_2 N$ phases where the i-th phase contains i stages. The last phase is called *merge phase*. There are $N/2$ compare and swap (CS) in each stage. The required number of CS units is $O(N \log^2 N)$ with $O(\log^2 N)$ latency. We would obtain sorted results after the data pass through the whole network. The original Bitonic network can be simplified to efficiently support top-k selection [160] as shown in Figure 5.5-(b). The basic idea is the remove those CS units in the merge phase that will not impact the top-k results. Assuming a top-k problem $k = 6$ with Bitonic network $N = 8$, the last stage together with the upper part of the second stage in the merge phase can be removed to

obtain correct top-k results. In this case, the latency is reduced from 6 to 5 and the required CS units decrease from 24 to 18.

Using a fully parallel Bitonic network would introduce prohibitive hardware overhead. Instead, we construct a stage-folded iterative Top-k selector in Figure 5.5-(c). A single Bitonic stage is reused, and the top-k results are computed iteratively. Specifically, the generated top-k results will be latched in the output pipeline register. Then k inputs are reserved for the last top-k results while the rest $N - k$ inputs are used to receive new input from the spectra filter. The last top-k results and the new data from the spectra filter are sent to the next round of selection. During the period of top-k computation, the serial to parallel register (S/P) is continuously caching new *m/z* and intensity pairs. The new top-k results are sent to the next computation with new cached data in the S/P register. Considering the typical k value is 30 up to 100 for MS preprocessing [12, 16], we choose $N = 128$ for channel-level and SSD-level accelerators to support these k values. This configuration delivers a peak throughput of 5.8GB/s for $k = 100$.

Scale and Normalization Module: The k most intensive peaks from the top-k selector need to be scaled and normalized to eliminate the dominant effect of large intensity. The scale and normalization module supports three commonly used functions, including log scaling, square root scaling, and unit normalization. For unit normalization, the k peak intensities are scaled to $(0, 1]$ with the accumulation value of peaks. The accumulation of all k intensities are computed in the stage-pipelined adder tree in Figure 5.4-(c). In turn, the normalized values are computed by the divider in the processing elements (PE). The log and square root normalizations are computed by the PE in Figure 5.4-(d).

5.3.3 Data Mapping Scheme in MSAS

The available physical address space varies for different MSAS accelerators. There are two basic requirements for spectra data mapping to maximize bandwidth and hardware utilization. First, they need to be stored in a page-aligned manner. Second, the data of continuous spectrum should be stored in continuous space. For the SSD-level accelerator, the entire SSD address space

Table 5.1: Spectra datasets for evaluation

MS Spectra Datasets					
Class	**Sample Type**	**PRIDE ID**	**# Spectra**	**Avg. Len.**	**Size**
PXD-Tiny	Kidney cell [134]	PXD001468	1.1×10^6	≈181	5.6GB
PXD-Small	Kidney cell [135]	PXD001197	1.1×10^6	≈816	25GB
PXD-Medium	HeLa proteins [137]	PXD003258	4.1×10^6	≈509	54GB
PXD-Large	HEK293 cell [136]	PXD001511	4.2×10^6	≈798	87GB

is accessible. Thus, the generic SSD page allocation scheme can work as the SSD-level data mapping scheme. Each channel-level design is exposed to the address space of continuous C_{share} channels. Thus, the spectra data should be evenly allocated to each channel group composed of C_{share} channels.

5.4 Evaluation

5.4.1 Methodology

Baselines: We evaluate four state-of-the-art spectrum clustering tools, including MS-Cluster [16], spectra-cluster [15], MaRaCluster [17] and msCRUSH [13]. The fragment mass tolerance and precursor mass tolerance are set to 0.05 Da and 20 ppm, respectively. The most 50 intensive peaks are preserved. We filter those peaks whose *m/z* is out of range 200 to 2000. MS-Cluster is set to run three rounds of clustering and uses the integrated LTQ_TRYP model. spectra-cluster is tested under its *fast mode*. The other options and parameters are set to default. On top of the comparison to the CPU-based implementation, we show the performance improvement of MSAS over the state-of-the-art in-storage computing solution INSIDER [18].

Spectra Datasets: Table 5.1 summarizes the used datasets at different data scales and average lengths. We classify the datasets into four categories based on their size. The data are publicly available and downloaded from the PRIDE repository [161]. All raw data are converted into MGF format using ThermoRawFileParser [139] with release version 1.3.4.

Area and Power Modeling: The baseline MS clustering tools were evaluated on a server with Intel Xeon E5-2680 CPU, 64GB DDR4-2133MHz memory, and a 2TB commercial

Table 5.2: MSAS implementation and area breakdown

Design	**SSD-Level**	**Channel-Level**
Frequency	800MHz	
Bitonic Top-K	$N = 128$ (0.09mm^2)	
Norm. Scale	0.27mm^2	0.27mm^2
FIFO	32b×64	
Buffer	256KB (0.36mm^2)	64KB (0.09mm^2)
Area	0.72mm^2	0.45mm^2
Number	1	4
Total Area	0.72mm^2	1.81mm^2
Average Power	2.22W	8.06W

PCIe-based SSD. The maximum read speed is 3.2GB/s. The energy consumption of CPU baselines is measured using CPU Energy Meter [162]. MSAS accelerators are implemented using *Verilog* and synthesized by *Synopsys Design Compiler* on TSMC 28nm process. The clock frequency is set to 800MHz. The area and energy consumption of FIFO and buffer are estimated using CACTI [77].

MSAS is evaluated on 1TB Intel DC P4500 SSD, providing a sequential read bandwidth of 3.2GB/s and sequential write bandwidth of 600MB/s [150]. The active power under sequential access mode is 11W and 9.6W for write and read, respectively. We assume each flash array has a read latency of 64us, 16 channels, 4 flash chips per channel, 2 dies per chip, 2 planes per die, 512 blocks per plane, and 1024 pages per block. Each flash page is 8KB, and each channel using ONFI [158] has a bandwidth of 800MB/s. We combine performance data obtained via SSD simulation with the SSD power model in [163] to estimate the energy consumption. The configurations and area breakdown of MSAS accelerator in three storage levels are summarized in Table 5.2. We let each channel-level accelerator share $C_{\text{share}} = 4$ channels, requiring $16/4 = 4$ channel-level accelerators. The flash-level design contains a total of $16 \times 4 = 64$ accelerators.

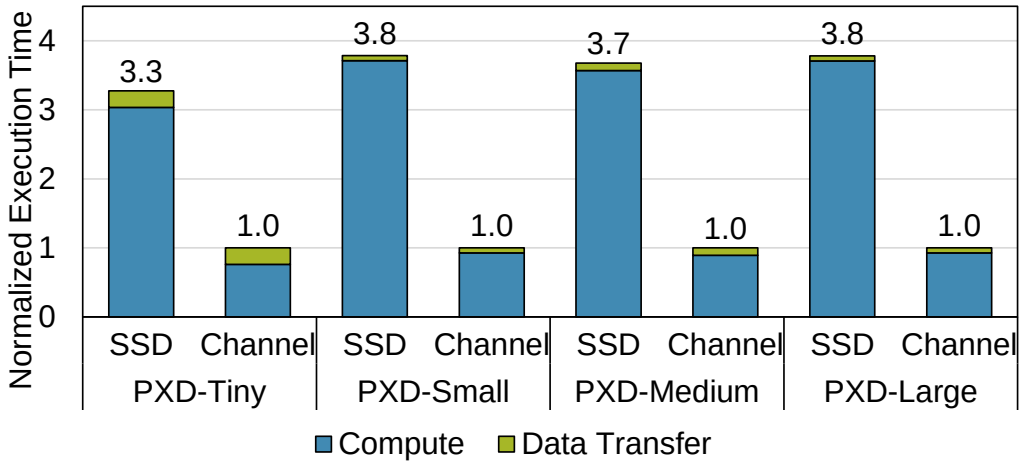


Figure 5.6: Execution time comparison for SSD-level and channel-level designs.

