FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

HIGH-PERFORMANCE COMPUTING ALGORITHMS FOR ACCELERATING PEPTIDE IDENTIFICATION FROM MASS-SPECTROMETRY DATA USING HETEROGENEOUS SUPERCOMPUTERS

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Muhammad Haseeb

2023

To: Dean John L. Volakis
College of Engineering and Computing

This dissertation, written by Muhammad Haseeb, and entitled High-Performance Computing Algorithms for Accelerating Peptide Identification from Mass-Spectrometry Data using Heterogeneous Supercomputers, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Ananda Mondal

Jason Liu

Janki Bhimani

Jun Li

Ashok Srinivasan

Fahad Saeed, Major Professor

Date of Defense: March 15, 2023

The dissertation of Muhammad Haseeb is approved.

Dean John L. Volakis
College of Engineering and Computing

Andrés G. Gil
Vice President for Research and Economic Development
and Dean of the University Graduate School

Florida International University, 2023

DEDICATION

I dedicate this work to my loving wife, Bisma, my amazing siblings, Aaqib, Aqsa and Husnain, and most of all, my parents. This has been possible due to your unconditional and constant love and support.

ACKNOWLEDGMENTS

This dissertation marks the end of the journey of a lifetime for me. Looking back at the last five years, the road to PhD has been long and tedious but also rewarding, as summed in J.R.R. Tolkien's words, "All's well that ends better". For this, first and foremost, I would like to acknowledge my PhD advisor, Dr. Fahad Saeed for guiding me through this journey with his excellent mentorship. His constant push for shooting for the stars and either landing on them or proving it is impossible, provided me with the essential skills and attitude to perform and *disseminate* impactful research. It has been an honor and privilege to be a part of the lab. I would also like to thank my PhD committee members, Dr. Ananda Mondal, Dr. Jason Liu, Dr. Janki Bhimani, Dr. Jun Li, and Dr. Ashok Srinivasan, who always provided their prompt and useful feedback, instrumental to the completion of this work. I would also like to thank Dr. Saeed's wife, Saba, for inviting the lab group over for dinners on holidays and events, especially Eid. It always refreshed a little part of the celebrations and feasts from Pakistan. I would also like to thank my lab mates, Muaaz, Usman, Sandino, Taban, Sumesh, Fahad, Maryam, Fatima, Tianren and Umair, and my friends, Talha, Arslan, Anwer, Talal, Daim, Ayaz, Vitalii, and Daniel for all the technical help, discussions, hangouts and laughs. Also, I would like to thank Dr. Giri and his research group for social gatherings and treats at Vicky's Cafe. Also, I would like to acknowledge all the prompt help from the SCIS staff including Rebeca, Olga, Luis, Sydney, Eric and John. Finally, I would like to thank Jonathan, Muaaz and Jack for providing an inspiring and learning experience at the Berkeley Lab.

ABSTRACT OF THE DISSERTATION

HIGH-PERFORMANCE COMPUTING ALGORITHMS FOR ACCELERATING PEPTIDE IDENTIFICATION FROM MASS-SPECTROMETRY DATA USING HETEROGENEOUS SUPERCOMPUTERS

by

Muhammad Haseeb

Florida International University, 2023

Miami, Florida

Professor Fahad Saeed, Major Professor

Fast and accurate identification of peptides and proteins from the mass spectrometry (MS) data is a critical problem in modern systems biology. Database peptide search is the most commonly used computational method to identify peptide sequences from the MS data. In this method, giga-bytes of experimentally generated MS data are compared against tera-byte sized databases of theoretically simulated MS data resulting in a compute- and data-intensive problem requiring days or weeks of computational times on desktop machines. Existing serial and high performance computing (HPC) algorithms strive to accelerate and improve the computational efficiency of the search, but exhibit sub-optimal performances due to their inefficient parallelization models, low resource utilization and high overhead costs.

In this dissertation, we design and develop data- and architecture-aware algorithms and optimizations to accelerate the database peptide search algorithms on heterogeneous distributed-memory (top-500) supercomputers. We first present an HPC framework which efficiently parallelizes both the compute- and memory-intensive portions of the database peptide search workloads across homogeneous supercomputers achieving a 10× speed improvement against the state-of-the-art algorithms. To achieve maximum performance, we also develop several optimizations

including a low-overhead algorithm for balanced distribution of the voluminous theoretical MS databases, and a novel data structure to reduce the memory footprint of these databases by 2× without compromising the query speeds. We also developed GPU-accelerated algorithms, data pipelines and optimizations to leverage the heterogeneous (CPU-GPU) supercomputing architectures and further accelerate our HPC framework by 4×, providing a combined acceleration of 40× over existing shared- and distributed-memory, and GPU-accelerated software infrastructure. Furthermore, we extensively analyze the performance of our developed methods and show near-optimal results for several metrics including the throughput, resource utilization and overheads. Finally, we explore possible extension methods for our methods to accelerate the existing and new numerical, and machine- and deep-learning based peptide identification algorithms.

Our advancements in the HPC software infrastructure for ultrafast peptide identification have key application in meta-proteomics, multiomics, and cancer research, which require astronomical computational resources to process tera-byte scale raw MS-data at swift rates leading to useful scientific investigations and discoveries in the respective domains.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI Arithmetic Intensity

BSP Bulk Synchronous Parallel

CFIR Compact Fragment-ion Index Representation

CPU Central Processing Unit

CUDA Compute Unified Device Architecture

DL Deep Learning

DNA Deoxyribonucleic Acid

ERT Empirical Roofline Toolkit

FFT Fast Fourier Transform

GPU Graphics Processing Unit

HPC High Performance Computing

LC Liquid Chromatography

ML Machine Learning

MPI Message Passing Interface

MR Map Reduce

MS Mass Spectrometry

NCU NVIDIA Nsight Compute

OpenMP Open Multi-Processing

PTM Post-translational Modifications

SPMD Single Program, Multiple Data

STA Sorted Tag Array

CHAPTER 1

INTRODUCTION

1.1 Motivation

Proteins and peptides are the cellular workhorses in all living organisms and play several critical roles in keeping us alive and functioning. These roles include protecting us from bacteria, viruses and tumors, digesting our food, carrying messages, healing wounds, and translating genes [RSSK14], [AS16]. Proteins and peptides are chain-structured biological molecules composed of combinations of twenty basic amino acid. Shorter amino acid chains are referred to as *peptides* whereas longer chains are referred to as proteins. Therefore, a fundamental and critical problem in computational proteomics is the swift and accurate identification of proteins (and peptides) in a biological mixture [HS21]. The peptide identification problem is also referred to as peptide sequencing problem as it involves deducing the amino acid sequence(s) of each peptide/protein.

1.2 Introduction to Peptide Sequencing

The peptide sequencing process involves a two-step approach involving 1) experimental data generation and 2) processing of the experimental data to deduce protein/peptide sequences.

1.2.1 Experimental Step

In the experimental step, protein mixture is proteolyzed into peptides using an enzyme, such as Trypsin [NRG$^+$06]. The peptides are then passed to a liquid-chromatography (LC) coupled two-step mass spectrometry (MS/MS or MS^2) pipeline

[AS16]. The LC-MS/MS pipeline separates the peptide molecules and fragments them. During fragmentation, the molecules of each peptide are ionized and broken into smaller (charged) fragments. These fragments are detected by the mass-spectrometer and for each peptide, a (fingerprint) histogram is generated with the mass-to-charge ratio of fragment-ions on the x-axis and their relative abundance, or simply intensity, on the y-axis [THA+20]. These histograms are referred to as the experimental MS/MS spectra data. The LC-MS/MS based experimental pipeline setup for data generation is illustrated in Figure 1.1.

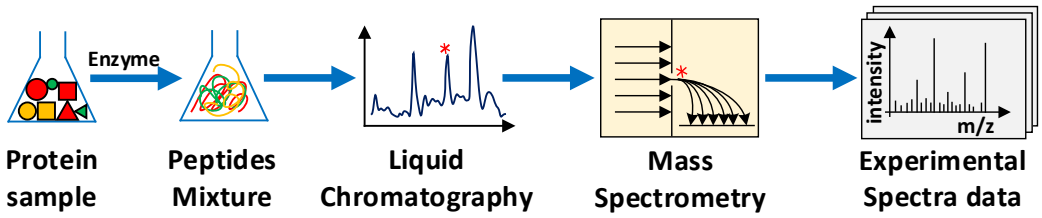

Figure 1.1: The proteins in the sample are digested into peptides. The resultant peptides are passed to a liquid chromatography (LC) coupled MS/MS pipeline yielding the spectra. This experimental technique is also known as the bottom-up proteomics or shotgun proteomics [HS21].

1.2.2 Computational Step

In the computational step, each experimental MS/MS spectrum is processed to deduce the sequence of the peptide. This is typically achieved by either de-novo methods [MDWC07], [YCZ+17], [QTL+19], spectral library search methods [LA10], [YFS+10], [BMNL18], or most commonly, database peptide search methods [KLA+17], [MTKF+14], [CLY+18]. In the database peptide search method, the experimental MS/MS spectra data are compared against a (indexed) database of theoretically simulated MS/MS spectra to find the correct match [NRG+06]. This theoretical MS/MS spectra database is simulated through probabilistic [HS21] or machine learning mod-

els [DM13] generating MS/MS data from proteome sequence databases. The generic database peptide search workflow is illustrated in Figure 1.2. Peptide deduction is often also followed by statistical significance scoring and false discovery rate computations to eliminate spurious and/or random matches between the experimental and reference MS/MS data [CB03], [MVN12]. More recent methods have also employed machine and deep learning to learn the translations between experimental MS/MS spectra and peptide sequences for their identification [TS21].

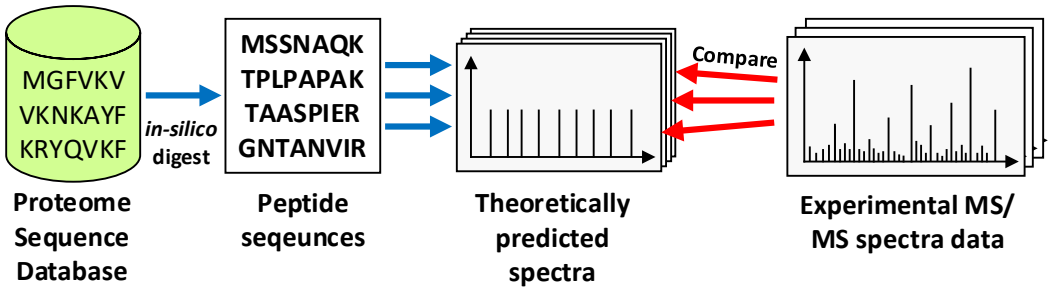

Figure 1.2: Protein sequences are in-silico digested (along with PTMs) followed by theoretical MS/MS spectra simulation using a probabilistic (or ML) model. The experimental MS/MS spectra are compared against the theoretical spectra database to find the best match [HS21].

1.3 Data Volumes

Modern mass spectrometry technologies such as the Thermo Orbitrap and TIMS-TOF, allow the generation of tens of thousands of raw, noisy experimental MS/MS spectra in only a few hours [HRB+14]. Each experimental spectrum contains, on average, between 1000 to 4000 two-dimensional (mass-to-charge and intensity) data points, yielding several gigabytes of raw data per experiment [AS16]. Similarly, the theoretically simulated MS/MS spectra databases grow exponentially ($O(2^n)$ yielding several hundred giga bytes to a few tera bytes databases [KLA+17], [HS19] as the commonly occurring post-translational modification (PTMs) are incorporated

in them. Consequently, the process of comparing several gigabytes of experimental MS/MS data against terabytes of theoretical MS/MS data results in trillions of spectral similarity computations requiring several days (or weeks) of execution on desktop computers [HS21]. Even with the use of data reductive techniques [DRP+19], [TSY03], [KLA+17], the asymptotic lower bounds remain quadratic [SHI22].

1.4 Problem Statement

Database peptide search is the most commonly employed peptide deduction technique in systems biology labs [KLA+17], [Nes10], [HS21]. For over three decades, researchers have developed several algorithmic advances including modern indexing, filtering, graph traversal, and machine and deep learning techniques [DLZ+19], [KLA+17], [CLY+18], [TS21] to accelerate the database peptide search algorithms. While these advances have been useful, they have tremendously shifted the computational profile of the modern database peptide search algorithms from compute-intensive to *data-intensive*, resulting in performance overheads from rapid memory contention, excessive out-of-core processing, and data communication [CLY+18], [KLA+17], [HS21] in shared-memory, multi and many-core architectures. In other words, the algorithmic techniques employed in the state-of-the-art algorithms lead to a sharp drop in their arithmetic intensity (AI) [WWP09]. i.e., the compute operations per byte, leading to performance saturations at a fraction of the maximum CPU throughput [WWP09].

With continued research and development [Mar13], these performance bottlenecks can be reduced through sophisticated (heterogeneous) HPC algorithms on supercomputing architectures. These scalable solutions will be inline with the resource demands of modern proteomics, cancer and systems biology, and multiomics

applications, where the data scales are astronomical and the experiment execution times are critical. Accelerating these pipelines also directly impacts the discoveries in microbiome research [KBC+18], personalized nutrition [RA19], and cancer therapeutics [YI19].

1.5 Developed Solutions and Objectives

In order to efficiently accelerate the large-scale systems biology pipelines, we designed and developed a high-performance computing (HPC) framework, containing:

1. Efficient distributed-memory parallel computational pipelines for scalable acceleration of the modern database peptide search algorithms on supercomputers (Chapter 3).
2. Algorithm for load balanced distribution of the database peptide search computational workload across the distributed-memory system (Chapter 4).
3. Data-structure for lossless compression of the massive fragment-ion index in modern database peptide search algorithms without compromising the search speed (Chapter 5).
4. GPU-acceleration of the HPC framework on modern heterogeneous computing distributed-memory architectures (Chapter 6).

Achieving these objectives allow our HPC framework to optimally exploit the homogeneous and heterogeneous distributed-memory architectures in modern supercomputers and extract the maximum compute, memory, I/O, and communication bandwidths. Further, to demonstrate its practical application, we implemented a MSFragger-like fragment-ion search based database peptide search algorithm, and its GPU-accelerated version in our framework, but the proposed algorithms, pipelines,

and optimizations would be designed as *search-algorithm-oblivious* to allow our HPC framework to be elastic and adaptable for efficiently accelerating most existing and future database peptide search algorithms. We believe that our advancements described in this dissertation will unlock the full performance of new and existing database peptide search algorithms and help advance science. Our HPC framework is available as open source at: https://github.com/pcdslab/gicops.

CHAPTER 2

A REVIEW OF DATABASE PEPTIDE SEARCH ALGORITHMS

In this chapter, we will discuss the algorithmic and computational advances in the database peptide search algorithms over the years. We will also discuss the bottlenecks in the current state-of-the-art serial, high-performance computing (HPC) and CPU-GPU algorithms that limit their application in real-world systems biology pipelines.

2.1 The Core Computational Problem

The core algorithmic problem in the database peptide search methods is the computation of *similarities* between the experimental MS/MS spectra and the theoretical MS/MS spectra database. Before processing, both the theoretical and experimental MS/MS spectra are represented as discrete signals by appropriately quantifying the m/z (x) and intensity (y) axes [AS16]. This allows leveraging the signal processing techniques such as Fast Fourier Transforms (FFTs) [EJH13] to compute correlations [EMY94] and convolutions between them which became the premise of the very first database peptide search algorithm, SEQUEST [EMY94]. Even the modern database peptide search algorithms, such as MSFragger, leverage the discrete signal representation of the MS/MS spectra to compute cosine-similarity-like scores between the spectra [KLA+17].

2.2 Algorithmic Advances in the Database Peptide Search

In 1994, SEQUEST [EMY94] was introduced as the first database peptide search algorithm employing a signal cross-correlation (xcorr) algorithm to compute the similarity between many-to-many pairs of the theoretical and experimental MS/MS

spectra. In subsequent years, the database peptide search and spectral library search algorithms [LDE+06] employed noise reductions, multi-tiered scoring [XPV+15], and Fast Fourier Transform (FFT) [EJH13], and estimations [MTKF+14] to improve the speed and accuracy of the SEQUEST algorithm. Simultaneously, several other spectral similarity metrics were proposed and employed in Mascot [PPCC99], X!Tandem [CB03], OMSSA [GMK+04], MaxQuant [CM08], Andromeda [CNM+11], HMMatch [WTE07], PEAKS [ZXS+12]. These similarity metrics exploited probabilistic modeling, indexing, scoring, filtering, graph and network traversals, and re-ranking techniques to improve the search quality and speed. However, even with significant algorithmic advances, a significant fraction of the experimental MS/MS spectra data remained unidentified, which are referred to as the *dark matter of the shotgun proteomics* [SK15].