5.4.2 Performance and Energy Evaluation

Comparison of SSD and channel-level designs. We first compare the preprocessing speed of MSAS accelerators at different storage levels. The configurations of SSD-level and channel-level designs are shown in Table 5.2. The spectra data are pre-stored in SSD before the preprocessing starts. Figure 5.6 illustrates the execution time on four MS datasets, where the channel-level execution time is normalized to 1. Channel-level design is 3.3× to 3.8× faster vs. SSD-level design. The gain comes from two aspects. First, spectra preprocessing is data-intensive rather than computation-intensive workload. In this case, higher bandwidth brings about more significant speedup. Second, each channel-level accelerator is connected to 4 channel buses and enjoys 3.2GB/s bandwidth. As discussed in Section 5.3.2, each fully pipelined channel-level accelerator can fulfill 4.8GB/s data rate. The four channel-level accelerators are able to fully exploit the aggregate 12.8GB/s internal bandwidth of 16 channels. In comparison, the performance of SSD-level design is restricted by the limited 3.2GB/s bandwidth of the on-chip bus.

Spectra preprocessing speedup. In Figure 5.7, we compare the performance of channel-level design and the in-storage computing prototype, INSIDER [18], over CPU on spectra preprocessing workload. We use msCRUSH as the CPU baselines as it yields the fastest speed.

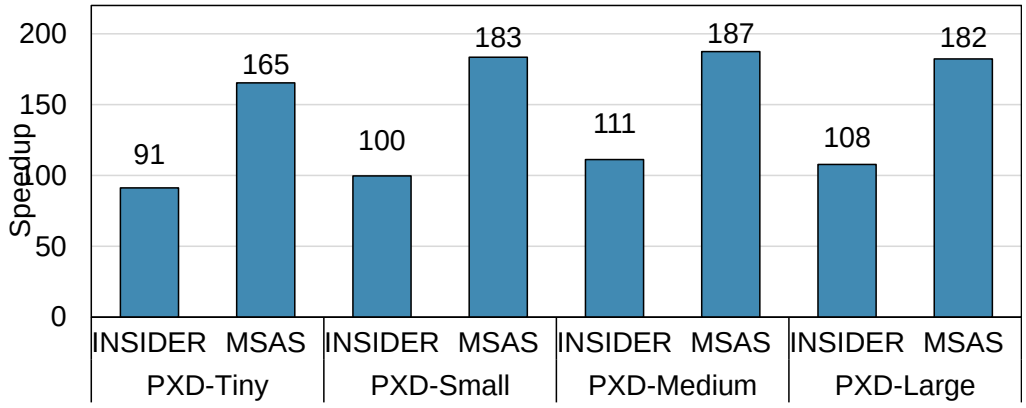


Figure 5.7: Preprocessing speedup of INSIDER [18] and MSAS over CPU baselines.

INSIDER uses 8-lane PCIe, which delivers 8GB/s peak bandwidth, and the spectra preprocessing is computed using FPGA in SSD. The channel-level MSAS achieves 165× to 187× speedup over CPU. Moreover, MSAS is 1.7× to 1.8× faster than INSIDER. The speedup is derived from two aspects. First, INSIDER needs to load raw spectrum data from SSD into FPGA and then transfer processed data back to SSD, incurring redundant data movement. MSAS only fetches data from NAND flash once. Second, the internal bandwidth of MSAS is 60% higher than INSIDER's PCIe bandwidth.

System-level improvements after integration. We integrate the channel-level design into the existing MS clustering tools (msCRUSH, MS-Cluster, and MaRaCluster) by replacing the preprocessing part with MSAS framework. This offloads the preprocessing into MSAS-enabled SSD. The clustering part is executed on the CPU. For CPU baselines, the entire preprocessing and clustering process is computed by CPU. Figure 5.8 gives the speedup and energy efficiency of MSAS over CPU baselines after integration. The channel-level MSAS achieves 3.5× to 9.8× speedup and 2.8× to 11.9× energy efficiency on four datasets. The preprocessing steps are dominant the runtime and energy of clustering workloads, as shown in Figure 5.1. Accelerating preprocessing provides significant overall speedup for clustering tools.

Comparing the performance gain within each clustering tool, we observe that the gain is more significant on datasets with longer average spectrum length (*e.g.* PXD-Small and

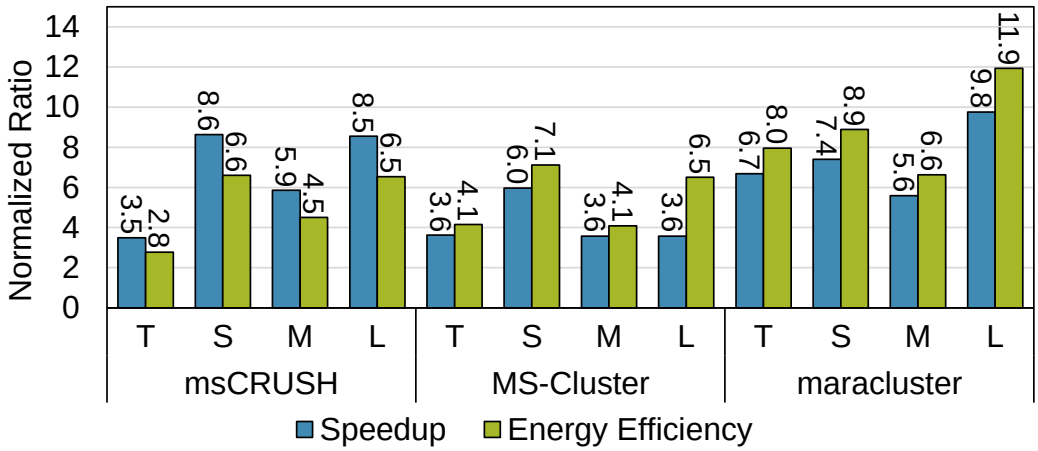


Figure 5.8: Speedup and energy efficiency over CPU after integrating MSAS into clustering tools.

PXD-Large). This is due to the fact that the average spectra length of PXD-Tiny is close to the top-k value ($k = 50$). The data size of preprocessed spectra is not greatly reduced compared to the raw spectra, leading to 3.5× to 6.7× with 2.8× to 8.0× lower energy consumption. The difference in performance gain between the three clustering tools is mainly benefitted from the code optimization.

5.4.3 Overhead Analysis

The channel-level design consumes the largest area among the proposed designs. Considering that the TLC NAND flash chip [156] has a die size over 100mm^2, the largest channel-level design only incurs 0.03% area overhead, which is negligible. SSD's 50W power budget [154] is more than sufficient for the added MSAS accelerators. The channel-level design, with the highest 8.06W power dissipation, meets the power supply constraints.

5.5 Conclusion

Chapter 5 presents MSAS, the near-storage computing framework that efficiently accelerates the mass spectrum data preprocessing. Based on the observation that preprocessing takes nearly 82% of the overall execution time and energy consumption for MS analysis, MSAS tackles

the challenge by performing the preprocessing in SSD. We present two types of accelerator designs to exploit the internal storage bandwidth at different levels of the storage hierarchy. Then we design scalable and energy-efficient accelerators to satisfy the data rate for each storage level. The channel-level MSAS generates the best efficiency with 1.81mm^2 area and 8.06W power. The experiments show that the channel-level MSAS is able to boost spectra preprocessing by up to 187× compared to the fastest MS analysis tool. Moreover, MSAS reduces the execution time up to 1.8× compared to in-storage computing prototype, INSIDER [18]. We show that the proposed solution can improve the clustering speed and energy efficiency of the overall MS clustering pipeline by 3.5× to 9.8× and 2.8× to 11.9×, respectively.

This chapter contains material from "A Near-Storage Framework for Boosted Data Preprocessing of Mass Spectrum Clustering", by Weihong Xu, Jaeyoung Kang, and Tajana Rosing, which appears in IEEE/ACM Design Automation Conference (DAC), 2022. The dissertation author was the primary investigator and author of this paper.

Chapter 6

Summary and Future Work

6.1 Thesis Summary

Over the past few decades, the fields of genomics and proteomics have undergone unprecedented expansion, driven by technological advancements in high-throughput sequencing and mass spectrometry (MS). This rapid growth has not only increased our understanding of biological processes at the molecular level but also transformed these disciplines into data-intensive sciences, where the sheer volume and complexity of the generated data present significant challenges in analysis and interpretation. This necessitates the development of high-performance software and hardware designs to handle increasingly large and complex datasets. This thesis systematically explores the benefits of software and hardware optimizations. At the software level, we develop more optimal kernels on CPU and GPU to mitigate major bottlenecks in data analysis. At the hardware level we use PIM and near-storage processing (NSP) to design data-centric hardware for the computation-intensive and IO-bound tasks, respectively. These designs together provide scalable and efficient solutions that pave the way for more effective data analysis for genomics and proteomics.

Chapter 2 and Chapter 3 discuss how to develop efficient hardware and software solutions for genome similarity computation aim for different application scenarios. For the accurate and computation-intensive genome alignemnt, Chapter 2 presents a novel Processing-in-Memory (PIM) accelerator, RAPIDx, which significantly enhances the performance. By employing a

parallelized dynamic programming (DP) algorithm with difference representation, RAPIDx reduces the required data width from 32-bit to 5-bit integers, thereby achieving near-linear complexity with minimal accuracy degradation. The ReRAM-based PIM architecture exploits four levels of data parallelism to implement this optimized algorithm efficiently. Our evaluation demonstrates that RAPIDx offers 131.1× and 46.8× better throughput for short-read alignment compared to CPU [8] and GPU [7] baselines, respectively. For long-read alignment, RAPIDx achieves 2.9× and 9.3× throughput improvements over state-of-the-art ASIC [10] and PIM [3–5] accelerators.

RAPIDx is accurate but not well-suited for latency-sensitive tasks and memory-constrained hardware. To address these limitations, Chapter 3 presents a memory-efficient tool, HyperGen, for genome similarity estimation. HyperGen is a genome sketching tool based on hyperdimensional computing (HDC) [104, 106] to enhance the accuracy, runtime performance, and memory efficiency in large-scale genomic analysis. By avoiding the costly alignment process and utilizing a combination of FracMinHash sampling and hyperdimensional vector encoding, HyperGen achieves superior accuracy in estimating Average Nucleotide Identity (ANI) while maintaining compact sketch file sizes. The tool, implemented in Rust, outperforms existing sketch-based tools [33–36] in both speed and memory efficiency.

Chapter 4 extends the application of HDC to MS clustering. HyperSpec, a HDC-based spectral clustering tool is presented to achieve both high clustering quality and runtime efficiency. HyperSpec encodes spectra into binary hypervectors, which simplifies their representation while maintaining high dimensionality. Our optimized preprocessing routines and efficient clustering flow enable HyperSpec to achieve the fastest clustering speed among all tools evaluated, outperforming existing spectral clustering tools [12–14, 37] by orders of magnitude in terms of runtime while maintaining comparable clustering quality.

To mitigate the significant overhead caused by MS data preprocessing, Chapter 5 presents MSAS, a near-storage computing framework aimed at accelerating the preprocessing of MS data. Given that preprocessing accounts for nearly 82% of the total execution time and energy

consumption in MS analysis, MSAS addresses this bottleneck by performing the preprocessing directly within the SSD. We develop scalable and energy-efficient accelerators that leverage internal storage bandwidth at various levels of the storage hierarchy. Our results show that the channel-level MSAS delivers up to 187× speedup in spectra preprocessing compared to the fastest MS analysis tools [13] and reduces execution time by 1.8× compared to the INSIDER in-storage computing prototype [18]. When integrated into existing MS clustering pipelines, MSAS enhances clustering speed by 3.5× to 9.8× and improves energy efficiency by 2.8× to 11.9×.

6.2 Future Work

The research presented in this thesis opens up several avenues for further exploration and enhancement in the domains of genomic and proteomic data analysis. These future work directions have the potential to enable more efficient, scalable, and accurate analysis.

The next step for RAPIDx in Chapter 2 is to develop comprehensive host-system integration. As a PIM-based domain-specific accelerator, RAPIDx functions as a co-processor for genome sequence alignment, with the potential to be integrated into existing computing systems with minimal hardware modifications. Future studies could explore optimizing data transfer mechanisms between RAPIDx and the host, particularly to minimize processing latency. Additionally, RAPIDx's capability to support flexible scoring functions without architectural modifications opens up the possibility of extending its application to a broader range of genomic alignment tasks. Exploring various scoring functions under different application scenarios could further enhance RAPIDx's versatility and performance.

The future work for HyperGen in Chapter 3 should focus on optimizing the vector representation of sketch hypervectors (HVs) for even better memory efficiency and faster large-scale genomic searches. Techniques such as lossy vector compression, including product quantization and residual quantization, could be employed to further reduce sketch size and memory footprint,

making HyperGen more suitable for embedded and mobile devices. Additionally, accelerating the search step, which involves intensive General Matrix Multiply (GEMM) operations, by leveraging advanced hardware architectures with high data parallelism, could yield significant performance gains. Extending HyperGen to support a wider range of genomic workloads, such as metagenome analysis and containment analysis, could also broaden its applicability in bioinformatics.

Several future research directions could be pursued to enhance both clustering quality and runtime performance for HyperSpec in Chapter 4 and MSAS in Chapter 5. HyperSpec should be integrated with the near-storage MSAS preprocessing hardware, which would provide higher energy efficiency and performance, particularly for repository-scale data processing. Furthermore, HyperSpec's runtime could be shortened by parallelizing HV distance computations and the clustering step across multiple GPU cards, potentially achieving near-linear speedup with minimal effort. Lastly, expanding HyperSpec to support other MS workloads, such as sequence database searching and spectral library searching, presents a promising direction. The modular design of HyperSpec's spectrum preprocessing routines allows for easy integration into other workflows, potentially benefiting a wider range of MS data analysis tasks. Additionally, the compact binary HV representation developed in HyperSpec could be leveraged to compress MS data, with significant reductions in data size while maintaining high clustering quality. Exploring the use of HDC-based pattern matching algorithms for spectrum identification could further enhance the utility of HyperSpec in downstream MS applications.

Bibliography

[1] Dna sequencing costs: Data from the nhgri genome sequencing program (gsp). www.genome.gov/sequencingcostsdata, 2022.

[2] Genbank and wgs statistics. https://www.ncbi.nlm.nih.gov/genbank/statistics/, 2023.

[3] Saransh Gupta, Mohsen Imani, Behnam Khaleghi, Venkatesh Kumar, and Tajana Rosing. Rapid: A ReRAM processing in-memory architecture for DNA sequence alignment. In *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2019.

[4] Shaahin Angizi, Jiao Sun, Wei Zhang, and Deliang Fan. Aligns: A processing-in-memory accelerator for dna short read alignment leveraging sot-mram. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[5] Farzaneh Zokaee, Hamid R Zarandi, and Lei Jiang. AligneR: A process-in-memory architecture for short read alignment in rerams. *IEEE Computer Architecture Letters*, 17(2):237–240, 2018.