2.2.1 Illuminating the Dark Matter

Chick et al. [CKN+15] showed that this dark matter originates from the presence of the unknown post-translational modifications (PTMs) in the experimental MS/MS spectra. These unknown PTMs allow spectra to evade the peptide precursor mass tolerance settings in the closed-search database peptide search algorithms and remain unidentified. To alleviate this, the authors also introduced the idea of *open-search* where the precursor mass tolerance settings are widened from $\pm \leq 1$ Daltons (Da) to ± 500 Da allowing the identification of mutated spectra [SK15]. However, widening the peptide precursor mass settings also leads to an exponential increase in the computational cost of the open-search algorithms, leading to weeks to months of execution times [KLA+17]. To alleviate this, several secondary database filtering techniques such as sequence-tagging [ZXS+12], [TSY03], and fragment-ion indexing

[CHY+15] are incorporated in modern database peptide search algorithms including Open-pFind [CLY+18], TagGraph [DLZ+19], MSFragger [KLA+17], significantly improving their speed while achieving similar or better identification rates.

2.2.2 Machine and Deep-Learning in Database Peptide Search

More recently, several machine and deep learning (ML/DL) based database peptide search techniques have been introduced. These techniques employ complex neural networks [GSZ+19], [GSG+22] and language translation models [TS21] to learn the relations between the experimental MS/MS spectra data and their peptide sequences, even with the presence of a few post-translational modifications (PTMs). Interestingly, these models can also simulate more accurate theoretical MS/MS spectra from the reference peptide sequences [DM13], which can be employed in the traditional database peptide search algorithms.

2.3 Computational Advances in the Database Peptide Search

In this section, we will discuss the advancements and limitations of the computational techniques employed in database peptide search algorithms.

2.3.1 Shared-Memory Database Peptide Search Algorithms

The computational trends in the database peptide search algorithms evolved with the advances in computing architectures and technologies [SHI22]. While the earlier algorithms were serial in nature, almost all algorithms developed beyond early 2000s [MTKF+14], [EJH13], [CNM+11] employ multicore parallelism. This is because the shared-memory multicore architectures allow for an intuitive and effective data

parallel design where the parallel cores can search a partition of the experimental MS/MS data are spread against a common (shared-memory, low latency) database. On the flip side, the maximum achievable performance in the modern (post-Moore) architectures is limited by the performance gap between the maximum CPU and memory throughputs.

Modern (open-search) database peptide search algorithms employ complex and sophisticated memory-lookups and graph traversals to achieve more than 100× speedup over their predecessors. However, by doing so, their computational profiles have greatly shifted from compute-intensive to memory- and data-intensive [SHI22] and the spectral similarity computations contribute less than 50% of the compute workload (previously 80-90% [LLC+19]). Consequently, their arithmetic intensity (AI), defined as the number of compute operations per byte, has greatly reduced leading to poor scalability as memory bandwidth bottlenecks the achievable performance. For instance, MSFragger reports performance saturation beyond 8 parallel cores (see Supplementary Figure 2 in [KLA+17]), and X!Tandem's performance saturates beyond 4 parallel cores [BCC+07], [LLLL19]. The memory bandwidth bottlenecks are particularly significant when post-translational modifications (PTMs) are incorporated in the theoretical MS/MS spectra database [SHI22] as illustrated in Figure 2.1. [Mar13].

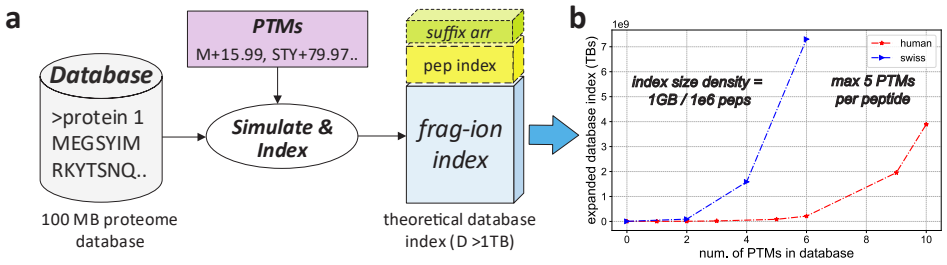

Figure 2.1: **(a)** The theoretical MS/MS spectra database grows combinatorially as the PTMs are added to the simulation. **(b)** The memory-footprint of adding PTMs to the Homo sapiens and the SwissProt (reviewed) database is shown [HS21].

2.3.2 Distributed-Memory Database Peptide Search Algorithms

Distributed-memory architectures allow distributing the compute load across a cluster of compute nodes and are employed to alleviate the performance bottlenecks in the shared-memory algorithms as also demonstrated by many scientific fields [Mar13].

Parallel Tandem [DCL05] is one of the first distributed-memory database peptide search algorithms. It achieves parallelism by dividing the experimental MS/MS spectra data across the available parallel compute nodes, all searching against a (local) replicated copy of the same database, using the X!Tandem algorithm, using MPI or PVM. Other methods including X!!Tandem [BCC+07], MR-Tandem [PHTN11], and SW-Tandem [LLC+19], also employ a similar parallelization model with some optimizations to achieve better efficiency and load balance across the parallel nodes. Bolt implements a distributed-memory version of a MSFragger-like algorithm, MS-PyCloud [CZS+18] implements a cloud-computing version of the MS-GF+ [KP14] algorithm, and UltraQuant implements a distributed-memory version of the MaxQuant37 algorithm.

Notice that the parallelization models employed the distributed-memory algorithms are identical to the shared-memory designs, except the experimental data are partitioned across distributed-memory nodes, each searching it against a local instance of the database. However, replicating voluminous databases on all system nodes and searching them results in the same memory bandwidth bottlenecks on all system nodes resulting in less than 50% strong-scale efficiency as seen in the reported results in Figure 2.2. To further show this, we performed our own experimentation using several existing distributed-memory tools including X!!Tandem,

SW-Tandem, as well as self-implemented distributed-memory versions of Comet-MS, MSFragger, and Tide/Crux. The results in Figure 2.3a to 2.3d corroborate that the existing methods result in a low compute-to-overhead ratio leading to low speedup efficiency, also reported in [SHI22].

In 2009, Kulkarni et al. [KKCB09] introduced a preliminary split-database and streaming based parallelization model greatly reducing the per-node memory intensity of database peptide search algorithms. However, the proposed models demonstrate about 50% parallel efficiency due to enormous amounts of required on-the-fly computations and frequent data communications reported by the authors [KKCB09].

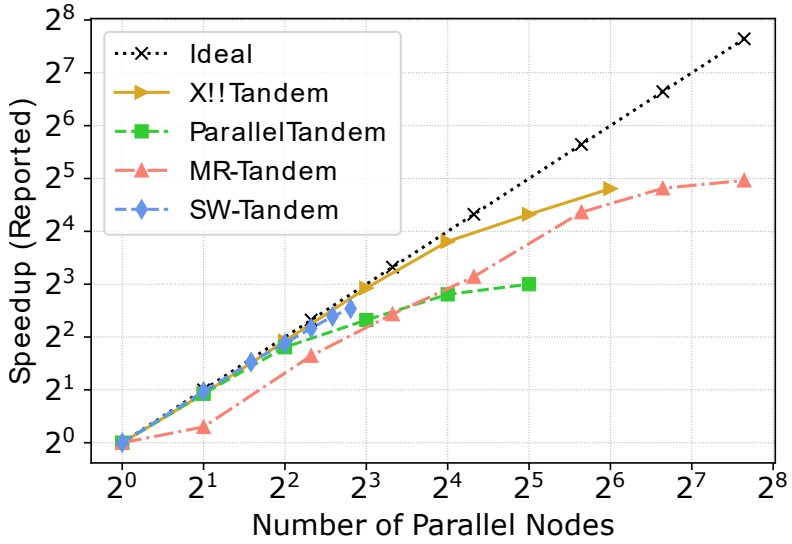

Figure 2.2: The scalability results reported by the existing distributed-memory database peptide search algorithms depict saturating strong-scale efficiency as the degree of parallelism increases.

2.3.3 GPU-Accelerated Database Peptide Search Algorithms

Graphics Processing Units (GPUs) have emerged as the primary and ubiquitous hardware accelerator in the next generation of HPC systems [MAB+20]. The main computational pattern (or motif) in the earlier database peptide search algorithms

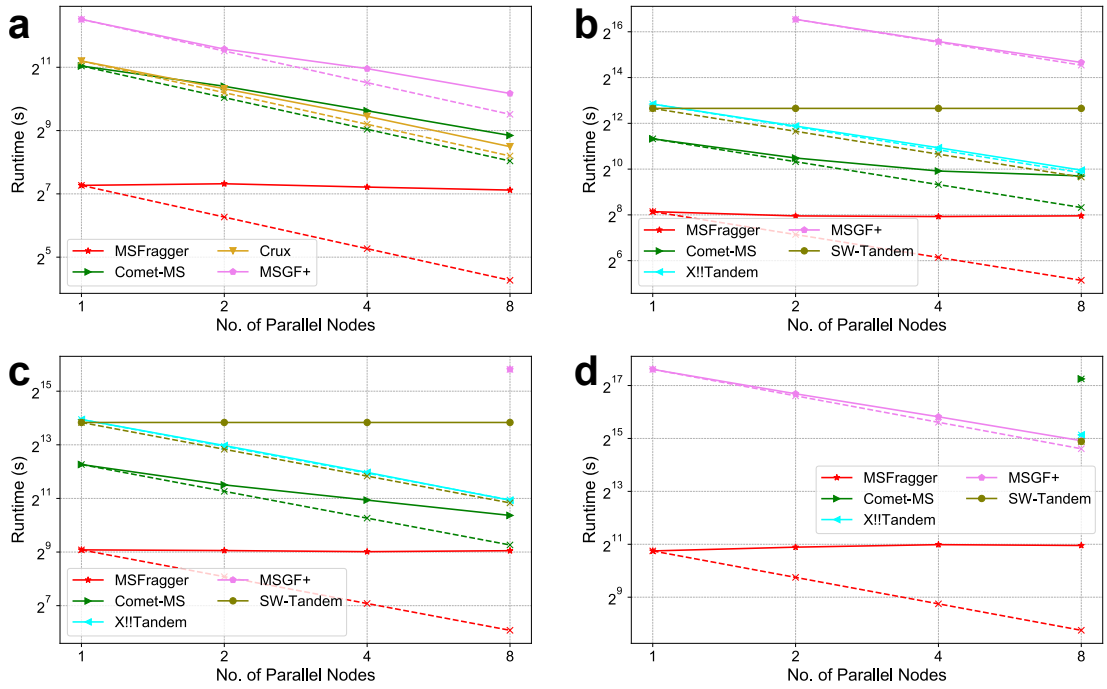

Figure 2.3: Our incrementing sized database peptide search experiments from **(a)** to **(d)** show that the existing distributed-memory database peptide search algorithms depict poor strong-scale efficiency (measured efficiency: solid lines, ideal performance: corresponding dotted lines, less positive deviation is better).

is the matrix-matrix and matrix-vector based spectral similarity computations. i.e., xcorr, FFT, and dot product, etc. These matrix computations [ABD+08] are efficiently parallelized by GPUs due to their Single Instruction, Multiple Threads (SIMT) architectures.

Consequently, several database peptide search algorithms including Tempest [MFG12], Tide-for-PTM-search [KHUP18], GPUScorer [LXCC14], ProteinByGPU [LCXC14], GPU SDP and KSDP [LC12], MIC-Tandem [HL15], and PaSER [Bru23] employed GPUs to significantly ($> 50\times$) accelerate their matrix computations as seen in Figure 2.4. Some algorithms also exploited *vector sparsity*, loop unrolling, cache and register usage for further optimize their compute times [MFG12], [LC12], [LCXC14].

Nonetheless, even with these speedups, these GPU-accelerated algorithms are easily outperformed by several modern CPU-based database peptide search algorithms which employ sophisticated optimizations to significantly ($> 150\times$) speed up their compute times [KLA+17], [CLY+18] (more details in section 6.5.3). Furthermore, as the dominant computational pattern in the modern database peptide search algorithms has shifted from (GPU-friendly) matrix-vector computations, the existing GPU-accelerated algorithms and data pipelines cannot be directly applied to accelerate the complex memory-intensive graph traversals and memory lookup patterns. We are not aware of any GPU-based algorithm that efficiently accelerates modern open-search database algorithms.

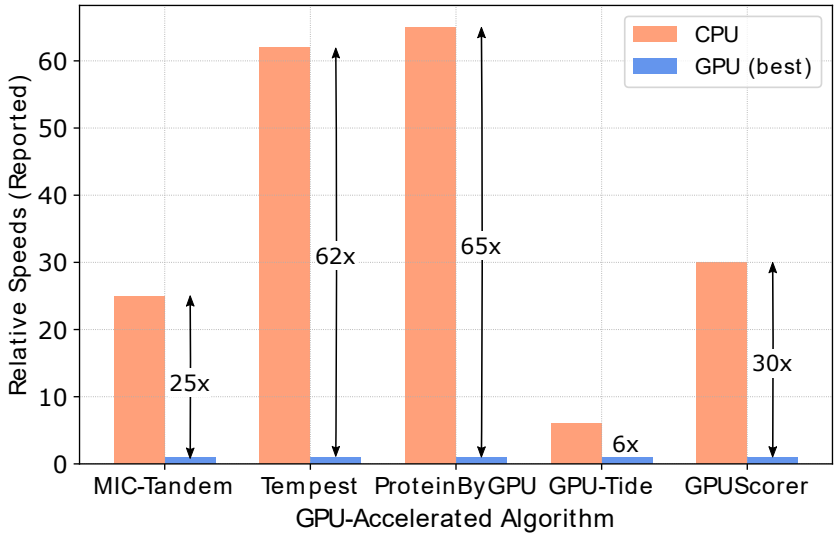

Figure 2.4: The existing GPU-accelerated database peptide search algorithms report over $50\times$ speedups over the CPU implementations. However, these GPU-accelerated algorithms are based on older (index-free) algorithms and are outperformed by orders-of-magnitude using the state-of-the-art CPU-based algorithms [KLA+17], [CLY+18], [HS21], highly limiting their application in the domain.

2.4 The Premise of this Dissertation

The existing computational infrastructure for the database peptide search is incapable of achieving serious performance and accelerations on modern heterogeneous (CPU-GPU) distributed-memory supercomputers. This is primarily because the existing algorithms and data pipeline are either based on (index-free) closed-search methods and/or do not optimally exploit the distributed-memory architectures to accelerate the complex computational patterns in the modern database peptide search algorithms, leaving a significant performance gap. We aim to bridging this gap, one sub-problem at a time, throughout the rest of this dissertation.

CHAPTER 3

HICOPS: DATABASE PEPTIDE SEARCH ON SUPERCOMPUTERS

In this chapter, we will discuss the design, development, and optimization of our high performance computing (HPC) computational software framework, called HiCOPS, for scalably accelerating the database peptide search workloads and algorithms on symmetric multiprocessor (top-500) supercomputers [HS21]. We also discuss the reported speed improvements, parallel performance, hardware utilization, and optimizations implemented in HiCOPS as well as its application in tera-scale database peptide search application.

3.1 Computational Steps in HiCOPS

HiCOPS strives to accelerate and optimize the four main computational steps of the modern database peptide search algorithmic workflow:

1. **Database Indexing:** Simulate the theoretical MS/MS spectra database and index.
2. **Experimental MS/MS Data Preprocessing:** Preprocess the experimental MS/MS spectra data
3. **Database Search:** Search the preprocessed experimental MS/MS data against the indexed database.
4. **Postprocessing:** Compute confidence scores and false discovery rates (FDRs).

These four steps are also applicable to the newly developed machine and deep learning based database peptide search algorithms where the database indexing (first) step is replaced by model training and testing and the database search (third) step is replaced by the inference.

3.2 The HiCOPS Framework

The HiCOPS software framework implements an inverted parallel design in which the theoretical MS/MS spectra databases are distributed across the system nodes and an asynchronous local database peptide search is executed [HS21]. On completion, the local (partial) results at the parallel nodes are assembled into global results and postprocessed in a communication-optimal manner. In stark contrast to the existing parallel designs, HiCOPS distributes both the compute- and data-workload across the parallel nodes, alleviating the memory bandwidth saturations.

The HiCOPS's design employs a Single Program, Multiple Data (SPMD) based *four* Bulk Synchronous Parallel (BSP) [Val90] model with *four* supersteps, each accelerating a respective database peptide search step. A superstep in BSP model [Tis11] refers to a set of algorithmic and data communication kernels executed by all parallel nodes (or process) ($p_i \in P$) in an asynchronous fashion. The parallel nodes executing these supersteps synchronize at the end of each superstep. A schematic of the BSP model is shown in the Figure 3.1.