[6] Shaahin Angizi, Jiao Sun, Wei Zhang, and Deliang Fan. Pim-aligner: A processing-in-mram platform for biological sequence alignment. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1265–1270, 2020.

[7] Nauman Ahmed, Jonathan Lévy, Shanshan Ren, Hamid Mushtaq, Koen Bertels, and Zaid Al-Ars. Gasal2: a gpu accelerated sequence alignment library for high-throughput ngs data. *BMC Bioinformatics*, 20:1–20, 2019.

[8] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018.

[9] Damla Senol Cali, Gurpreet S. Kalsi, Zülal Bingöl, Can Firtina, Lavanya Subramanian, Jeremie S. Kim, Rachata Ausavarungnirun, Mohammed Alser, Juan Gomez-Luna, Amirali Boroumand, Anant Norion, Allison Scibisz, Sreenivas Subramoneyon, Can Alkan, Saugata Ghose, and Onur Mutlu. GenASM: A high-performance, low-power approximate string matching acceleration framework for genome sequence analysis. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 951–966, 2020.

[10] Yi-Lun Liao, Yu-Cheng Li, Nae-Chyun Chen, and Yi-Chang Lu. Adaptively banded smith-waterman algorithm for long reads and its hardware accelerator. In *IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 1–9, 2018.

[11] Martin Šošić and Mile Šikić. Edlib: a c/c++ library for fast, exact sequence alignment using edit distance. *Bioinformatics*, 33(9):1394–1395, 2017.

[12] Wout Bittremieux, Kris Laukens, William Stafford Noble, and Pieter C Dorrestein. Large-scale tandem mass spectrum clustering using fast nearest neighbor searching. *Rapid Communications in Mass Spectrometry*, page e9153, 2021.

[13] Lei Wang, Sujun Li, and Haixu Tang. mscrush: fast tandem mass spectral clustering using locality sensitive hashing. *Journal of Proteome Research*, 18(1):147–158, 2018.

[14] Matthew The and Lukas Käll. Maracluster: A fragment rarity metric for clustering fragment spectra in shotgun proteomics. *Journal of Proteome Research*, 15(3):713–720, 2016.

[15] Johannes Griss, Yasset Perez-Riverol, Steve Lewis, David L Tabb, José A Dianes, Noemi Del-Toro, Marc Rurik, Mathias Walzer, Oliver Kohlbacher, Henning Hermjakob, Rui Wang, and Juan Antonio Vizcaíno. Recognizing millions of consistently unidentified spectra across hundreds of shotgun proteomics datasets. *Nature Methods*, 13(8):651–656, 2016.

[16] Ari M Frank, Nuno Bandeira, Zhouxin Shen, Stephen Tanner, Steven P Briggs, Richard D Smith, and Pavel A Pevzner. Clustering millions of tandem mass spectra. *Journal of Proteome Research*, 7(01):113–122, 2008.

[17] Matthew The and Lukas Kall. Maracluster: A fragment rarity metric for clustering fragment spectra in shotgun proteomics. *Journal of Proteome Research*, 15(3):713–720, 2016.

[18] Zhenyuan Ruan, Tong He, and Jason Cong. INSIDER: Designing in-storage computing system for emerging high-performance drive. In *USENIX Annual Technical Conference (ATC)*, pages 379–394, 2019.

[19] Devon M Cayer, Kristopher L Nazor, and Nicholas J Schork. Mission critical: the need for proteomics in the era of next-generation sequencing and precision medicine. *Human Molecular Genetics*, 25(R2):R182–R189, 2016.

[20] Carlos D Bustamante, Francisco M De La Vega, and Esteban G Burchard. Genomics for the world. *Nature*, 475(7355):163–165, 2011.

[21] Benjamin S Frey, Deidre E Damon, and Abraham K Badu-Tawiah. Emerging trends in paper spray mass spectrometry: Microsampling, storage, direct analysis, and applications. *Mass Spectrometry Reviews*, 39(4):336–370, 2020.

[22] Ruedi Aebersold and Matthias Mann. Mass spectrometry-based proteomics. *Nature*, 422(6928):198–207, 2003.

[23] Zainab Noor, Seong Beom Ahn, Mark S Baker, Shoba Ranganathan, and Abidali Mohamedali. Mass spectrometry–based protein identification in proteomics—a review. *Briefings in Bioinformatics*, 22(2):1620–1638, 2021.

[24] Illumina sequencing platforms. https://www.illumina.com/systems/sequencing-platforms.html, 2023.

[25] Wendy Weijia Soon, Manoj Hariharan, and Michael P Snyder. High-throughput sequencing for biology and medicine. *Molecular Systems Biology*, 9(1):640, 2013.

[26] Jay Shendure and Hanlee Ji. Next-generation dna sequencing. *Nature Biotechnology*, 26(10):1135–1145, 2008.

[27] Bonnie Berger and Yun William Yu. Navigating bottlenecks and trade-offs in genomic data analysis. *Nature Reviews Genetics*, 24(4):235–250, 2023.

[28] UCSD. MassIVE: Mass Spectrometry Interactive Virtual Environment. https://massive.ucsd.edu/, 2022.

[29] Muaaz Gul Awan and Fahad Saeed. Ms-reduce: an ultrafast technique for reduction of big mass spectrometry data for high-throughput processing. *Bioinformatics*, 32(10):1518–1526, 2016.

[30] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[31] Temple F Smith and Michael S Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[32] Yatish Turakhia, Gill Bejerano, and William J Dally. Darwin: A genomics co-processor provides up to 15,000× acceleration on long read assembly. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 199–213, 2018.

[33] Brian D Ondov, Todd J Treangen, Páll Melsted, Adam B Mallonee, Nicholas H Bergman, Sergey Koren, and Adam M Phillippy. Mash: fast genome and metagenome distance estimation using minhash. *Genome Biology*, 17(1):1–14, 2016.

[34] Daniel N Baker and Ben Langmead. Genomic sketching with multiplicities and locality-sensitive hashing using dashing 2. *Genome Research*, 33(7):1218–1227, 2023.

[35] XiaoFei Zhao. Bindash, software for fast genome distance estimation on a typical personal laptop. *Bioinformatics*, 35(4):671–673, 2019.

[36] C Titus Brown and Luiz Irber. sourmash: a library for MinHash sketching of DNA. *Journal of Open Source Software*, 1(5):27, 2016.

[37] Wout Bittremieux, Damon H May, Jeffrey Bilmes, and William Stafford Noble. A learned embedding for efficient joint analysis of millions of mass spectra. *Nature Methods*, 19(6):675–678, 2022.

[38] Aaron M. Wenger, Paul Peluso, William J. Rowell, Pi-Chuan Chang, Richard J. Hall, Gregory T. Concepcion, Jana Ebler, Arkarachai Fungtammasan, Alexey Kolesnikov, Nathan D. Olson, Armin Töpfer, Michael Alonge, Medhat Mahmoud, Yufeng Qian, Chen-Shan Chin, Adam M. Phillippy, Michael C. Schatz, Gene Myers, Mark A. DePristo, Jue Ruan, Tobias Marschall, Fritz J. Sedlazeck, Justin M. Zook, Heng Li, Sergey Koren, Andrew Carroll, David R. Rank, and Michael W. Hunkapiller. Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. *Nature Biotechnology*, 37(10):1155–1162, 2019.

[39] Heng Li and Richard Durbin. Fast and accurate long-read alignment with burrows–wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

[40] M. Gokhale, B. Holmes, and K. Iobst. Processing in memory: the terasys massively parallel pim array. *Computer*, 28(4):23–31, 1995.