In the first superstep, the theoretical spectra database is distributed across the parallel nodes where the sub-databases are locally indexed. In the second superstep, the parallel nodes preprocess the experimental MS/MS data and write it to the shared file system. In the third step, the parallel nodes search the entire (preprocessed) experimental MS/MS spectra data set against their (indexed) local sub-databases and write the (partial) results to the shared file system. In the final superstep, the parallel nodes assemble and compute the global results and communicate them among one another in one all-to-all step as shown in Figure 3.2. Figure 3.3 depicts the parallelization model and the workload profiles of each superstep. From

Figure 3.1, the HiCOPS execution time (T_H) for its four supersteps is given as:

$$T_H = \sum_{j=1}^{4} T_j$$

where a superstep (j)'s execution time is the maximum time for any BSP node ($p_i \in P$) to complete that superstep, given as:

$$T_j = max(T_{j,p_1}, T_{j,p_2}, \cdots, T_{j,p_P}) = max_{p_i}(T_{j,p_i})$$

Combining the above two equations, we have:

$$T_H = \sum_{j=1}^{4} max_{p_i}(T_{j,p_i}) \tag{3.1}$$

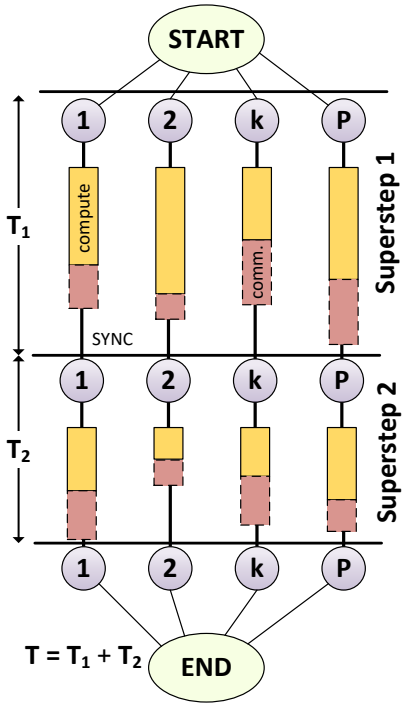

Figure 3.1: In the BSP model, the parallel nodes asynchronously execute each superstep including all compute and communication operations and synchronize between the supersteps. Notice that in case of load imbalance, the faster nodes wait until all nodes reach the synchronization point (horizontal black lines).

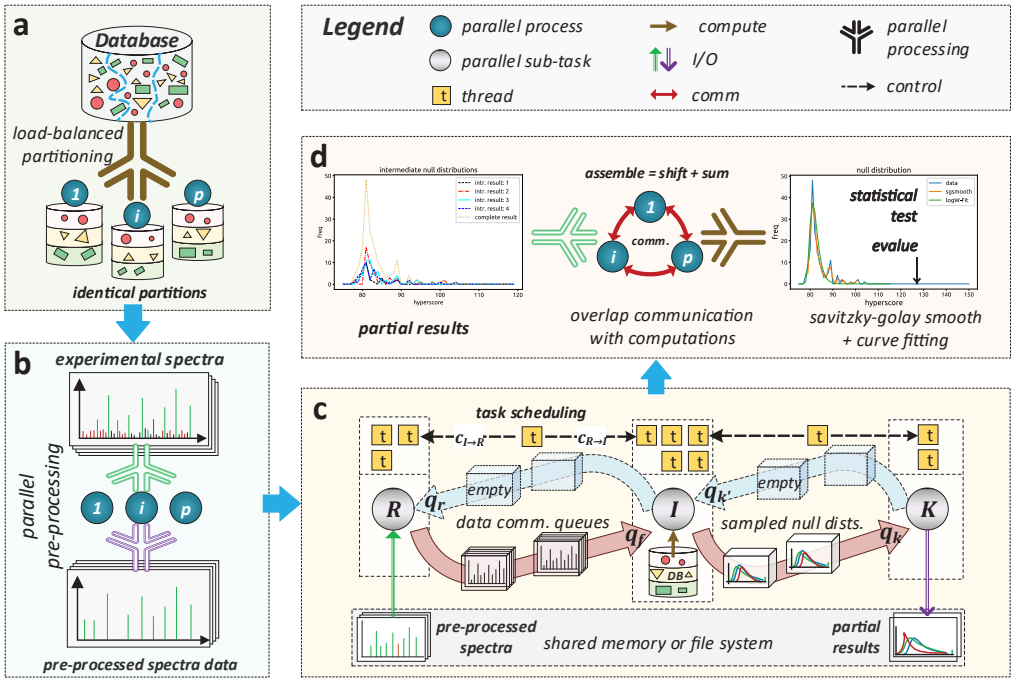

Figure 3.2: **(a)** The theoretical spectra database is partitioned across the parallel nodes using the LBE algorithm and locally indexed. **(b)** The experimental MS/MS spectra data are preprocessed and indexed and written back. **(c)** The experimental spectra are searched against local sub-databases in a pipeline fashion, and the partial results are written to the file system. **(d)** Partial results are assembled into complete results, inter-communicated, and written to the file system [HS21].

3.2.1 Notations and Symbols

For the rest of this chapter, we will denote the indexed theoretical spectra database as $(D = \zeta(2^m))$ where ζ is number of peptide sequences and (m) is the average number of PTMs per peptide sequence. Further, we will represent the experimental MS dataset as $(Q = q_1, q_2, \cdots)$ containing (q) spectra, with average length of (η) split across (b) batches and the total size being $|Q| = q\eta$, the number of parallel nodes as (P), and the number of cores per parallel node as (c_{p_i}). Moreover, the execution time of superstep (j) at the parallel node (p_i) will be denoted as (T_{j,p_i}), and the miscellaneous boilerplate overheads will be captured by (γ_{p_i}). Note that the (indexed) theoretical spectra database will be referred to as simply *the database*.

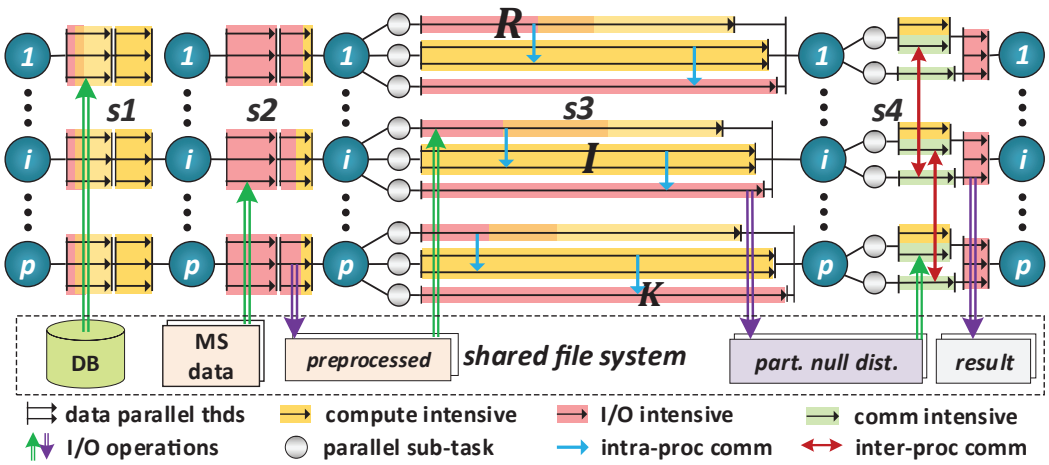

Figure 3.3: The first two supersteps are designed as data-parallel whereas the last two supersteps are designed as hybrid task and data parallel [HS21].

3.2.2 Runtime Cost Model

Since the execution model in HiCOPS is based on SPMD, the runtime cost of the same algorithm at a given node can be modeled by its (local) input size and the amount of available resources. Further, for symmetrical nodes based supercomputers, we will also assume that all nodes have the same resources available.

The algorithmic kernels in HiCOPS are computed in data-parallel and hybrid fashions. The runtime cost of a superstep j on parallel node p, executing an algorithm with $O(D)$ complexity followed by another algorithm with $O(D \log D)$ complexity using all c_{p_i} cores in data parallel fashion will be given as:

$$T_{j,p_i} = k_{j1}(D) + k_{j2}(D) + \gamma_{p_i} \tag{3.2}$$

Here k_{jl} are constant factors. Similarly, if the above two algorithms are executed in the hybrid (i.e., task and data parallel) fashion, the number of cores for each task are also considered. For instance, if both the algorithms in Equation 3.2 are assigned half of the c_{p_i} cores, we can write the runtime cost as:

$$T_{j,p_i} = max(k_{j1}(D, c_{p_i}/2), k_{j2}(D, c_{p_i}/2)) + \gamma_{p_i} \tag{3.3}$$

3.2.3 Superstep 1: Database Indexing

In this superstep, the parallel HiCOPS nodes execute the LBE algorithm [HAS19] to partition the theoretical database among themselves and (locally) index it as shown in Figure 3.2a. This superstep involves the following three algorithmic tasks: **1)** Construct and extract a local partition of the database. **2)** Simulate the local theoretical MS/MS spectra. **3)** Index the local database peptides and theoretical MS/MS spectra to build peptide arrays and the fragment-ion index.

The LBE algorithm [HAS19] used to generate and partition the theoretical spectra data across the parallel HiCOPS nodes in a load-balanced (load imbalance $<$ 10%) fashion is discussed in detail in Chapter 4. To summarize here, the LBE algorithm first clusters the theoretical MS/MS spectra data and then distributes each cluster across the parallel node to optimize the load-balance. The partitioned MS/MS spectra data are locally indexed at each parallel node using the CFIR-Index data structure (discussed in Chapter 5) [HS19] due to its low memory-footprint. To summarize here, the CFIR-Index leverages the low entropy and sparsity in the theoretical MS/MS spectra to compress the resultant fragment-ion index by at least 2×.

Runtime Cost: The parallel nodes generate and partition their sub-databases of size $D/P = D_{p_i}$) in time $O(D)$ as they have to iterate once through roughly the entire database when extracting. Then, the theoretical MS/MS spectra data are generated from the peptides using probabilistic simulation models in time: $O(\tau D_{p_i})$ where τ is average peptide sequence length. The CFIR-Index is then constructed in time: $O(D_{p_i} \log D_{p_i})$. Note that all these steps run sequentially in data parallel fashion resulting in the collective time in Equation 3.4.:

$$T_1 = max_{p_i}(k_{11}(D) + k_{12}(\tau D_{p_i}) + k_{13}(D_{p_i} \log D_{p_i})) + \gamma_{p_i} \tag{3.4}$$

3.2.4 Superstep 2: Experimental Data Preprocessing

In this superstep, the parallel HiCOPS nodes preprocess the experimental MS/MS spectra data using the supplied algorithm in a data parallel fashion as shown in Figure 3.2b. This superstep involves three following algorithmic tasks: **1)** Read the experimental MS/MS dataset from the shared file system, **2)** Pre-process using the supplied algorithm and (optional) index for later use and **3)** Write the pre-processed data back to the shared file system to be used in subsequent experiments.

The experimental MS/MS spectra data are split into smaller, more manageable batches, before writing them back to the file system, to avoid long bursts of I/O times (10,000 spectra per batch by default). The batches are also indexed using running pointer stack to allow quick access in superstep 3, which is also written to the disk for subsequent runs.

Runtime Cost: The parallel nodes read a partition (of size $Q_{p_i} = q\eta/P$) of the experimental MS/MS data in runtime $O(Q_{p_i})$. The pre-processing may be done using a supplied algorithm. e.g., pClean [DRP+19], MS-REDUCE [AS16], or [DSPW09]. Currently, HiCOPS implements a top-K peak filtration algorithm similar to [KLA+17] which runs in time $O(Q_{p_i} \log Q_{p_i})$. Finally, the preprocessed data batches and the index is written back to the disk in time $O(Q_{p_i}$. Note that this superstep is typically skipped in subsequent runs of HiCOPS and only the (few kilo bytes) index is read into the memory pointing to the preprocessed data from the previous runs. Since all steps are sequential data-parallel in this superstep, the collective runtime is given by Equation 3.5:

$$T_2 = max_{p_i}(k_{21}(Q_{p_i}) + k_{22}(Q_{p_i} \log Q_{p_i}) + k_{23}(Q_{p_i})) + \gamma_{p_i} \tag{3.5}$$

3.2.5 Superstep 3: Database Peptide Search

In this superstep, the parallel HiCOPS nodes execute a local database peptide search using the supplied algorithm, currently the fragment-ion search coupled hyperscore, in a hybrid fashion as shown in Figure 3.2c. This superstep involves the following three parallel pipeline algorithmic tasks: **1)** Read batches of the preprocessed experimental MS/MS data from the file system and queue them in the pipeline, **2)** Search the queued batches against the local sub-database using the supplied algorithm and (separately) queue the results and distributions, and **3)** Process the queued results by sampling and encoding them and writing to the file system.

The three algorithmic tasks in this superstep are executed by respective multithreaded sub-tasks namely R, I, and K in a producer-consumer pipeline fashion as shown in Figure 3.2c [HS21]. The sub-tasks communicate data with each other using two sets of bidirectional buffer queues. The number of assigned parallel cores to each sub-task at any time are given by: $\mid r \mid$, $\mid i \mid$ and $\mid k \mid$.

Runtime Cost: The sub-task R reads the batches of experimental data in time $O(q\eta)$. The sub-task I first executes the fragment-ion search in time $O(q \log(D_{p_i})) + O(q\eta\alpha_{p_i})$ where α_{p_i} is the average number of fragment-ion matches per experimental fragment-ion searched. Then, the spectral similarity computations (hyperscores) are computed in $O(q\mu)$ time where μ is the average number of candidate peptide matches, or peptide-to-spectrum matches (PSM)s, per experimental MS/MS spectrum searched. The sub-task K writes the partial results to the shared file system in time $O(q)$. Since these sub-tasks run in a hybrid-fashion, their collective runtime will be the total time for the pipeline given as:

$$T_3 = max_{p_i}(max(k_{30}(q\eta, \mid r \mid), k_{31}(q \log(D_{p_i}), \mid i \mid) + k_{32}(q\eta\alpha_{p_i}, \mid i \mid) + k_{33}(q\mu_{p_i}, \mid i \mid) + k_{34}(q, \mid k \mid))) + \gamma_{p_i} \tag{3.6}$$

Overhead Costs: Overhead costs originating from memory congestion, load-imbalance, pipeline halts may significantly affect the runtime of this superstep. Therefore, it is pertinent to capture them using an additional factor: $V_{p_i}(q, D_{p_i}, P)$ and optimize them using the techniques discussed in Section 3.4. The Equation 3.6 with the overheads can be written as:

$$\begin{aligned} T_3 = max_{p_i}(max(k_{30}(q\eta, \mid r \mid), k_{31}(q\log(D_{p_i}), \mid i \mid) + k_{32}(q\eta\alpha_{p_i}, \mid i \mid) + \\ k_{33}(q\eta\sigma_{p_i}) + k_{33}(q\mu_{p_i}, \mid i \mid) + k_{34}(q, \mid k \mid))) + V_{p_i}(q, D_{p_i}, P) + \gamma_{p_i} \end{aligned} \tag{3.7}$$

3.2.6 Superstep 4: Result Assembly and Postprocessing

In this superstep, the parallel HiCOPS nodes assemble the partial peptide search results computed in superstep 3 and postprocess them for confidence scores in a hybrid fashion as shown in Figure 3.2d. This superstep involves the following three algorithmic tasks **1)** Read subsets of partial results, assemble, compute confidence scores, and communicate them to the origin nodes, **2)** Receive final results from other nodes, and **3)** Write the final results to the shared file system. The algorithmic tasks in this superstep are executed using two multithreaded sub-tasks. The first sub-task reads portions of partial results from the shared file system, assembles them, and computes the confidence scores using regression and curve fitting techniques [FB03], [HS21] as shown in Figure 3.2d. The confidence scores along with origin information (16 bytes) is accumulated in a hashmap of P packets (one for each node) and on completion, are communicated in one all-to-all communication load. The second sub-task using 2 oversubscribed threads simply waits for this transmission for the other $P-1$ nodes and then writes the received results to the file system.

Runtime Cost: The first sub-task reads the partial results, assembles, computes confidence score, and sends the final results to other nodes in time: $O(Q_{p_i}, c_{p_i})+$ $O(Q_{p_i}, c_{p_i}) + O(P, 1)$. The second sub-task waits in sleep to receive the final results

from other nodes in runtime: $O(P,1)$ and writes them to the shared file system in runtime: $O(Q_{p_i})$. The complete runtime for this superstep can be written as:

$$\begin{aligned} T_4 = max_{p_i}(max(k_{41}(Q_{p_i}, c_{p_i}) + k_{42}(Q_{p_i}, c_{p_i}) + \\ k_{43}(P,1), k_{44}(P,1)) + k_{45}(Q_{p_i})) + \gamma_{p_i} \end{aligned} \tag{3.8}$$

Equation 3.8 can be simplified by writing it as the sum of computation and communication costs ($k_{com}(P,1)$) as:

$$\begin{aligned} T_4 = max_{p_i}(k_{41}(Q_{p_i}, c_{p_i}) + k_{42}(Q_{p_i}, c_{p_i}) + \\ k_{com}(P,1) + k_{45}(Q_{p_i})) + \gamma_{p_i} \end{aligned} \tag{3.9}$$

3.3 Performance Analysis

The parallel performance of HiCOPS can be theoretically analyzed by quantifying its runtimes into parallel (T_p), serial (T_s) and overhead runtimes (T_o) as:

$$T_H = \sum_{j=1}^{4} max_{p_i}(T_{j,p_i}) = T_o + T_s + T_p \tag{3.10}$$

Using equations 3.1, 3.4, 3.5, 3.7, and 3.9, we quantify the three runtimes as:

$$T_o = V_{p_i}(q, D_{p_i}, P) + \gamma_{p_i} \tag{3.11}$$

$$T_s = k_{11}(D) + k_{21}(q\eta) + k_{com}(P,1) \tag{3.12}$$

and:

$$\begin{aligned} T_p = k_{12}(D_{p_i}) + k_{13}(D_{p_i} \log D_{p_i}) + k_{22}(Q_{p_i}) + k_{23}(Q_{p_i}) + \\ max(k_{30}(q\eta, \mid r \mid), k_{31}(q \log(D_{p_i}), \mid i \mid) + k_{32}(q\eta\alpha_{p_i}), \mid i \mid) + \\ k_{33}(q\mu_{p_i}, \mid i \mid), k_{34}(q, \mid k \mid)) + k_{41}(Q_{p_i}, c_{p_i}) + \\ k_{42}(Q_{p_i}, c_{p_i}) + k_{45}(Q_{p_i}) \end{aligned} \tag{3.13}$$

T_s is the serial time, which is not optimizable so, we will remove it from our analysis and focus on the other two components: $T_F = T_p + T_o$. Using equations 3.11 and 3.13, we have:

$$\begin{aligned} T_F = k_{12}(D_{p_i}) + k_{13}(D_{p_i} \log D_{p_i}) + k_{22}(Q_{p_i}) + \\ k_{23}(Q_{p_i}) + max(k_{30}(q\eta, \mid r \mid), k_{31}(q \log(D_{p_i}), \mid i \mid) + \\ k_{32}(q\eta\alpha_{p_i}), \mid i \mid) + k_{33}(q\mu_{p_i}, \mid i \mid), k_{34}(q, \mid k \mid)) + \\ k_{41}(Q_{p_i}, c_{p_i}) + k_{42}(Q_{p_i}, c_{p_i}) + k_{45}(Q_{p_i}) + T_o \end{aligned} \tag{3.14}$$

As supersteps 1 and 2 are fairly parallelized and superstep 4 does not contribute much to the execution time, we can prune the respective terms to simplify the analysis:

$$\begin{aligned} T_F = max(k_{30}(q\eta, \mid r \mid), k_{31}(q \log(D_{p_i}), \mid i \mid) + k_{32}(q\eta\alpha_{p_i}), \mid i \mid) + \\ k_{33}(q\mu_{p_i}, \mid i \mid), k_{34}(q, \mid k \mid)) + T_o \end{aligned}$$

Notice that the sub-task R (producer) will always complete before its consumers so can be removed resulting in:

$$\begin{aligned} T_F = max(k_{31}(q \log(D_{p_i}), \mid i \mid) + k_{32}(q\eta\alpha_{p_i}), \mid i \mid) + k_{33}(q\mu_{p_i}, \mid i \mid), \\ k_{35}(q, \mid k \mid)) + T_o \end{aligned}$$

In above equation, we rewrite the max term as the database search time (sub-task I) plus the overhead time $t_x(k)$ to complete the pipeline (sub-task K). Therefore, using equation 3.6 we have:

$$\begin{aligned} T_F = k_{31}(q \log(D_{p_i}), \mid i \mid) + k_{32}(q\eta\alpha_{p_i}), \mid i \mid) + \\ k_{33}(q\mu_{p_i}, \mid i \mid) + t_x(k) + T_o \end{aligned} \tag{3.15}$$

Further, in equation 3.15, we can prune the first two terms as their $O(\log(.))$ times are relatively negligible. Finally, using equation 3.11 in 3.15, we have:

$$T_F = k_{33}(q\eta\alpha_{p_i}, \mid i \mid) + k_{34}(q\mu_{p_i}, \mid i \mid) + t_x(k) + V_{p_i}(q, D_{p_i}, P) + \gamma_{p_i} \tag{3.16}$$

3.4 Optimizations

In this section, we will discuss the optimization techniques to minimize the overhead terms in Equation 3.16.

3.4.1 Buffer Queues

Two sets of bidirectional buffer queues are initialized ($\{q_f, q_r\}, \{q'_f, q'_r\}$) are initialized between the producer-consumer sub-tasks R, I and K in the superstep 3. q_r is populated with 20 (default) empty buffers which are filled with the experimental data by R pushed to q_f. I pops the buffers from q_f, processes them, and recycles them back into the q_r while also pushing the results into q'_f. Similar process is done at sub-task K as well. Three compute regions are defined using q_f depending on its fullness at any time. i.e. $w_1 : (|q_f| < 5)$, $w_2 : (5 \leq |q_f| < 15)$ and $w_3 : (|q_f| \geq 15)$ to allow the core management algorithm (task-scheduling algorithm) to optimally reallocate the cores between the sub-tasks.

3.4.2 Task Scheduling Algorithm

HiCOPS employs a forecasting-based task-scheduling algorithm to maintain a synergy between the producers and consumers in superstep 3 and eliminating any pipeline halts. The algorithm begins with a thread pool of $c_{p_i} + 2$ parallel threads and assigns 2 threads to R and K each, while assigning the rest to I. Then, in each iteration, the q_f compute regions w_i, and the pipeline halt times (t_{wait}), and the cumulative pipeline halt (t_{cum}) are measured and recorded in a time series. Using this data and double exponential smoothing [LaV03], the halt time in the next iteration is also forecasted (t_{fct}). Then, based on the relative speed of the producers

and consumers, the cores are assigned between R and I using Algorithm 1. Note that the sub-task K is assigned 2 oversubscribed threads in the beginning and does not participate in task-scheduling.

Algorithm 1: Superstep 3: Task Scheduling [SHI22]

Data: buffer queue: q_f, halt time (t_{wait}), minimum halt time (t_{min}), surge (t_{surge}) and cumulative halt (t_{acc})
Result: thread allocation between sub-tasks R, and I

```
/* cumulative halt time                                          */
1  t_cumu ← 0
/* while experimental data available                             */
2  while (b_rem > 0) do
       /* if pipeline halt time > t_min                          */
3      if t_halt ≥ t_min then
4          t_cumu ← t_cumu + t_halt;
5          if t_cumu > t_acc ∨ t_halt > t_surge then
6              MoveThread(I, R);
7              t_cumu ← 0;
8      else
9          if q_F.size() < 5 ∧ | r |= 0 then
10             MoveThread(I, R);
11             t_cumu ← 0;
12         else if q_F.size() > 15 ∧ | r |> 1 then
13             MoveThread(R, I);
14 return MoveAllthreads(R, I);
```

3.4.3 Load Balancing

HiCOPS employs the LBE algorithm for load-balancing the database partitions across the parallel nodes. The LBE algorithm is discussed in detail in Chapter 4.

3.4.4 Sampling

HiCOPS employs a sampling technique to minimize the communication and I/O costs in the supersteps 3 and 4 without deteriorating the results quality. In the employed technique, the null distribution along with its metadata (2KB) produced for each searched experimental MS/MS spectrum in superstep 3, is sampled to reduce the communication overhead. To do this, HiCOPS uses the data distribution information (i.e., the Gumbel distribution [FB03]) to sample in a way that the original distribution could be recovered in smoothing and curve fittings employed in superstep 4. To do this, the distribution mean is first computed and then $s = 120$ samples are picked around it by prioritizing the head (unskewed and sharp slope) of the distribution over the tail (skewed and slow decay). The sampled data are then encoded into `ushort`, which along with the meta data constitute 256 bytes per experimental MS/MS spectra (16× improvement). Figure 3.4 illustrates an example of sampling.

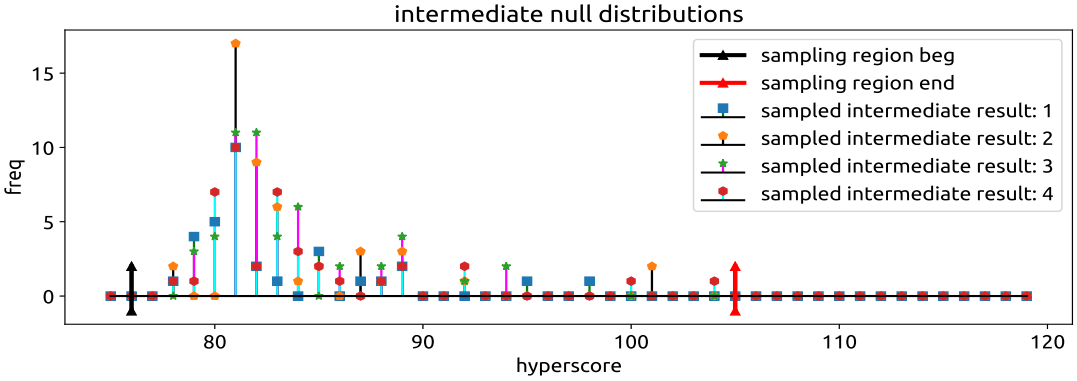

Figure 3.4: The mean is roughly computed by computing the mean of the 3 or 5 most intense samples in the distribution (81 in this figure) and then $s = 120$ data points are sampled around the mean prioritizing the head of the distribution as the tail can be recovered in superstep 4 during curve smoothing and regression [HS21].

3.5 Results

3.5.1 Experimental Setup

We used five datasets (E_i) constructed by unionizing multiple publicly available Pride Archive datasets for our experimentation and evaluation. The details of these experiments are as follows: E_1: PXD009072, E_2: PXD020590, E_3: PXD015890 E_4: PXD007871, $\cup$ PXD009072 $\cup$ PXD010023 $\cup$ PXD012463 $\cup$ PXD013074 $\cup$ PXD013332 $\cup$ PXD014802 $\cup$ PXD015391, and E_5: $E_1 \cup E_2 \cup E_3 \cup E_4$. The datasets were searched against PTM variants of the databases constructed from D_1: UniProtKB Homo sapiens and D_2: UniProtKB SwissProt (reviewed) databases. The database was digested in-silico using Trypsin with peptide lengths from 6 to 46, allowed missed cleavages up to 2, and peptide masses from 500 to 5000Da. Cysteine carbamidomethylation was used as static modification for all experiments, and a maximum of 5 variable PTMs per peptide were chosen from combinations of M-oxidation, NQ-deamidation, STY-phosphorylation, CK-gly-gly adducts, and Y-biotin-tyramide. The closed-search criterion was set to $\delta M \leq \pm$10Da and the opens-search criterion was set to $\delta M \leq \pm$500Da unless mentioned otherwise. We intentionally set the closed-search criteria to a few Daltons instead of ppms to cover the peptide mass differences due to average or monoisotopic masses and isotopes used across different tools. The preprocessing settings for the experimental MS/MS spectra were set to the minimum in all tools so that the same amount and nature of the data (fairness) are used. Some of these settings include: precursor charges up to +4, minimum 4 matched peaks for candidacy, picking only top 100 (or 150 in some experiments) peaks by intensity for processing. Any and all other preprocessing settings including calibrations, peak transformations, precursor peak removal, clipping N-term M and partial spectrum re-construction were disabled when possible.

Runtime Environment: All experiments were run on the Comet cluster via the Extreme Science and Engineering Discovery Environment (XSEDE) [TCD+14] program. The Comet compute nodes consist of 12 cores × 2 Intel Xeon E5-2680v3, 64GB DRAM × 2 NUMA nodes, 56 Gbps Infiniband interconnect. The maximum job allocation is 72 nodes for 48 hours. Note that the XSEDE program has now been concluded and replaced by Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which provides access to the Expanse cluster which recently replaced the now-decommissioned Comet cluster. Note that the single-node Crux, X!Tandem, and MSFragger experiments needing more than 48 hours to complete were run on a local server named *raptor*, consisting of a 22 core Intel Xeon Gold processor, 128GB RAM and 6TB storage.

3.5.2 Correctness Analysis

We measured the correctness of HiCOPS using a two-fold approach. First, we measured the correctness across the degree of parallelism. i.e., correctness across serial and parallel runs. For this, we ran all five datasets E_i against several databases of varying sizes. The results were said to be correct if the identified peptide sequences were the same and the computed scores were within 2 decimal points. Our experiments in Figure 3.5a and 3.5b depict that the HiCOPS depicts correct and consistent results (>99.5% consistency) across serial and parallel runs. A negligible number of results depicted small negative errors due to sampling and floating point errors.

Second, we measured the correctness across tools. For this, we ran three different experiments first in closed- ($\delta M = 1Da$) and then open-search ($\delta M = 200Da$) modes using HiCOPS and MSFragger as MSFragger also employs a similar fragment-

ion search coupled hyperscore algorithm (fairness). The first experiment involved searching a subset of 860K spectra from E_4 against D_1 with M-oxidation and NQ-deamidation (size: 18 million). The second experiment involved searching the dataset: S_3 against the database: D_1 with M-oxidation and STY-phosphorylation (size: 66 million). The third experiment involved searching the dataset: S_3 against the database: D_2 with M-oxidation and S-phosphorylation (size: 80 million). Our experimental results for the three experiments in closed search in Figures 3.5c to 3.5e respectively show a Pearson Correlation coefficient < 0.90 for the 1% FDR filtered results, depicting strong correlation. On the other hand, the correlation drops to about 0.78 for the open-search versions of the three experiments in Figures 3.5f to 3.5h respectively. We suspect that the divergence in the results from MSFragger and HiCOPS originated from the open-search specific (proprietary and closed-source) MSFragger components.

3.5.3 Speed Comparison Against Existing Algorithms

We measured the speed improvement provided by HiCOPS over several existing HPC database peptide search tools including Crux/Tide [MTKF+14], Comet v2020.01 [EJH13], MSFragger v3.0 [KLA+17], X! Tandem v17.2.1 [CB04], X!! Tandem v10.12.1 [BCC+07], and SW-Tandem [LLC+19]. Notice that the Comet, Tide and MSFragger are not HPC tools by themselves so we implemented a runner Python script which emulates the parallelization techniques employed by the existing HPC database peptide search tools [DCL05], [BCC+07], [LLC+19], involving partitioning of the experimental data across the parallel nodes and simultaneously launching instances of these tools on those nodes (also discussed in Section 2.3.2).

We ran six increasing workload sized experiments to measure the speed improve-

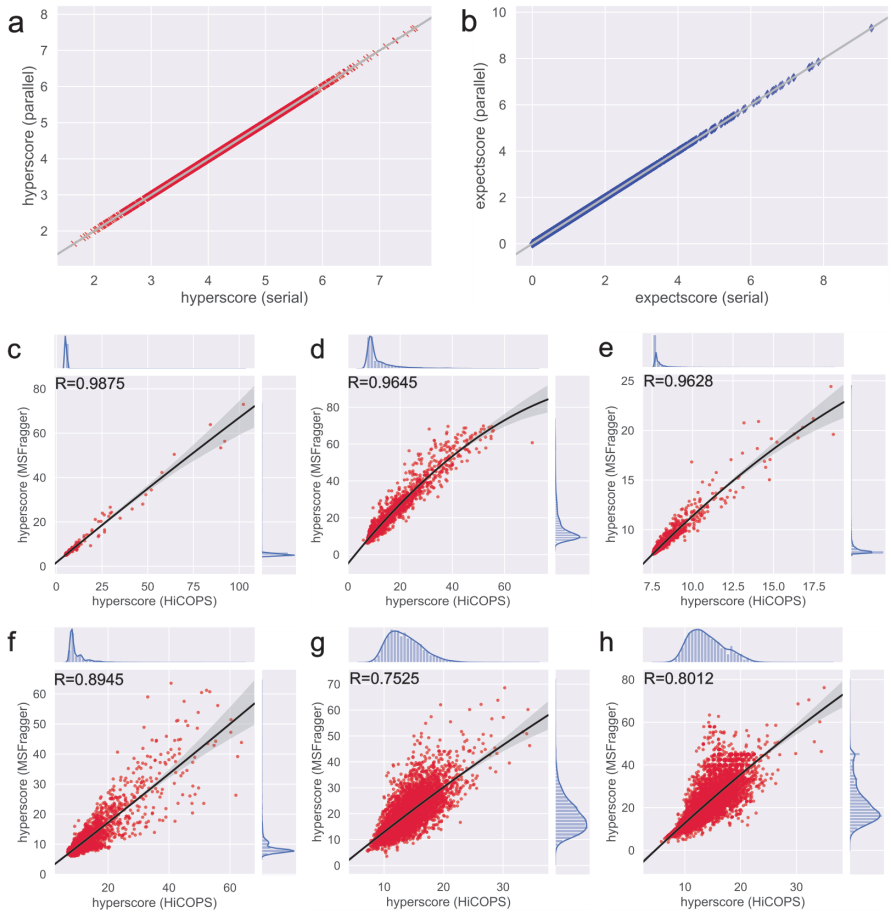

Figure 3.5: **(a,b)** 5K samples out of 251K data points are shown for computed scores from HiCOPS in serial and parallel runs. **(c to e)** Correlation between MSFragger and HiCOPS scores is about 0.90 for closed-search experiments. **(d to f)** Correlation between MSFragger and HiCOPS scores is about 0.78 for open-search experiments [HS21].

ment of HiCOPS. The first and second experiments involved searching a subset of 8K spectra from dataset: S_3 against D_2 with M-oxidation, and Y-Biotin-tyramide (size: 93.5 million) at δM=10Da (first experiment, Figure 3.6a) and δM=500Da (second experiment, Figure 3.6b). The third and fourth experiments involved searching the dataset: S_3 against D_1 with M-oxidation, and Y-Biotin-tyramide (size: 7.1 million spectra) at δM=10Da (third experiment, Figure 3.6c) and δM=500Da (fourth experiment, Figure 3.6d). The fifth and sixth experiments involved search-

ing the dataset: S_4 against D_1 with M-oxidation, STY-phosphorylation and NQ-deamidation (size: 213 million spectra) at δM=10Da (fifth experiment, Figure 3.6e) and δM=100Da (sixth experiment, Figure 3.6f).