[41] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. In *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, pages 336–348, 2015.

[42] Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2016.

[43] Saransh Gupta, Mohsen Imani, Harveen Kaur, and Tajana Rosing. Nnpim: A processing in-memory architecture for neural network acceleration. *IEEE Transactions on Computers*, 68(9):1325–1337, 2019.

[44] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature Methods*, 9(4):357–359, 2012.

[45] Edans Flavius de Oliveira Sandes, Guillermo Miranda, Xavier Martorell, Eduard Ayguade, George Teodoro, and Alba Cristina Magalhaes Melo. CUDAlign 4.0: Incremental speculative traceback for exact chromosome-wide alignment in GPU clusters. *IEEE Transactions on Parallel and Distributed Systems*, 27(10):2838–2850, 2016.

[46] James Arram, Thomas Kaplan, Wayne Luk, and Peiyong Jiang. Leveraging fpgas for accelerating short read alignment. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 14(3):668–677, 2016.

[47] Roman Kaplan, Leonid Yavits, Ran Ginosar, and Uri Weiser. A resistive CAM processing-in-storage architecture for DNA sequence alignment. *IEEE Micro*, 37(4):20–28, 2017.

[48] Roman Kaplan, Leonid Yavits, and Ran Ginosasr. BioSEAL: In-memory biological sequence alignment accelerator for large-scale genomic data. In *ACM International Systems and Storage Conference*, pages 36–48, 2020.

[49] Wenqin Huangfu, Shuangchen Li, Xing Hu, and Yuan Xie. Radar: A 3D-ReRAM based DNA alignment accelerator architecture. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2018.

[50] Kevin Liu, Serita Nelesen, Sindhu Raghavan, C Randal Linder, and Tandy Warnow. Barking up the wrong treelength: the impact of gap penalty on alignment and tree accuracy. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(1):7–21, 2008.

[51] Daichi Fujiki, Shunhao Wu, Nathan Ozog, Kush Goliya, David Blaauw, Satish Narayanasamy, and Reetuparna Das. SeedEx: A genome sequencing accelerator for optimal alignments in subminimal space. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 937–950, 2020.

[52] Kun-Mao Chao, William R Pearson, and Webb Miller. Aligning two sequences within a specified diagonal band. *Bioinformatics*, 8(5):481–487, 1992.

[53] Saransh Gupta, Mohsen Imani, and Tajana Rosing. Felix: Fast and energy-efficient logic in memory. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7, 2018.

[54] Michael Burrows and David Wheeler. A block-sorting lossless data compression algorithm. In *Digital SRC Research Report*, 1994.

[55] Gene Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM (JACM)*, 46(3):395–415, 1999.

[56] Hajime Suzuki and Masahiro Kasahara. Introducing difference recurrence relations for faster semi-global alignment of long sequences. *Bioinformatics*, 19(1):33–47, 2018.

[57] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[58] David J Lipman and William R Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, 1985.

[59] Subho Sankar Banerjee, Mohamed El-Hadedy, Jong Bin Lim, Zbigniew T Kalbarczyk, Deming Chen, Steven S Lumetta, and Ravishankar K Iyer. Asap: Accelerated short-read alignment on programmable hardware. *IEEE Transactions on Computers*, 68(3):331–346, 2018.

[60] Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, 1982.

[61] Kyeongho Lee, Jinho Jeong, Sungsoo Cheon, Woong Choi, and Jongsun Park. Bit parallel 6T SRAM in-memory computing with reconfigurable bit-precision. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[62] Jalil Boukhobza, Stéphane Rubini, Renhai Chen, and Zili Shao. Emerging nvm: A survey on architectural integration and research challenges. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 23(2):1–32, 2017.

[63] Dayane Reis, Michael Niemier, and X Sharon Hu. Computing in memory with FeFETs. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2018.

[64] Minsu Kim, Muqing Liu, Luke R Everson, and Chris H Kim. An embedded NAND flash-based compute-in-memory array demonstrated in a standard logic process. *IEEE Journal of Solid-State Circuits (JSSC)*, 57(2):625–638, 2021.

[65] Cheng-Xin Xue, Je-Min Hung, Hui-Yao Kao, Yen-Hsiang Huang, Sheng-Po Huang, Fu-Chun Chang, Peng Chen, Ta-Wei Liu, Chuan-Jia Jhang, Chin-I Su, Win-San Khwa, Chung-Chuan Lo, Ren-Shuo Liu, Chih-Cheng Hsieh, Kea-Tiong Tang, Yu-Der Chih, Tsung-Yung Jonathan Chang, and Meng-Fan Chang. A 22nm 4Mb 8b-precision ReRAM computing-in-memory macro with 11.91 to 195.7 TOPS/W for tiny AI edge devices. In *IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 245–247, 2021.

[66] Nishil Talati, Saransh Gupta, Pravin Mane, and Shahar Kvatinsky. Logic design within memristive memories using memristor-aided loGIC (MAGIC). *IEEE Transactions on Nanotechnology*, 15(4):635–650, 2016.

[67] Julien Borghetti, Gregory S Snider, Philip J Kuekes, J Joshua Yang, Duncan R Stewart, and R Stanley Williams. 'memristive'switches enable 'stateful'logic operations via material implication. *Nature*, 464(7290):873–876, 2010.

[68] Byung Chul Jang, Yunyong Nam, Beom Jun Koo, Junhwan Choi, Sung Gap Im, Sang-Hee Ko Park, and Sung-Yool Choi. Memristive logic-in-memory integrated circuits for energy-efficient flexible electronics. *Advanced Functional Materials*, 28(2):1704725, 2018.

[69] Shahar Kvatinsky, Misbah Ramadan, Eby G Friedman, and Avinoam Kolodny. VTEAM: A general model for voltage-controlled memristors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(8):786–790, 2015.

[70] Ameer Haj-Ali, Rotem Ben-Hur, Nimrod Wald, and Shahar Kvatinsky. Efficient algorithms for in-memory fixed point multiplication using magic. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2018.

[71] Shahar Kvatinsky, Dmitry Belousov, Slavik Liman, Guy Satat, Nimrod Wald, Eby G Friedman, Avinoam Kolodny, and Uri C Weiser. MAGIC—memristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(11):895–899, 2014.

[72] Mohsen Imani, Saransh Gupta, Yeseong Kim, and Tajana Rosing. Floatpim: In-memory acceleration of deep neural network training with high precision. In *International Symposium on Computer Architecture (ISCA)*, pages 802–815, 2019.

[73] Hajime Suzuki and Masahiro Kasahara. Acceleration of nucleotide semi-global alignment with adaptive banded dynamic programming. *BioRxiv*, page 130633, 2017.

[74] J Joshua Yang, Dmitri B Strukov, and Duncan R Stewart. Memristive devices for computing. *Nature Nanotechnology*, 8(1):13–24, 2013.

[75] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P Jouppi. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(7):994–1007, 2012.

[76] James E. Stine, Ivan Castellanos, Michael Wood, Jeff Henson, Fred Love, W. Rhett Davis, Paul D. Franzon, Michael Bucher, Sunil Basavarajaiah, Julie Oh, and Ravi Jenkal. FreePDK: An open-source variation-aware design kit. In *IEEE international conference on Microelectronic Systems Education*, pages 173–174, 2007.

[77] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. CACTI 6.0: A tool to model large caches. *HP laboratories*, 27:28, 2009.

[78] National Center for Biotechnology Information. Genome reference consortium human build 38. https://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.26, 2013.

[79] Yukiteru Ono, Kiyoshi Asai, and Michiaki Hamada. PBSIM: PacBio reads simulator—toward accurate genome assembly. *Bioinformatics*, 29(1):119–121, 2013.