Our experimental results (Figure 3.6a to 3.6f) depict that the HiCOPS outperforms all other tools by at least $> 10\times$ regardless of the number of nodes and experimental sizes. HiCOPS also exhibits better strong-scale efficiency (shown as the deviation from the dotted black lines (positive = sub-linear; negative = hyper-linear)) than that of other tools as the experimental sizes increase (from a$\rightarrow$f). Further, MSFragger depicted a peculiar trend in runtimes as it behaves superlinearly at first and then drops. We investigated this by analyzing the runtime components of MSFragger and the results in Figure 3.6g and 3.6i show that the I/O and load imbalance overheads are significant in MSFragger. In contrast, HiCOPS depicts little overheads as seen in Figure 3.6h and 3.6j. Finally, the results in Figures 3.6a to 3.6c show that the other HPC tools are $> 100\times$ slow even for small experiments.

3.5.4 Performance Evaluation

Experimental Setup

We measured the parallel performance of HiCOPS using 12 extensive sets of experiments curated from the five datasets and databases by varying the number and combinations of PTMs and the peptide precursor mass tolerances (δM) to cover a wide-range of real-world application.

Parallel Scalability

We measured the parallel efficiency for all 12 experiments. The experimental results in Figures 3.7a and 3.7b depict an overall efficiency between 70 to 80% for larger

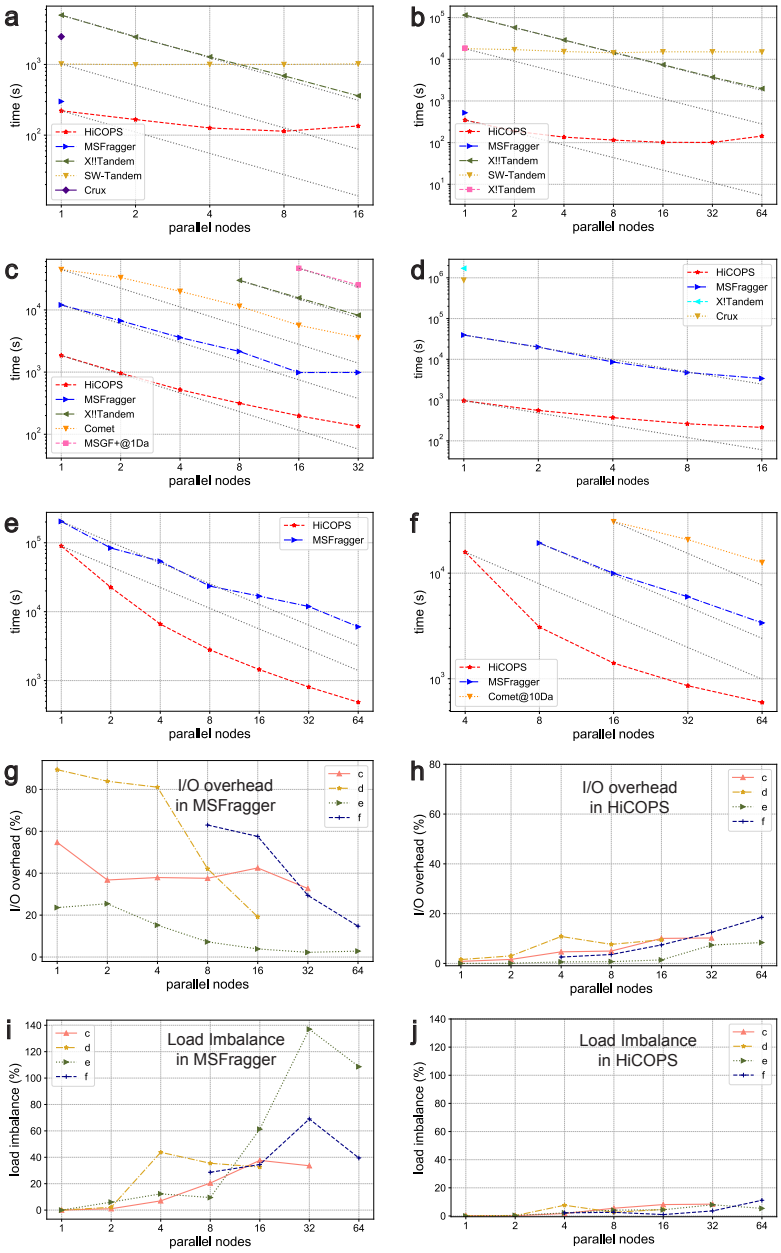

Figure 3.6: **(a to f)** HiCOPS's speed is compared with several existing tools for increasing parallel nodes. The δM window was reduced for MSGF+ and Comet (shown as @) for feasibility **(g to i)** Percentage overheads for HiCOPS and MS-Fragger for experiments in (**c** to **f**) are shown [HS21].

experiments and between 30 to 50% for smaller experiments (Amdahl's Law). As the minimum HiCOPS's nodes (P_{min}) are limited to $P_{min} \geq D/M$; where M is the

RAM per node, the computed strong-scale efficiencies used the experiments with P_{min} nodes as the base case. Hyperlinear speedups were also seen in large-scale experiments due to the corresponding improved hardware performance including CPU and cache performance as seen in Figures 3.7c to 3.7f. We also measured the superstep-by-superstep times for these 12 experiments and the results in 3.8 show that the superstep 3 dominates the total runtime for larger experiments.

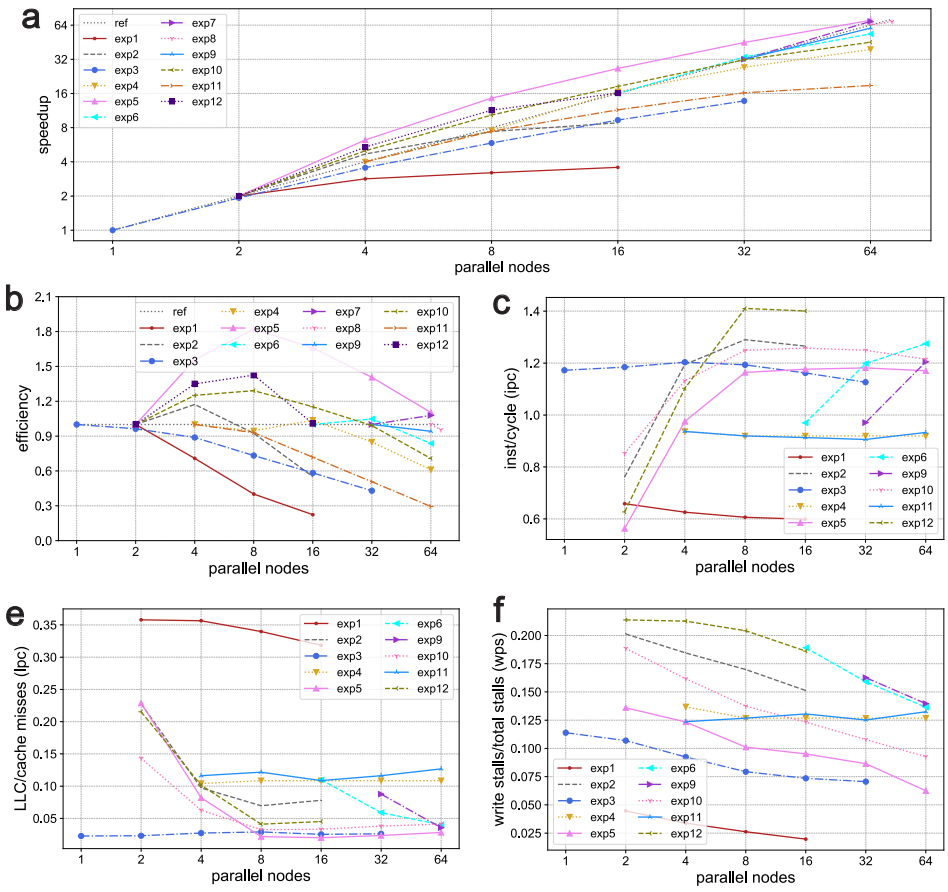

Figure 3.7: **(a), (b)** Parallel speedup and strong-scale efficiency for the 12 experiments is shown for HiCOPS. Note that the dotted black lines show the ideal speedup and efficiency respectively. **(c) to (f)** Hardware utilization metrics for the 12 experiments are shown for HiCOPS [HS21].

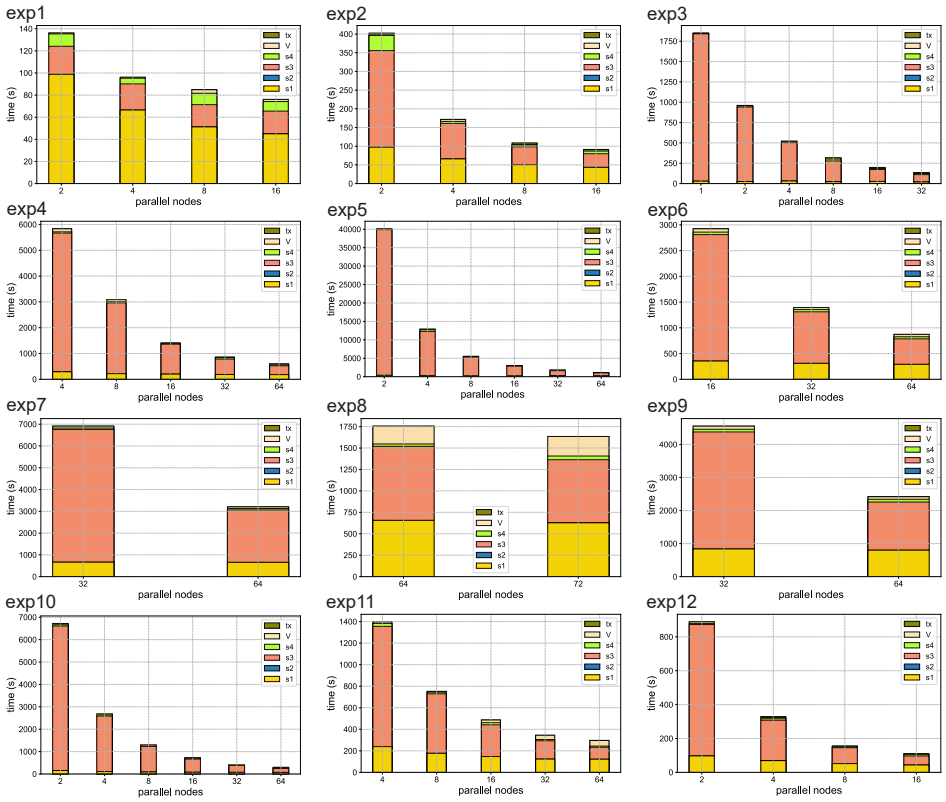

Figure 3.8: The runtime breakdown across the four HiCOPS supersteps and overhead metrics in Equation 3.10for the 12 experiments is shown depicting the large contribution of superstep 3 in the total execution times, especially in open-search experiments that depict superlinear strong-scale efficiency as well [HS21].

Performance Overheads

We measured several overheads metrics including load balance, communication, I/O, and pipeline halt stalls to quantify the severity of performance overheads in HiCOPS. The experimental results in Figures 3.9a to 3.9c depict that the sum of all the overheads remains $< 25\%$. The performance of our task-scheduling algorithm is also seen in the time series in Figure 3.9e plotting the pipeline halt times. It can be seen that our task-scheduling algorithm instantly performs measures to eliminate the stalls as soon as they are discovered, if any.

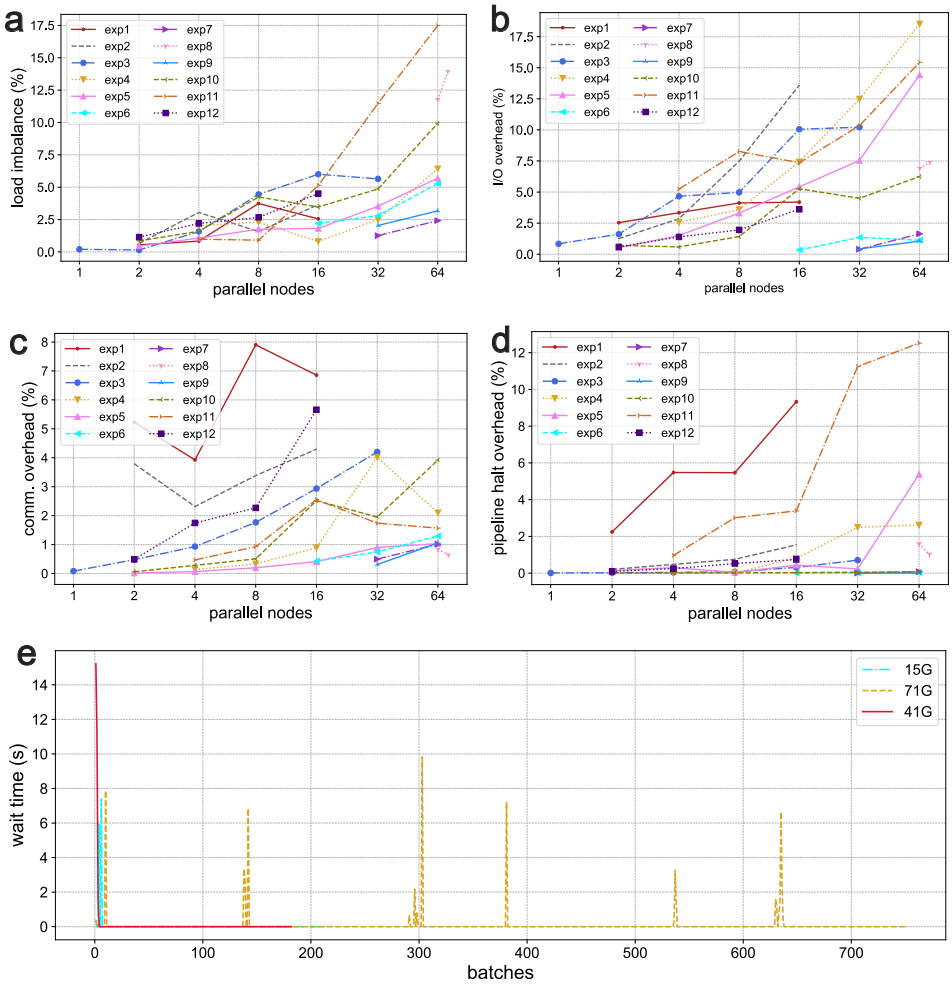

Figure 3.9: **(a) to (d)**The combined performance cost of several overhead metrics is shown to be less than 25% in HiCOPS. Pipeline stall timeline is shown depicting the active performance of the task-scheduling algorithm [HS21].

3.6 Summary

Open-search database peptide search algorithms require astronomical amounts of computational resources to feasibly search several gigabytes of experimental MS/MS spectra data against terabytes of indexed database. Shared-memory computers are unable to meet these resource demands. Moreover, the post-Moore shared-memory computers are bottle-necked by memory-bounds for open-search database peptide search experiments. While these resource demands could be met by modern su-

percomputers, the HPC algorithms and methods deployed in the existing shared- and distributed-memory database peptide search software infrastructure deliver sub-optimal performance. This is because they are designed for compute-intensive algorithms and do not optimize for the memory-intensity of the algorithms thereby replicating the same memory-bounds on all supercomputer nodes. HiCOPS presents a novel Bulk Synchronous Parallel (BSP) based parallelization model as well as optimizations that efficiently distribute both the compute and memory loads across the supercomputing nodes, achieving over 10× speedup over all existing HPC tools at a strong-scale efficiency between 70-80%. Further, the algorithms and optimizations presented in HiCOPS are algorithm-oblivious allowing it to be adaptable for future traditional and machine and deep learning based database peptide search algorithms. Our results show that the database peptide search experiments that require over 6 months of execution time using single-node algorithms can be executed in about 2 hours by HiCOPS using 72 nodes.

CHAPTER 4

LBE: LOAD BALANCED DATABASE PARTITIONING

In this chapter, we will discuss our efficient and intuitive algorithm, called *LBE* [HAS19], for load balanced partitioning and distributing the theoretical MS/MS spectra databases, or simply the *databases*, across a symmetric distributed-memory HPC cluster. We will also discuss its effectiveness in achieving system load balance in HiCOPS, analyze its performance, as well as its application in extending HiCOPS for heterogeneous compute clusters.

4.1 Introduction

In stark contrast to the previous HPC database peptide search algorithms, HiCOPS partitions the database across a symmetrical compute cluster to distribute its computational workload and achieve massive parallelism, as discussed in Chapter 3. This is because distributing, instead of replicating, the voluminous databases across the parallel nodes alleviates the memory pressure per node yielding superior throughput as also shown in chapter 3. However, to achieve the maximum parallel throughput in HiCOPS, its compute workload must be partitioned evenly across the system nodes especially when a large number of nodes are involved [HAS19]. To understand this, consider a 32 node parallel system executing HiCOPS with each node completing its execution in $T_{avg} = 100s$, with straggler node(s) finishing execution in $T_{max} = 150s$. Then, on average, each compute node spent $50s$ waiting for the stragglers to complete resulting in a wasted CPU time of $T_{wst} = 1600s$ or a $16\times$ performance degradation.

4.2 The LBE Algorithm

Before delving into the details of the LBE algorithm, let us first define the computational workload in HiCOPS which is to be balanced across the system. From Equation 3.16, the computational workload in HiCOPS is proportional to the cost of the executing the fragment-ion search plus the spectral similarity score computations. The results in Figure 3.8 also corroborate this estimation. Using this definition, the LBE algorithm design strives to ensure that for a random experimental MS/MS spectrum, the number of fragment-ion matches and similar theoretical spectra (i.e., the computational workload) seen at *all* systems nodes is roughly identical. In other words, the LBE algorithm strives to partition the database across the parallel nodes such that the sub-database at each node is near-identical, implying that the computational time to search a random experimental MS/MS spectrum against any sub-database (approximately equal to the number of database matches) is also near-identical.

4.2.1 LBE Algorithm Overview

To achieve this, the LBE algorithm employs a two-step approach to (load balanced) partition the HiCOPS's database across the symmetric parallel nodes. In the first step, the LBE algorithm clusters the similar database peptides, both normal and PTM modified, using a heuristics based technique. In the second step, each peptide cluster is split across the parallel nodes in either round-robin (default), random, or chunked fashion as shown in Figure 4.1. The mathematical theory along with the details of each of the two steps in LBE are explained in the following sections.

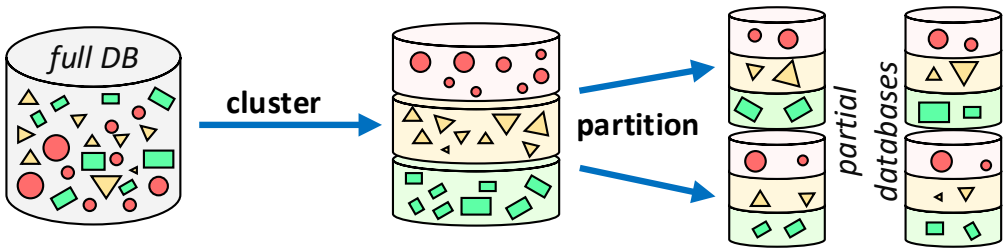

Figure 4.1: The LBE algorithm first constructs clusters of similar database elements which are round-robin partitioned across the system nodes to construct near-identical (similar) sub-databases at each node [HS21].

4.2.2 Correctness of the LBE

Theorem. *Let the data distribution of a HiCOPS's theoretical database is $g(x)$ where m is the peptide precursor mass, then the LBE algorithm will yield load balanced database partitions across the system nodes [HS21].*

Proof. The algorithmic workload $w(q, g)$ to search an experimental MS/MS spectrum (q) of mass (m) against the database $g(x)$ is the cost of filtering the database and computing spectral similarities sim given as:

$$w(q, g) = cost(sim(q, filter(q, g(m)) \tag{4.1}$$

The above equation can be expanded for the entire experimental dataset Q as:

$$w(Q, g) = cost(\sum_{q \in Q} sim(q, filter(q, g(m)) \tag{4.2}$$

In HiCOPS, the $filter$ is replaced by the fragment-ion search kernel with tolerances δM, δF resulting in:

$$w(Q, g) = cost(\sum_{q \in Q} q. \sum_{z=-\delta M}^{z=\delta M} fragIon(q, g(m + z), \delta F) \tag{4.3}$$

Equation 4.3 implies that in order for the computational cost $w(Q, g)$ to be balanced across the system nodes in HiCOPS, the data distribution of the partitioned

sub-databases $g(x)$ must also be identical to yield identical ($\sum fragIon(q, g(m + z), \delta F)$) at all nodes. Note that the LBE algorithm can also model other database filtration methods in Equation 4.3. □

4.2.3 LBE Clustering

Distance Metric

Unlike experimental MS/MS spectra data clustering, the LBE algorithm clusters the database peptide sequences, both normal and modified. Since similar peptide sequences result in similar corresponding MS/MS spectra, the algorithm need not to move around and cluster large volumes of complex (floating point) data. Since the peptide sequences are essentially character strings, it is natural to use a normalized Edit Distance, represented by Δe, as the main distance metric. However, the Edit Distance itself is not sufficient to efficiently separate the peptide sequence pairs. This is because in the context of peptide sequences and their corresponding MS/MS spectra, the distance between a pair of peptide sequences with $\Delta e \geq 2$ also depends on the location of those edits. To solve this, we introduce a secondary distance metric, called Mod Distance (ΔM), allowing better clustering of peptide sequences leading to better load balance. The Mod Distance (Δm) is defined as follows:

Definition. *Given a pair of simulated spectra (p, q), the number of ion-series to be generated is s, and the sum of lengths of unmodified amino acid sequences in each ion-series is aa, the Mod Distance (Δm) is given as:*

$$\Delta m(x, y) = s - \frac{aa}{max(|x|, |y|)}$$

The application of the Mod Distance is best explained using an example. Let three peptide sequences be p: PEPTIDE, q: PPPTIKE and r: PEPYKDE. Here,

the blue and red letters are the unedited and edited amino acids, respectively. In this example, $\Delta e(p,q) = \Delta e(p,r) = 2$ does not differentiate between the two pairs. Applying Δm to this, will allow correctly setting the pair (p,q) farther than the pair (p,r) due to the edit locations being far from either terminal in the latter one. This can also be seen in their corresponding theoretical MS/MS spectra, with b- and y-series ($s = 2$), where p and q share: PPPTIKE = 2 ions, yielding $\Delta m(p,q) = 1.42$, and the p and r share: PEPYKDE = 5 ions, yielding $\Delta m(p,r) = 0.57$.

Clustering Algorithm

The LBE algorithm employs a single-pass heuristics based algorithm for clustering. The algorithm begins by reading in all peptide sequences and generating their PTM mutated versions. These sequences, both normal and mutated, are then lexicographically sorted to form the initial clusters. Then, a single pass algorithm is employed to form the peptide clusters as follows. Choose the first encountered sequence s_i as the centroid of a new cluster and add all subsequent peptide sequences s_j to the cluster that satisfy: $\Delta e(s_i, s_j) + \Delta m(s_i, s_j) \leq min(d, len(s_j)/2$ (default: $d = 3$). Another criterion employed is: $(\Delta e(s_i, s_j) + \Delta m(s_i, s_j))/max\{len(s_i), len(s_j)\} \leq d'$ (default: $d' = 0.86$). The same algorithm repeats until all clusters are formed as shown in Algorithm 2.

4.2.4 LBE Cluster Partitioning

The clustered peptide sequence data are read by all system nodes where each node extracts a distinct partition of each cluster based on the partitioning policy. The LBE algorithm supports several partitioning policies including chunk, random, cyclic (default), and zigzag. Our empirical experiments showed that the choice of policy

Algorithm 2: The LBE Clustering [HAS19]

```
   Data: Peptide sequences (Li)
   Result: LBE Clusters (Lz)
   /* lexicographical sort                                          */
1  Li.LexSort();
   /* start the first cluster                                       */
2  Lz.Append([Li[0]]);
3  for k ∈ size(Li) do
       /* distance metrics                                          */
4      dist = Δe(seq, Li[k]) + Δm(seq, Li[k]);
       /* compute the cluster cutoff - either one                   */
5      c1 ← max(d, len(Li[k])/2);
6      c2 ← dist/max(len(seq), len(Li[k])) ≤ d';
       /* check conditions                                          */
7      if (dist > cx) ∨ (Lz[size(Lk)] = csize) then
           /* start a new cluster                                   */
8          Lz.Append([seq]);
9      else
           /* append to the current cluster                         */
10         Lz[−1].Append(Li[k]);
11 return Lz;
```

here does not significantly affect the final load balance results as long as the clusters are well formed in the last step. Once all system nodes extract their partitions, they generate the theoretical MS/MS spectra data and index it to form their sub-databases, to be used in the subsequent database peptide search steps.

4.3 Results

The same experimental setup including the databases, experimental datasets, digestion and pre-processing settings employed for HiCOPS (explained in Section 3.5.1) was also used to evaluate the LBE algorithm as well. We ran the same twelve performance evaluation experiments explained in the Section 3.5.4 to evaluate the performance of the LBE algorithm and its application in HiCOPS.

4.3.1 Load Imbalance with and without the LBE

We measured the load balancing factor of the LBE algorithm by comparing the load-imbalance in the sub-databases constructed with and without the LBE algorithm. To measure the load imbalance, we searched the experimental MS/MS dataset: E_1 against increasing size databases split across 16 sub-databases and measured the load imbalance between the MPI processes (or sub-database). The load imbalance factor employed in the analysis is defined as follows.

Definition. *Given the average compute time for all processes to complete the search is T_{avg} and the maximum time is T_{max}, then the Load Imbalance (LI) is defined as the ratio of the (+ve) maximum deviation in compute time ($T_{max} - T_{avg}$) to the average compute time [HAS19], given as:*

$$LI = \frac{T_{max} - T_{avg}}{T_{avg}} \tag{4.4}$$

Our experimental results in Figure 4.2 show that the LBE provides orders of magnitude improvement in the load balance (significantly lower LI) across increasing database sizes compared to the naive (simple chunk or random) database partitioning.

4.3.2 Load Imbalance in HiCOPS

We measured the effectiveness of the LBE algorithm in the HiCOPS algorithm using all 12 sets of performance evaluation experiments (exp_i) described in Section 3.5.4. Our experimental results in Figure 4.3 show that the load imbalance (LI) related overheads in HiCOPS with the LBE algorithm remains below 10% for nearly all experiments for databases up to 4TB for up to 72 parallel nodes (144 database partitions).

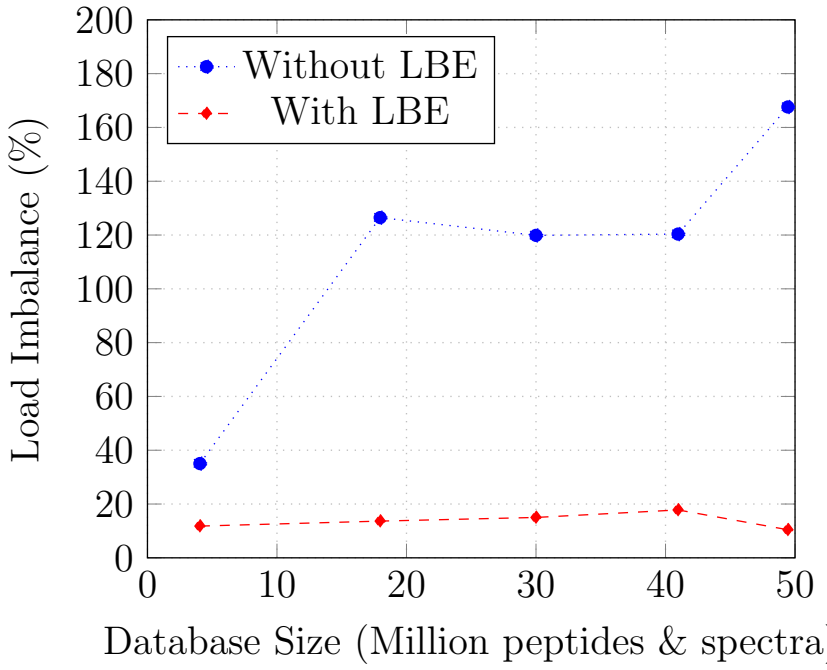

Figure 4.2: The LBE based partitioning exhibits $\sim 5\times$ improvements over a conventional data partitioning for 16 partitions [HAS19].

4.4 Summary

Load-balance is a critical component of the achieved parallelization efficiency in HiCOPS as it is a direct measurement of the CPU utilization efficiency. As reported by several other HPC database peptide search tools and corroborated by our own experimentation, the load imbalance in distributed-memory database peptide search algorithms can reach up to 120% - corresponding to $\sim 6\times$ performance degradation - with existing and random experimental and/or theoretical MS/MS spectra data distribution techniques. We present the LBE algorithm, which models and exploits the sequence similarity and edit positional information between the database peptide sequences and their PTM-variants to first cluster them and then finely distribute them across the database partition. This way, the data *sketch* across the constructed sub-databases is identical yielding similar compute workloads

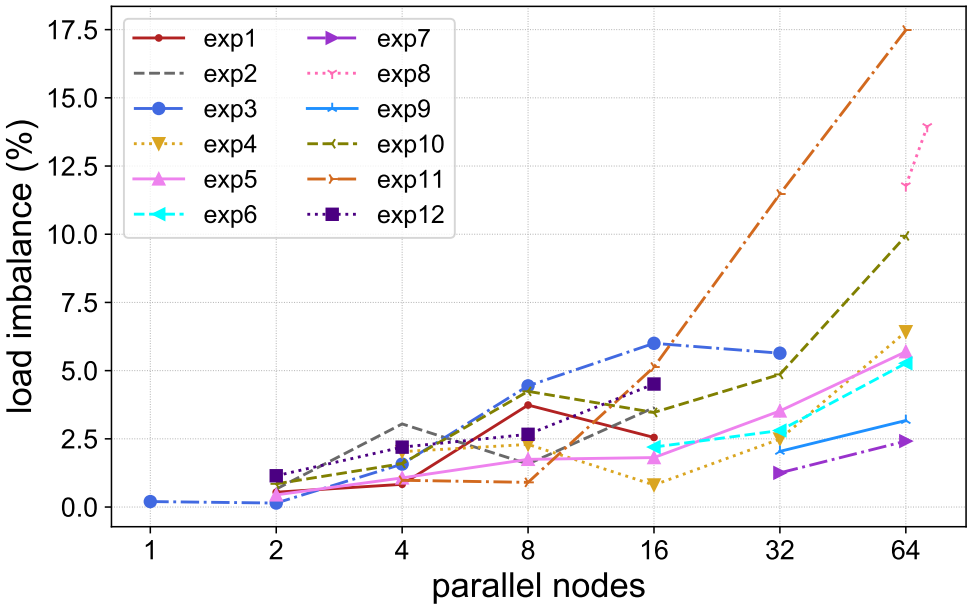

Figure 4.3: Employing the LBE algorithm in HiCOPS results in keeping the load imbalance overheads $< 10\%$ across all 12 experiments for up to 72 parallel nodes (144 partitions) and databases up to 4TB [HS21].

when a random peptide is searched against either of them. The results show that by using the LBE algorithm, the load imbalance LI across the HiCOPS nodes is maintained under 10% even for terabyte-scale databases spanning across more than 100 sub-database partitions yielding a decent strong-scale efficiency.

CHAPTER 5

CFIR: COMPACT AND LOSSLESS FRAGMENT-ION INDEXING

In this chapter, we will discuss our compact and lossless data structure, called CFIR-Index, to compress the memory footprint of the fragment-ion index by $2\times$ with without impacting its indexing or search speed complexities. We will also discuss its application in optimizing the memory and time complexity costs of fragment-ion index based database peptide search algorithms, including the HiCOPS.

5.1 Introduction to Fragment-Ion Search

Fragment-Ion searching, also called shared-peak counting, is one of the most commonly employed filtering techniques in the open-search database peptide search application [KLA+17], [THS+05], [BLY+18]. In this technique, the theoretically generated MS/MS fragment-ion data are indexed in a (inverted) way such that it can be queried and accessed in $O(1)$ time. This constructed index is known as a fragment-ion index [KLA+17], [THS+05].

Then, during the database peptide search, the fragment-ion index is queried to quickly filter the otherwise terabytes of database into only the candidate peptides that share $\geq k$ fragment-ions (or peaks) with the experimental MS/MS spectra being searched [CHY+15], [LCW+10], [ZXS+12]. However, constructing a fragment-ion data index incurs a substantially large memory footprint often beyond the available main memory requiring partitioning into smaller independent chunks and swapping (page faults) from the disk to be processed one at a time [KLA+17]. Further, querying the fragment-ion index involves enormous numbers of random memory reads and writes, which leads to severe memory contentions and bottlenecks. Let us discuss application of the fragment-ion search (or shared-peak counting) in database peptide search in more detail.