[80] M Holtgrewe. Mason–a read simulator for second generation sequencing data. *Technical Report FU Berlin*, 2010.

[81] Wenqin Huangfu, Xueqi Li, Shuangchen Li, Xing Hu, Peng Gu, and Yuan Xie. Medal: Scalable dimm based near data processing accelerator for dna seeding algorithm. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 587–599, 2019.

[82] Qing Luo, Jie Yu, Xumeng Zhang, Kan-Hao Xue, Jun-Hui Yuan, Yan Cheng, Tiancheng Gong, Hangbing Lv, Xiaoxin Xu, Peng Yuan, Jiahao Yin, Lu Tai, Shibing Long, Qi Liu, Xiangshui Miao, Jing Li, and Ming Liu. Nb_{1-x} o2 based universal selector with ultra-high endurance ($> 10^{12}$), high speed (10ns) and excellent v_{th} stability. In *Symposium on VLSI Technology*, pages T236–T237, 2019.

[83] Pierre-Alain Chaumeil, Aaron J Mussig, Philip Hugenholtz, and Donovan H Parks. Gtdb-tk v2: memory friendly classification with the genome taxonomy database. *Bioinformatics*, 38(23):5315–5316, 2022.

[84] Donovan H Parks, Maria Chuvochina, Pierre-Alain Chaumeil, Christian Rinke, Aaron J Mussig, and Philip Hugenholtz. A complete domain-to-species taxonomy for bacteria and archaea. *Nature Biotechnology*, 38(9):1079–1086, 2020.

[85] Julie E Hernández-Salmerón, Tanya Irani, and Gabriel Moreno-Hagelsieb. Fast genome-based delimitation of enterobacterales species. *Plos One*, 18(9):e0291492, 2023.

[86] Stefan Kurtz, Adam Phillippy, Arthur L Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven L Salzberg. Versatile and open software for comparing large genomes. *Genome Biology*, 5:1–9, 2004.

[87] Imchang Lee, Yeong Ouk Kim, Sang-Cheol Park, and Jongsik Chun. OrthoANI: an improved algorithm and software for calculating average nucleotide identity. *International Journal of Systematic and Evolutionary Microbiology*, 66(2):1100–1103, 2016.

[88] Chirag Jain, Luis M Rodriguez-R, Adam M Phillippy, Konstantinos T Konstantinidis, and Srinivas Aluru. High throughput ani analysis of 90k prokaryotic genomes reveals clear species boundaries. *Nature Communications*, 9(1):5114, 2018.

[89] Jim Shaw and Yun William Yu. Fast and robust metagenomic sequence comparison through sparse chaining with skani. *Nature Methods*, 20:1661–1665, 2023.

[90] Chirag Jain, Alexander Dilthey, Sergey Koren, Srinivas Aluru, and Adam M Phillippy. A fast approximate algorithm for mapping long reads to large reference databases. In *International Conference on Research in Computational Molecular Biology*, pages 66–81, 2017.

[91] Daniel N Baker and Ben Langmead. Dashing: fast and accurate genomic distances with hyperloglog. *Genome Biology*, 20:1–12, 2019.

[92] Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pages 21–29, 1997.

[93] Mahmudur Rahman Hera, N Tessa Pierce-Ward, and David Koslicki. Deriving confidence intervals for mutation rates across a wide range of evolutionary distances using fracminhash. *Genome Research*, pages gr–277651, 2023.

[94] Luiz Irber, Phillip T Brooks, Taylor Reiter, N Tessa Pierce-Ward, Mahmudur Rahman Hera, David Koslicki, and C Titus Brown. Lightweight compositional analysis of metagenomes with fracminhash and minimum metagenome covers. *BioRxiv*, pages 2022–01, 2022.

[95] Anshumali Shrivastava. Optimal densification for fast and accurate minwise hashing. In *International Conference on Machine Learning (ICML)*, pages 3154–3163, 2017.

[96] Otmar Ertl. Setsketch: filling the gap between minhash and hyperloglog. *Proceedings of the VLDB Endowment*, 14(11):2244–2257, 2021.

[97] Zhuowen Zou, Hanning Chen, Prathyush Poduval, Yeseong Kim, Mahdi Imani, Elaheh Sadredini, Rosario Cammarota, and Mohsen Imani. Biohd: an efficient genome sequence search platform using hyperdimensional memorization. In *International Symposium on Computer Architecture (ISCA)*, pages 656–669, 2022.

[98] Jaeyoung Kang, Weihong Xu, Wout Bittremieux, Niema Moshiri, and Tajana Rosing. Accelerating open modification spectral library searching on tensor core in high-dimensional space. *Bioinformatics*, 39(7):btad404, 2023.

[99] Yeseong Kim, Mohsen Imani, Niema Moshiri, and Tajana Rosing. Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 115–120, 2020.

[100] Taha Shahroodi, Mahdi Zahedi, Can Firtina, Mohammed Alser, Stephan Wong, Onur Mutlu, and Said Hamdioui. Demeter: A fast and energy-efficient food profiler using hyperdimensional computing in memory. *IEEE Access*, 10:82493–82510, 2022.

[101] Weihong Xu, Jaeyoung Kang, Wout Bittremieux, Niema Moshiri, and Tajana Rosing. Hyperspec: Ultrafast mass spectra clustering in hyperdimensional space. *Journal of Proteome Research*, 2023.

[102] Can Firtina, Jisung Park, Mohammed Alser, Jeremie S Kim, Damla Senol Cali, Taha Shahroodi, Nika Mansouri Ghiasi, Gagandeep Singh, Konstantinos Kanellopoulos, Can Alkan, and Onur Mutlu. Blend: a fast, memory-efficient and accurate mechanism to find fuzzy seed matches in genome analysis. *NAR Genomics and Bioinformatics*, 5(1):lqad004, 2023.

[103] Igor Nunes, Mike Heddes, Pere Vergés, Danny Abraham, Alex Veidenbaum, Alex Nicolau, and Tony Givargis. DotHash: Estimating Set Similarity Metrics for Link Prediction and Document Deduplication. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1758–1769, 2023.

[104] Magnus Sahlgren. An introduction to random indexing. In *International Conference on Terminology and Knowledge Engineering*, 2005.

[105] Pentii Kanerva, Jan Kristoferson, and Anders Holst. Random indexing of text samples for latent semantic analysis. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 22, 2000.

[106] Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation*, 1:139–159, 2009.

[107] Shaopeng Liu and David Koslicki. Cmash: fast, multi-resolution estimation of k-mer-based jaccard and containment indices. *Bioinformatics*, 38:i28–i35, 2022.

[108] Brian D Ondov, Gabriel J Starrett, Anna Sappington, Aleksandra Kostic, Sergey Koren, Christopher B Buck, and Adam M Phillippy. Mash screen: high-throughput sequence containment estimation for genome discovery. *Genome Biology*, 20:1–13, 2019.

[109] Nicholas D Matsakis and Felix S Klock. The rust language. *ACM SIGAda Ada Letters*, 34(3):103–104, 2014.

[110] Lama Sleem and Raphaël Couturier. Testu01 and practrand: Tools for a randomness evaluation for famous multimedia ciphers. *Multimedia Tools and Applications*, 79:24075–24088, 2020.

[111] Donovan H Parks, Christian Rinke, Maria Chuvochina, Pierre-Alain Chaumeil, Ben J Woodcroft, Paul N Evans, Philip Hugenholtz, and Gene W Tyson. Recovery of nearly 8,000 metagenome-assembled genomes substantially expands the tree of life. *Nature Microbiology*, 2(11):1533–1542, 2017.

[112] Donovan H Parks, Maria Chuvochina, David W Waite, Christian Rinke, Adam Skarshewski, Pierre-Alain Chaumeil, and Philip Hugenholtz. A standardized bacterial taxonomy based on genome phylogeny substantially revises the tree of life. *Nature Biotechnology*, 36(10):996–1004, 2018.

[113] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2010.

[114] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning (ICML)*, pages 3887–3896, 2020.

[115] Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11523–11532, 2022.

[116] Paul Ka Po To, Long Wu, Chak Ming Chan, Ayman Hoque, and Henry Lam. Clustersheep: A graphics processing unit-accelerated software tool for large-scale clustering of tandem mass spectra from shotgun proteomics. *Journal of Proteome Research*, 20(12):5359–5367, 2021.

[117] Xiyang Luo, Wout Bittremieux, Johannes Griss, Eric W. Deutsch, Timo Sachsenberg, Lev I. Levitsky, Mark V. Ivanov, Julia A. Bubis, Ralf Gabriels, Henry Webel, Aniel Sanchez, Mingze Bai, Lukas Käll, and Yasset Perez-Riverol. A comprehensive evaluation of consensus spectrum generation methods in proteomics. *Journal of Proteome Research*, 21(6):1566–1574, May 2022.

[118] Yasset Perez-Riverol, Jingwen Bai, Chakradhar Bandla, David García-Seisdedos, Suresh Hewapathirana, Selvakumar Kamatchinathan, Deepti J Kundu, Ananth Prakash, Anika Frericks-Zipper, Martin Eisenacher, Mathias Walzer, Shengbo Wang, Alvis Brazma,

and Juan Antonio Vizcaíno. The pride database resources in 2022: a hub for mass spectrometry-based proteomics evidences. *Nucleic Acids Research*, 50(D1):D543–D552, 2022.

[119] Min-Sik Kim, Sneha M. Pinto, Derese Getnet, Raja Sekhar Nirujogi, Srikanth S. Manda, Raghothama Chaerkady, Anil K. Madugundu, Dhanashree S. Kelkar, Ruth Isserlin, Shobhit Jain, Joji K. Thomas, Babylakshmi Muthusamy, Pamela Leal-Rojas, Praveen Kumar, Nandini A. Sahasrabuddhe, Lavanya Balakrishnan, Jayshree Advani, Bijesh George, Santosh Renuse, Lakshmi Dhevi N. Selvan, Arun H. Patil, Vishalakshi Nanjappa, Aneesha Radhakrishnan, Samarjeet Prasad, Tejaswini Subbannayya, Rajesh Raju, Manish Kumar, Sreelakshmi K. Sreenivasamurthy, Arivusudar Marimuthu, Gajanan J. Sathe, Sandip Chavan, Keshava K. Datta, Yashwanth Subbannayya, Apeksha Sahu, Soujanya D. Yelamanchi, Savita Jayaram, Pavithra Rajagopalan, Jyoti Sharma, Krishna R. Murthy, Nazia Syed, Renu Goel, Aafaque A. Khan, Sartaj Ahmad, Gourav Dey, Keshav Mudgal, Aditi Chatterjee, Tai-Chung Huang, Jun Zhong, Xinyan Wu, Patrick G. Shaw, Donald Freed, Muhammad S. Zahari, Kanchan K. Mukherjee, Subramanian Shankar, Anita Mahadevan, Henry Lam, Christopher J. Mitchell, Susarla Krishna Shankar, Parthasarathy Satishchandra, John T. Schroeder, Ravi Sirdeshmukh, Anirban Maitra, Steven D. Leach, Charles G. Drake, Marc K. Halushka, T. S. Keshava Prasad, Ralph H. Hruban, Candace L. Kerr, Gary D. Bader, Christine A. Iacobuzio-Donahue, Harsha Gowda, and Akhilesh Pandey. A draft map of the human proteome. *Nature*, 509(7502):575–581, 2014.

[120] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[121] Sergei Nakariakov. *The Boost C++ Libraries: Generic Programming*. 2013.

[122] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6, 2015.

[123] Witold E Wolski, Malcolm Farrow, Anne-Katrin Emde, Hans Lehrach, Maciej Lalowski, and Knut Reinert. Analytical model of peptide mass cluster centres with applications. *Proteome Science*, 4:1–19, 2006.

[124] Anthony Thomas, Sanjoy Dasgupta, and Tajana Rosing. Theoretical foundations of hyperdimensional computing. *Journal of Artificial Intelligence Research*, 72:215–249, 2021.

[125] NVIDIA. Cuda, release: 11.6, 2022.

[126] Jaeyoung Kang, Behnam Khaleghi, Yeseong Kim, and Tajana Rosing. Xcelhd: An efficient gpu-powered hyperdimensional computing with parallelized training. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 220–225, 2022.

[127] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, volume 96, pages 226–231, 1996.

[128] Daniel Müllner. fastcluster: Fast hierarchical, agglomerative clustering routines for R and Python. *Journal of Statistical Software*, 53:1–18, 2013.

[129] Vera Rieder, Karin U Schork, Laura Kerschke, Bernhard Blank-Landeshammer, Albert Sickmann, and Jorg Rahnenfuherer. Comparison and evaluation of clustering algorithms for tandem mass spectra. *Journal of Proteome Research*, 16(11):4035–4044, 2017.

[130] Sebastian Raschka, Joshua Patterson, and Corey Nolet. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information*, 11(4):193, 2020.

[131] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39, 2011.

[132] RAPIDS Development Team. *RAPIDS: Collection of Libraries for End to End GPU Data Science*, 2018.

[133] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 410–420, 2007.

[134] Joel M Chick, Deepak Kolippakkam, David P Nusinow, Bo Zhai, Ramin Rad, Edward L Huttlin, and Steven P Gygi. A mass-tolerant database search identifies a large proportion of unassigned spectra in shotgun proteomics as modified peptides. *Nature Biotechnology*, 33(7):743–749, 2015.

[135] Andreas Roos, Laxmikanth Kollipara, Stephan Buchkremer, Thomas Labisch, Eva Brauers, Christian Gatz, Chris Lentz, José Gerardo-Nava, Joachim Weis, and René P Zahedi. Cellular signature of sil1 depletion: disease pathogenesis due to alterations in protein composition beyond the er machinery. *Molecular Neurobiology*, 53(8):5527–5541, 2016.

[136] Timo Glatter, Erik Ahrné, and Alexander Schmidt. Comparison of different sample preparation protocols reveals lysis buffer-specific extraction biases in gram-negative bacteria and human cells. *Journal of Proteome Research*, 14(11):4472–4485, 2015.

[137] François-Michel Boisvert, Yasmeen Ahmad, Marek Gierliński, Fabien Charrière, Douglas Lamont, Michelle Scott, Geoff Barton, and Angus I Lamond. A quantitative spatial proteomics analysis of proteome turnover in human cells. *Molecular & Cellular Proteomics*, 11(3), 2012.

[138] Yasset Perez-Riverol, Attila Csordas, Jingwen Bai, Manuel Bernal-Llinares, Suresh Hewapathirana, Deepti J Kundu, Avinash Inuganti, Johannes Griss, Gerhard Mayer, Martin Eisenacher, Enrique Pérez, Julian Uszkoreit, Julianus Pfeuffer, Timo Sachsenberg, Şule Yılmaz, Shivani Tiwary, Jürgen Cox, Enrique Audain, Mathias Walzer, Andrew F Jarnuczak, Tobias Ternent, Alvis Brazma, and Juan Antonio Vizcaíno. The pride database and related tools and resources in 2019: improving support for quantification data. *Nucleic Acids Research*, 47(D1):D442–D450, 2019.