5.1.1 Fragment-Ion Searching in Database Peptide Search

The fragment-ion search essentially computes the dot product (or cosine distance) between a pair of MS/MS spectra, using a relaxed fragment-ion tolerance range (δF) [CB04]. This same tolerance is also often employed to discretize the otherwise floating-point data points in the MS/MS spectra using a bin width $b \geq 1/\delta F$ [EJH13], [HS19]. As a result, each spectrum can be represented as a sparse binary vector of length $= M/r$ where M is the maximum (discretized) fragment-ion mass. Using this representation, a δF-relaxed matrix-matrix multiplication operation between the database matrix D and the experimental MS/MS spectra matrix Q will give the shared-peak counts between each $D \times Q$ spectral pair in the scorecard matrix B as follows [HS19]:

$$D_{|D| \times M/r} \times Q_{M/r \times |Q|} = B_{|D| \times |Q|} \tag{5.1}$$

$$\begin{bmatrix} 0 & 1 & .. & 0 & .. & 1 \\ 1 & 0 & .. & 1 & .. & 0 \\ 0 & 0 & .. & 0 & .. & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & .. & 0 & .. & 1 \\ 1 & 0 & .. & 1 & .. & 0 \\ 0 & 0 & .. & 0 & .. & 1 \end{bmatrix} = \begin{bmatrix} 0 & 3 & .. & 0 & .. & 5 \\ 7 & 0 & .. & 2 & .. & 4 \\ 0 & 4 & .. & 9 & .. & 12 \end{bmatrix}$$

Since all matrices D, Q and S are sparse, most existing fragment-ion data structures employ Compressed Sparse Rows (CSR) or Compressed Sparse Columns (CSC) [BG08] to store and query the non-zero (NNZ) data. For each indexed fragment-ion, the following information is also stored to query its properties: fragment-ion mass (m/z) m_i, parent vector id ($osid$), ion-series s_i, ion-charge z_i. To speed up the search, the existing fragment-ion data structures [CHY+15], [KLA+17] optimize the two query operations: *rank* and *select* [HS19] described as follows.

1. $rank(T, m)$: Retrieve the frequency of the fragment-ion with mass $= m$ in T.

2. *select*(T, m, k): Retrieve the properties of the k^{th} instance of fragment ion with mass = m in T. Retrieve the properties of all ions with mass = m, if k is not specified.

5.2 The CFIR-Index Data Structure

We built the **C**ompact **F**ragment-**I**on Index **R**epresentation (CFIR-Index) [HS19] using two observations. First, the number of unique ion masses (or m/z's) in a typical fragment-ion index is extremely small compared to the total number of ions as shown in Figure 5.1 meaning that the m/z (or simply the mass) information which typically requires 4-bytes can be represented in a compact format. Second, the theoretical experimental MS/MS spectra matrix D can be split into distinct sub-matrices, all having the same lengths proportional to their parent peptide sequence lengths. This, combined with a restructuring of the fragment-ion data vectors allows encoding their ion-properties within their respective position within only 4 byte integers. The details of the split-matrix fragment-ion data representation along with algorithms for index construction and querying are discussed in the following sections.

5.2.1 Data Representation in CFIR-Index

The theoretical MS/MS spectra in the CFIR-Index are simulated as vectors of integer fragment-ion masses represented as $S_j = [i_1, .., i_k, .., i_n]$ where $0 \leq i_k \leq M - 1$. The generated ions in each vector i_k are ordered first by their ion-series, then by their fragment charge, and then by the peptide sub-sequence they correspond to (also equivalent to their m/z). Then, the vectors of the same length (corresponding to the peptide sequences of the same length) are stacked together to form an

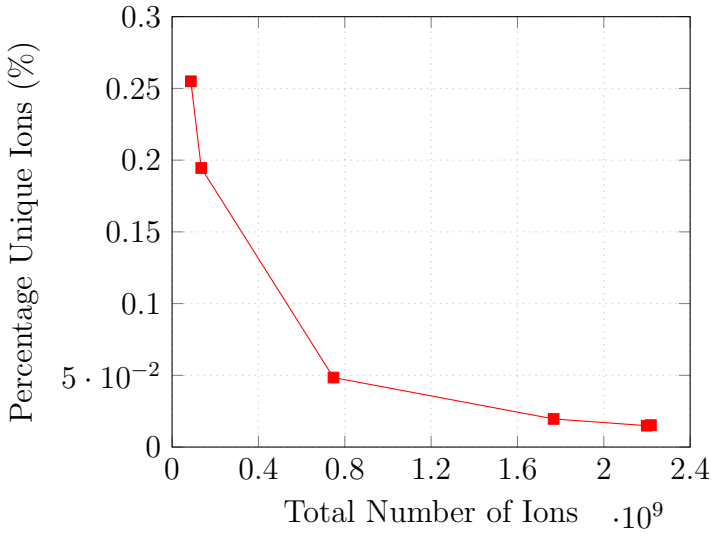

Figure 5.1: The plot depicts that the ratio of unique ion masses to total number of ions in a fragment-ion is extremely small for increasing size of index [HS19].

instance N_i of the CFIR-Index given as: $N_i = [S_1, S_2, S_3, \cdots, S_Y]$. The ordering of the spectra vectors in CFIR-Index and the construction of the CFIR-Index instance sub-matrix N_i with Y spectra of length n is shown in Figure 5.2, and can be written as:

$$N_i = \begin{bmatrix} i_{11} & .. & i_{1k} & .. & i_{1n} \\ i_{j1} & .. & i_{jk} & .. & i_{jn} \\ i_{X1} & .. & i_{Xk} & .. & i_{Yn} \end{bmatrix}$$

where i_{jk} is the mass of k^{th} fragment-ion in the j^{th} spectrum. Since all vectors in N_i have the same length $= n$, we can flatten N_i into a vector $T = [i_0, i_1, i_2, ..., i_{Yn-1}]$ where the original positions can be inferred as $dim(N) = (Y \times n)$.

5.2.2 The CFIR-Indexing Algorithm

Let T be the flattened vector version of an instance (sub-matrix N_i) of the CFIR-Index with Y vectors each with length n. Then, we define another vector A containing indices from 0 to $len(T) = Yn - 1$ and apply a *Stable Key-Value Sort* with T as

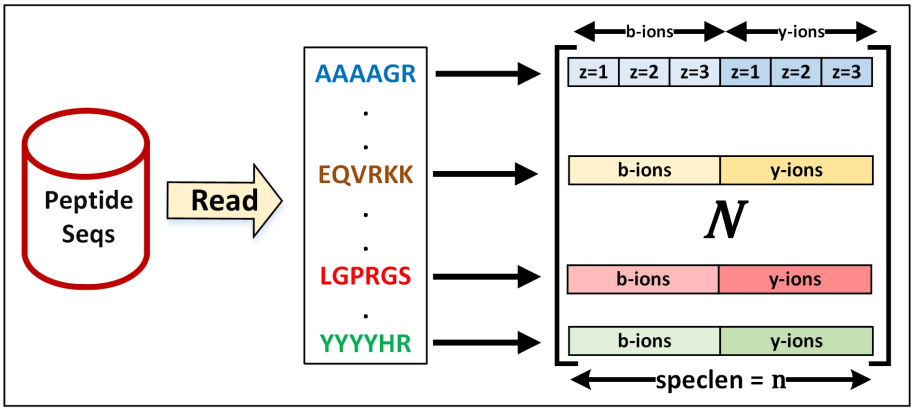

Figure 5.2: The ions are first ordered by the ion-series, then by fragment charge and finally by their m/z. The spectra of the same length are stacked in a sub-matrix of CFIR-Index N_i [HS19].

keys and A as values yielding the rearranged versions of the two vectors namely Ω and Λ respectively as:

$$\Omega, \Lambda = StableKeyValueSort(T, A)$$

Here, Ω is simply the sorted T, and from Figure 5.1 we know the ratio of unique values in Ω will be $<< |\Omega|$ and also since all data are sorted, we can encode it into a much smaller representation by storing a running counter of the frequency of each unique ions as:

$$\Omega'[i] = \{freq[i] \ ; \ i \ \epsilon[0, M)\}$$

$$\Omega''[i] = prefixSum(\Omega'[i])$$

Notice that the $|\Omega''| = M << Xn = |T|$ where M is the maximum ion mass in T resulting in a memory footprint of $Xn + M \approx Xn = |T|$ ($\sim$ 4.01bytes/ion) [HS19]. The same algorithm is applied to all sub-matrices of the CFIR-Index to construct the complete index. The CFIR-Index transformation is shown in Figure 5.3 and an example is illustrated in Figure 5.4 while the final CFIR-Index I for p sub-matrices can be written as:

$$I = [\{\Lambda_1, \Omega''_1\}, \{\Lambda_2, \Omega''_2\}, \cdots, \{\Lambda_p, \Omega''_p\}] \tag{5.2}$$

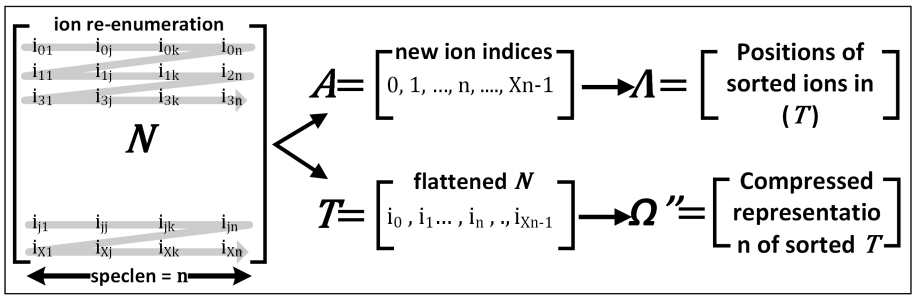

Figure 5.3: The flattened fragment-ion data T along with the vector of ion indices A are Stable Key Value sorted using T as key and A as value. The frequencies of each ion are counted into Ω' and a prefixSum is applied to yield Ω' [HS19].

5.2.3 The CFIR-Index Querying Algorithm

A fragment-ion f in an experimental MS/MS spectrum $q = [f_1, f_2, \cdots]$ can be queried against the CFIR-Index by first locating its occurrences in an instance of the CFIR-Index using the Ω'' array at $[\Omega''[f], \Omega''[f+1])$ [HS19]. Then, the ion-positions in Λ in this range can be decoded (using Algorithm 3) to compute the ion properties including the parent spectrum id *osid*, ion series s_i and ion-charge z_i using the ion ordering in Figure 5.2. Formally, the *rank* and *select* operations can be computed as follows and an example of *rank* and *select* is illustrated in Figure 5.5.

1. $rank(T, f)$: is given as $\Omega''[f+1] - \Omega''[f]$, computed in $O(1)$ time.
2. $select(T, f, k)$: is given at position: $\Lambda[\Omega''[f] + k]$, whereas all occurrences can be located at the range: $[\Lambda[\Omega''[f]]$, $\Lambda[\Omega''[f] + rank(T, f) - 1]]$; also computed in in $O(1)$ time.

The decoded shared-peaks computed in Algorithm 3 can be used to update a scorecard with counts of shared ions as well as corresponding running sums of

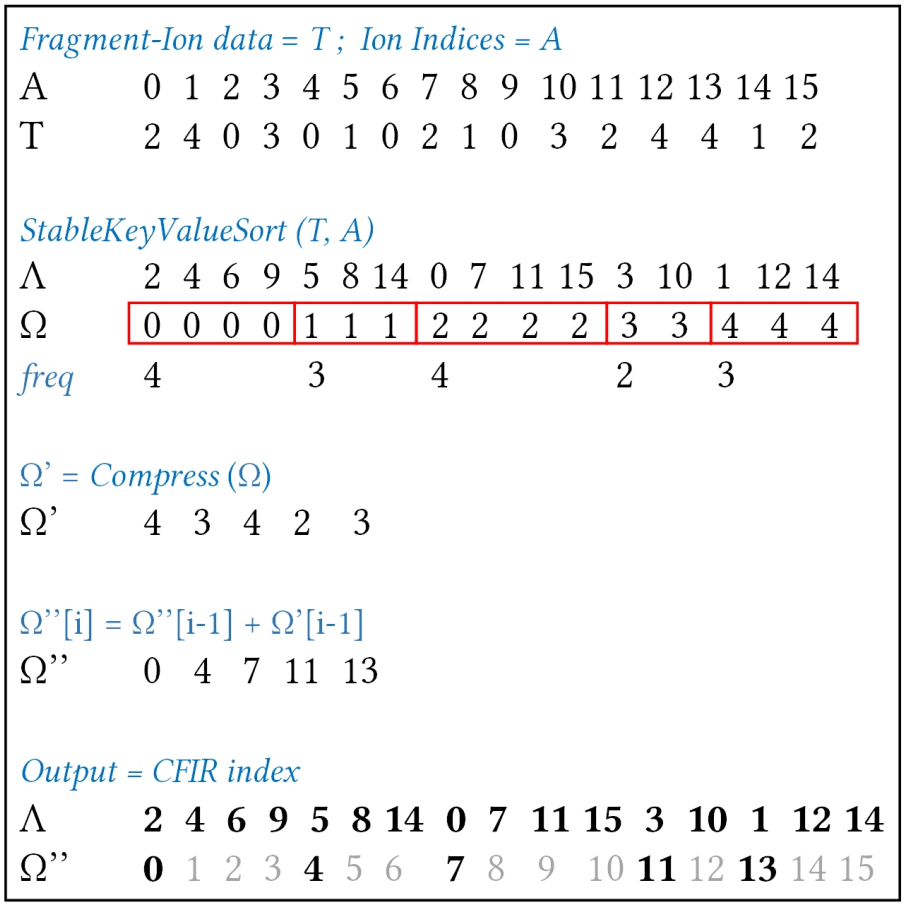

Figure 5.4: An example of CFIR-Index construction for fragment-ion data in the range $[0, 4]$. Notice that the output list Ω'' contains only the bolded black entries while the greyed ones are omitted [HS19].

intensities to later assemble the hyperscore in one-pass. Given a pair of spectra, theoretical or experimental, ν and ξ, the number of shared b- and y-ions between them be n_b and n_y respectively with corresponding intensities $i_{b,j}$ and $i_{y,j}$, then the hyperscore between them is as follows:

$$hyperscore(\nu, \xi) = \log(n_b!) + \log(n_y!) + \log(\sum_{j=1}^{n_b} i_{b,j}) + \log(\sum_{k=1}^{n_y} i_{y,k}) \tag{5.3}$$

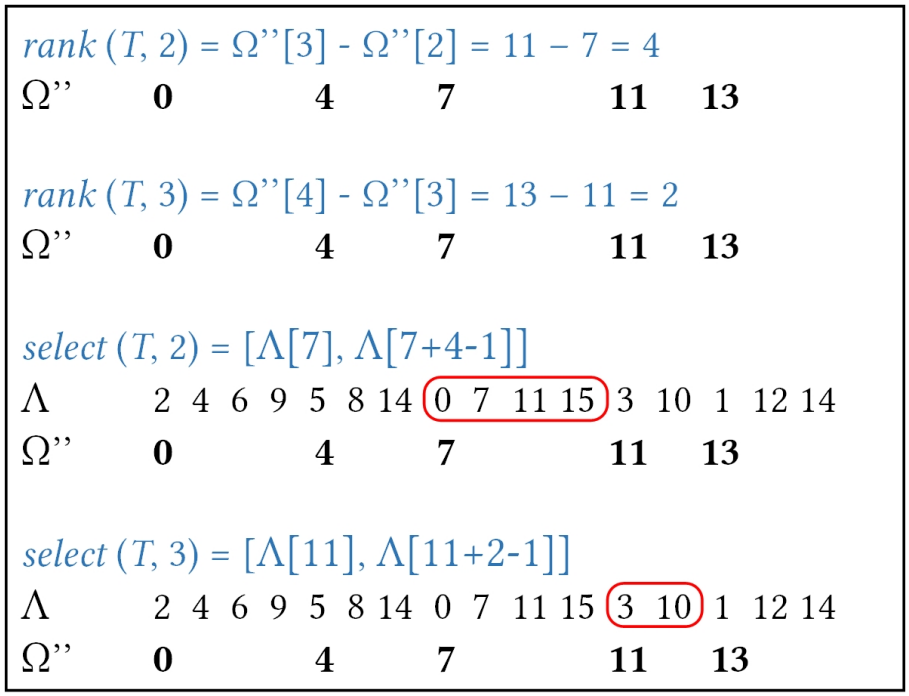

Figure 5.5: Examples of computing *rank* and *select* in the CFIR-index for fragment-ions with mass 2 and 3 [HS19].

5.3 Results

All our experiments were run on a Windows machine with 20GB RAM for compatibility purposes.