[139] Niels Hulstaert, Jim Shofstahl, Timo Sachsenberg, Mathias Walzer, Harald Barsnes, Lennart Martens, and Yasset Perez-Riverol. ThermoRawFileParser: modular, scalable, and cross-platform RAW file conversion. *Journal of Proteome Research*, 19(1):537–542, 2019.

[140] Sangtae Kim and Pavel A Pevzner. MS-GF+ makes progress towards a universal database search tool for proteomics. *Nature Communications*, 5(1):1–10, 2014.

[141] Lionel Breuza, Sylvain Poux, Anne Estreicher, Maria Livia Famiglietti, Michele Magrane, Michael Tognolli, Alan Bridge, Delphine Baratin, Nicole Redaschi, and UniProt Consortium. The UniProtKB guide to the human proteome. *Database*, 2016:bav120, 2016.

[142] Jimmy K. Eng, Brian C. Searle, Karl R. Clauser, and David L. Tabb. A face in the crowd: Recognizing peptides through database search. *Molecular & Cellular Proteomics*, 10(11):R111.009522, November 2011.

[143] Henry Lam, Eric W. Deutsch, James S. Eddes, Jimmy K. Eng, Nichole King, Stephen E. Stein, and Ruedi Aebersold. Development and validation of a spectral library searching method for peptide identification from MS/MS. *Proteomics*, 7(5):655–667, March 2007.

[144] Wout Bittremieux, Kris Laukens, and William Stafford 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*, 18(10):3792–3799, August 2019.

[145] Mohsen Imani, Tarek Nassar, Abbas Rahimi, and Tajana Rosing. Hdna: Energy-efficient dna sequencing using hyperdimensional computing. In *IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*, pages 271–274, 2018.

[146] Jaeyoung Kang, Weihong Xu, Wout Bittremieux, and Tajana Rosing. Massively parallel open modification spectral library searching with hyperdimensional computing. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 536–537, 2022.

[147] Weihong Xu, Jaeyoung Kang, and Tajana Rosing. A near-storage framework for boosted data preprocessing of mass spectrum clustering. In *ACM/IEEE Design Automation Conference (DAC)*, pages 313–318, 2022.

[148] Lennart Martens, Henning Hermjakob, Philip Jones, Marcin Adamski, Chris Taylor, David States, Kris Gevaert, Joël Vandekerckhove, and Rolf Apweiler. PRIDE: the proteomics identifications database. *Proteomics*, 5(13):3537–3545, 2005.

[149] NVM Express Base Specification 2.0. https://nvmexpress.org/developers/nvme-specification/, 2022.

[150] Intel SSD DC P4500 Series. https://ark.intel.com/content/www/us/en/ark/products/series/96935/intel-ssd-dc-p4500-series.html, 2017.

[151] Wikipedia. Mass Spectrometry. https://en.wikipedia.org/wiki/Mass_spectrometry, 2022.

[152] David N Perkins, Darryl JC Pappin, David M Creasy, and John S Cottrell. Probability-based protein identification by searching sequence databases using mass spectrometry data. *Electrophoresis*, 20(18):3551–3567, 1999.

[153] Şule Yilmaz, Elien Vandermarliere, and Lennart Martens. Methods to calculate spectrum similarity. *Proteome Bioinformatics*, pages 75–100, 2017.

[154] Vikram Sharma Mailthody, Zaid Qureshi, Weixin Liang, Ziyan Feng, Simon Garcia De Gonzalo, Youjie Li, Hubertus Franke, Jinjun Xiong, Jian Huang, and Wen-mei Hwu. Deepstore: In-storage acceleration for intelligent queries. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 224–238, 2019.

[155] Feng Chen, Binbing Hou, and Rubao Lee. Internal parallelism of flash memory-based solid-state drives. *ACM Transactions on Storage (TOS)*, 12(3):1–39, 2016.

[156] Ryuji Yamashita, Sagar Magia, Tsutomu Higuchi, Kazuhide Yoneya, Toshio Yamamura, Hiroyuki Mizukoshi, Shingo Zaitsu, Minoru Yamashita, Shunichi Toyama, Norihiro Kamae, Juan Lee, Shuo Chen, Jiawei Tao, William Mak, Xiaohua Zhang, Ying Yu, Yuko Utsunomiya, Yosuke Kato, Manabu Sakai, Masahide Matsumoto, Hardwell Chibvongodze, Naoki Ookuma, Hiroki Yabe, Subodh Taigor, Rangarao Samineni, Takuyo Kodama, Yoshihiko Kamata, Yuzuru Namai, Jonathan Huynh, Sung-En Wang, Yankang He, Trung Pham, Vivek Saraf, Akshay Petkar, Mitsuyuki Watanabe, Koichiro Hayashi, Prashant Swarnkar, Hitoshi Miwa, Aditya Pradhan, Sulagna Dey, Debasish Dwibedy, Thushara Xavier, Muralikrishna Balaga, Samiksha Agarwal, Swaroop Kulkarni, Zameer Papasaheb, Sahil Deora, Patrick Hong, Meiling Wei, Gopinath Balakrishnan, Takuya Ariki, Kapil Verma, Chang Siau, Yingda Dong, Ching-Huang Lu, Toru Miwa, and Farookh Moogat. A 512Gb 3b/cell flash memory on 64-word-line-layer BiCS technology. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 196–197, 2017.

[157] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D Davis, Mark Manasse, and Rina Panigrahy. Design tradeoffs for SSD performance. In *USENIX Annual Technical Conference (ATC)*, 2008.

[158] Open NAND Flash Interface Specification. http://www.onfi.org/specifications, 2021.

[159] Wooseong Cheong, Chanho Yoon, Seonghoon Woo, Kyuwook Han, Daehyun Kim, Chulseung Lee, Youra Choi, Shine Kim, Dongku Kang, Geunyeong Yu, Jaehong Kim, Jaechun Park, Ki-Whan Song, Ki-Tae Park, Sangyeun Cho, Hwaseok Oh, Daniel D.G. Lee, Jin-Hyeok Choi, and Jaeheon Jeong. A flash memory controller for 15μs ultra-low-latency SSD using high-speed 3D NAND flash with 3μs read time. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 338–340, 2018.

[160] Anil Shanbhag, Holger Pirk, and Samuel Madden. Efficient top-k query processing on massively parallel hardware. In *International Conference on Management of Data*, pages 1557–1570, 2018.

[161] Juan A Vizcaíno, Eric W Deutsch, Rui Wang, Attila Csordas, Florian Reisinger, Daniel Ríos, José A Dianes, Zhi Sun, Terry Farrah, Nuno Bandeira, Pierre-Alain Binz, Ioannis Xenarios, Martin Eisenacher, Gerhard Mayer, Laurent Gatto, Alex Campos, Robert J Chalkley, Hans-Joachim Kraus, Juan Pablo Albar, Salvador Martinez-Bartolomé, Rolf Apweiler, Gilbert S Omenn, Lennart Martens, Andrew R Jones, and Henning Hermjakob. Proteomexchange provides globally coordinated proteomics data submission and dissemination. *Nature Biotechnology*, 32(3):223–226, 2014.

[162] Dirk Beyer and Philipp Wendler. Cpu energy meter: A tool for energy-aware algorithms engineering. *Tools and Algorithms for the Construction and Analysis of Systems*, 12079:126, 2020.

[163] Myoungsoo Jung, Wonil Choi, Shuwen Gao, Ellis Herbert Wilson III, David Donofrio, John Shalf, and Mahmut Taylan Kandemir. NANDFlashSim: High-fidelity, microarchitecture-aware NAND flash memory simulation. *ACM Transactions on Storage (TOS)*, 12(2):1–32, 2016.