5.3.1 Memory Footprint

We measured the memory footprint of the CFIR-Index against the fragment-ion index data structures implemented in SpecOMS, MSFragger, and ANN-SoLo for increasing database sizes. These increasing index sizes were constructing by increasing the number and type of PTMs in the fragment-ion index construction. The index size for ANN-SoLo was increased by concatenating spectral libraries from multiple sources including Human Orbitrap spectral library from ISB and Mouse spectral

Algorithm 3: Search Algorithm [HS19]

Data: fragment-Ion positions in T (p), spectrum length (len), max ion-charge in index (z_{max})
Result: Fragment-Ion properties

```
/* Origin spectrum ID                                          */
osid = ⌊p/len⌋;
/* ion-number in the series                                    */
inum ← (p mod (len/2)) + 1;
/* ion-charge in parent spectrum vector                        */
iz ← ((inum − 1)/(len/2z_max) + 1);
/* ion-series parent spectrum vector                           */
is ← y;
if (p mod len) ≤ (len/2) then
    is ← b;
return osid, inum, iz, is;
```

library from NIST. The results in Figure 5.6 depict that the CFIR-Index provides at least a 2× improvement over all other data structures. We also extended the memory footprint results showing a consistent 2× improvement for CFIR-Index and MSFragger for larger databases on a Linux based server machine with 128GB RAM as depicted in Figure 5.7.

5.3.2 Indexing Speed

We also measured the indexing speed for the three tools for varying index sizes. The experimental results in Figure 5.8 show that MSFragger and CFIR-Index outperform ANN-SoLo and SpecOMS by several folds. It can also be seen that the MSFragger is slightly faster than CFIR-Index, but their overall trend (asymptotic time complexity) is similar.

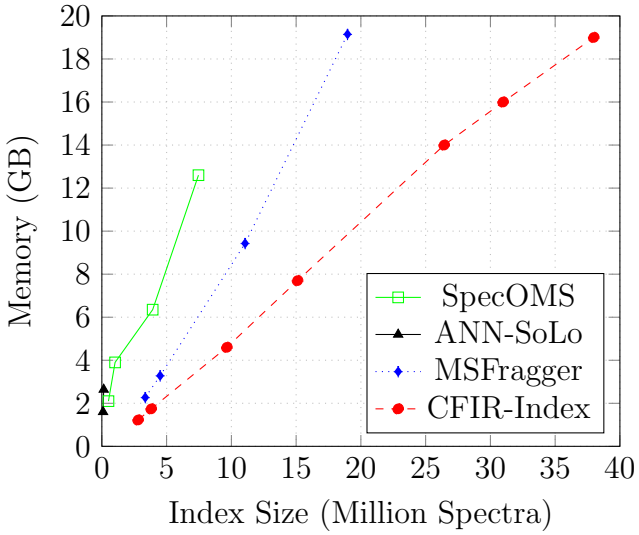

Figure 5.6: CFIR-Index is capable of indexing ∼40 million spectra (in 20GB RAM) which is 2× than any other data structure [HS19].

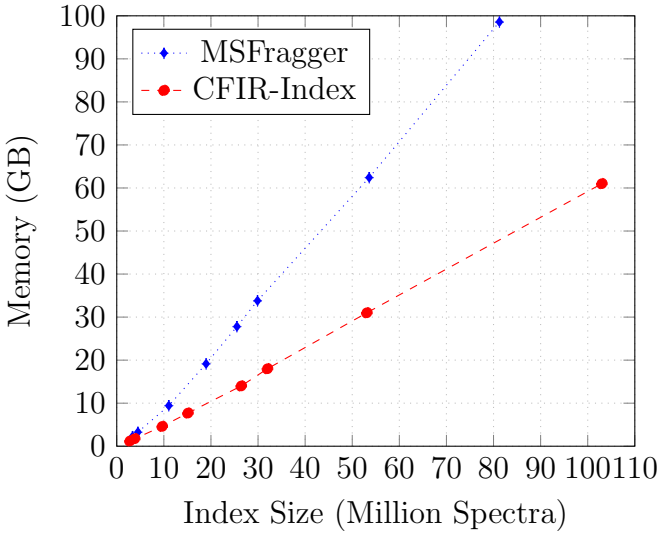

Figure 5.7: Extended memory footprint results for CFIR-Index and MSFragger for up to 128GB index sizes [HS19].

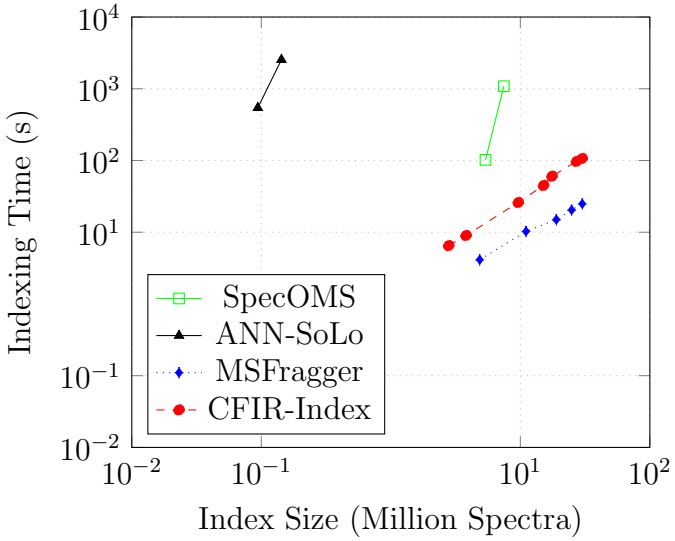

Figure 5.8: Indexing time results show that MSFragger and CFIR-Index exhibit similar time complexity and outperforms others by several folds [HS19].

5.3.3 CFIR-Index Querying Time

We measured the CFIR-Index search (query) time for by searching subsets of the dataset E_1 against increasing size CFIR-Index sizes constructed by incrementally adding M-oxidation, CK-gly-gly adducts, and NQ-deamidation. Our results in Figure 5.9 depict an almost linear relationship between the search speed and the experimental MS/MS data size indicating the constant (slope) search time complexity.

5.3.4 Application in the Database Peptide Search

We measured the performance scalability of the querying time for the CFIR-Index and the MSFragger to illustrate the application of the CFIR-Index in large-scale and especially distributed-memory database peptide search algorithms such as the HiCOPS. To do this, we (multicore) searched a subset of the dataset: E_1 against increasing size databases on a computer equipped 32GB RAM. The results in Figure 5.10 depicts that the MSFragger depicts a performance loss of about 12% beyond

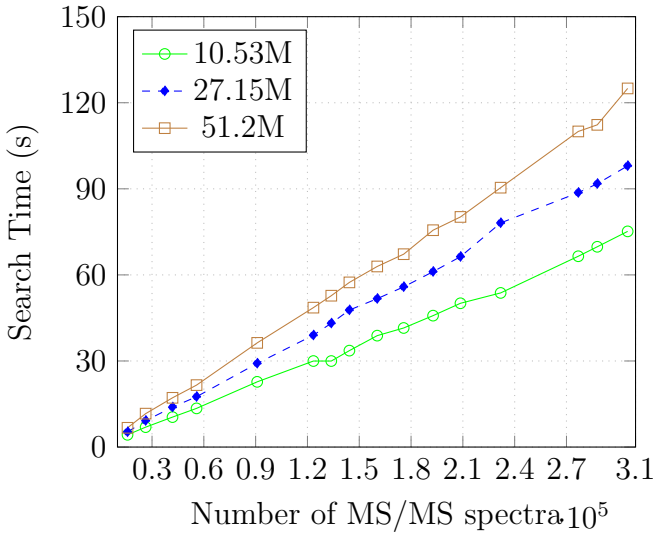

Figure 5.9: The CFIR-Index search time depicts linear relation with experimental data size for a given index size (individual color lines) indicating constant time (slope) complexity [HS19].

the 16 million spectra (requiring out of core processing) whereas the CFIR-Index, which consumes about half the memory as MSFragger, performs linearly until 32 million spectra.

5.4 Summary

The search speed of a distributed- or shared-memory open-search database peptide search algorithm such as the HiCOPS, heavily relies on its memory performance due to the low AI of the algorithms and the data volumes involved. Further, the minimum number of nodes (or memory resources) required by a distributed-memory algorithm is also determined by the memory-footprint of the database. The best existing data structures for the fragment-ion index in database peptide search algorithms require at least 8 bytes per indexed ion resulting in massive index sizes. We present the CFIR-Index data structure, which exploits the low entropy and sparsity of the fragment-ion data to encode the positions thereby, improving the memory

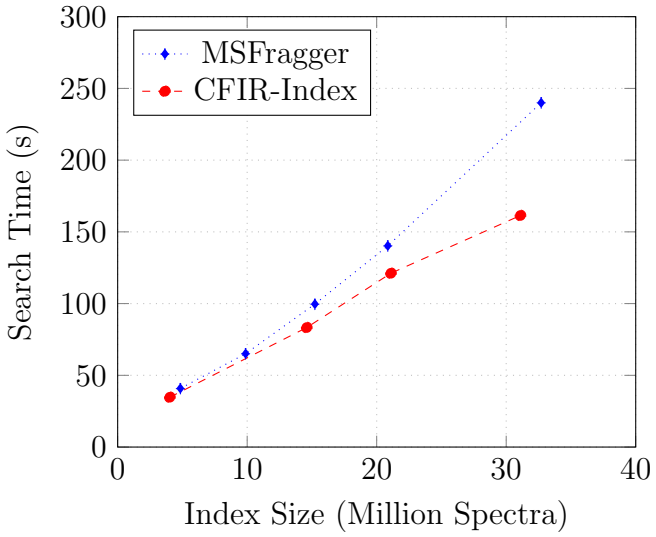

Figure 5.10: MSFragger shows a performance drop in search time speed beyond 16 million index size mark due to out-of-core computing while the CFIR-Index performs linearly up to 32 million size on a 32GB RAM computer [HS19].

density to about 4.01 bytes per indexed ion or 2× improvement compared to all existing data structures.

Further, the asymptotic runtime bounds for the database indexing and querying (search) speeds remain the same with only $O(N)$ and $O(1)$ extra computations required for each respective operation. This 2× memory-footprint by the CFIR-Index is also significant in reducing and optimizing the number of required parallel nodes (resource utilization optimization) in including HiCOPS by half. Therefore, running a database peptide search experiment with a database of 1 billion indexed peptides with CFIR-Index will require about 1TB RAM (or 10 nodes ×100GB) compared to 2TB RAM (or 20 nodes ×100GB) with the best existing data structure.

CHAPTER 6

GICOPS: THE GPU-ACCELERATED HICOPS

In this chapter, we will discuss the algorithms, data pipelines, and optimizations for GPU-accelerating our HiCOPS framework (introduced in Chapter 3) on heterogeneous (CPU-GPU) supercomputing architectures. The new GPU-accelerated HiCOPS will be called as the GiCOPS framework. We will also discuss the speed improvements and the GPU throughput exhibited by GiCOPS for database peptide search.

6.1 Computational Steps in GiCOPS

GiCOPS aims to GPU-accelerate the algorithms in the four supersteps - discussed in Section 3.1 - of HiCOPS . The supersteps are simply referred to as *steps* in this chapter as the developed GPU-accelerated are at the intra-node level. Note that the developed CPU-GPU methods automatically scale system-wide by the SPMD parallel design of the underlying HiCOPS. Therefore, GiCOPS strives to GPU-accelerate and optimize the following four steps:

1. Database Indexing: Simulate the theoretical MS/MS spectra from the peptide sequences and construct the CFIR-Index [HS19].
2. Preprocessing: Preprocess the experimental MS/MS spectra data.
3. Database Search: Compute the fragment-ion search coupled hyperscore based database peptide search algorithm
4. Postprocessing: Postprocess the search results and compute the confidence scores.

6.2 The GiCOPS Methods

6.2.1 Notations and Symbols

For consistency, we will use the same symbols and notations defined in the Section 3.2.1. Apart from the predefined symbols, we will denote the GPU thread block size as (ψ), the CPU-GPU communication latency as (ω) and the bandwidth as (π). Similar to the chapter 3, we will refer the indexed theoretical MS/MS spectra database as simply *the database* in this chapter as well.

6.2.2 Runtime Cost Model

We will import the same runtime cost model employed in HiCOPS in Section 3.2.2 for consistency. However, we will drop the *per-node* (i.e., p_i) subscripts as the GiCOPS methods will be designed and analyzed for single-node case, and extended system-wide via the pre-established HiCOPS methods (discussed in Chapter 3). Furthermore, given the CPU-GPU latency ω and bandwidth π, the communication cost of a one-way transmission of B-bytes data will be given as: $O(\omega + B/\pi)$.

6.2.3 CPU-GPU Pipeline

The CPU-GPU pipeline in GiCOPS consists of a global work queue, a priority queue and a scheduling thread. The work units (database, experimental data etc.) to be processed are queued in the global queue either at once in the beginning or through the data producer sub-tasks. The scheduling thread then schedules the work units from the global queue to either CPUs or GPU based on their priority and availability in the priority queue. The priority queue allows fine-tuning the preferred compute unit for each algorithm or step depending on the workload profile and the nature

of the data and computations. For instance, GPUs may be the preferred compute unit for compute-intensive floating point workload and CPUs may be the preferred compute unit for memory- or I/O intensive workloads. A schematic of the GiCOPS's CPU-GPU pipeline is illustrated in Figure 6.1.

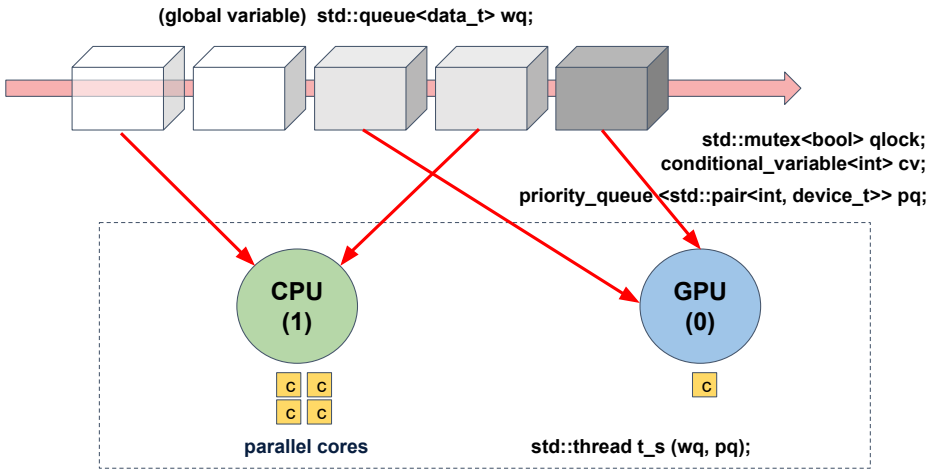

Figure 6.1: The CPU-GPU pipeline in GiCOPS consists of a global work queue, a priority queue, synchronization intrinsics, and a task scheduling thread.

6.2.4 Step 1: Database Indexing

In this step, GiCOPS generates the theoretical MS/MS spectra data and constructs the CFIR-Index. In case of HiCOPS, the CPU-side first executes the LBE algorithm [HAS19] to construct the local peptide sequence sub-databases. In single-node case, the CPU-side simply reads the peptide sequences, separates by length, for each instance of the CFIR-Index [HS19], and communicates them to the GPU. The GPU executes a two-fold parallel data generation kernel where each thread block of size sl where s is the number of ion-series to generate and l is the peptide sequence length, generates the corresponding theoretical MS/MS spectrum for each peptide sequence. Inside each block, each thread operates on a peptide sequence character to compute

the respective ion-series using warp and block-wise reduction trees. On completion, the respective instance of the CFIR-Index is constructed using array-wide *stable sort-by-key*, and *lowerbound* kernels [BH12]. The computed index is communicated back to the CPU-side. The process repeats until all instances of the CFIR-Index are computed. Figure 6.2a illustrates the GPU-algorithm and the data pipeline of this step.

Runtime Cost: The GPU-side runtime cost of this step involves generating the theoretical MS/MS data in $O(D)$, building the CFIR-Index in $O(D \log D)$ plus computing the CFIR-Index's lower-bound array (i.e., Ω'') in time $O(a \log a)$ where a is the (discretized) maximum ion mass in CFIR-Index. Finally, the overhead cost of communicating the CFIR-Index back to the CPU is given by $O(\omega + D/\pi)$ as:

$$T_1 = k_{1,1}(D) + k_{1,2}(D \log D) + k_{1,3}(a \log(a)) + k_{1,4}(\omega + D/\pi) \tag{6.1}$$

6.2.5 Step 2: Experimental Data Preprocessing

In this step, GiCOPS, similar to HiCOPS, preprocesses the experimental MS/MS data and writes it back to the file system for subsequent runs. The CPU side reads the experimental MS/MS data in batches, indexes them and streams to the GPU. The GPU side globally preprocesses each batch using the Sorted Tag Approach (STA) [AS16] for better scalability. The preprocessing involves extracting top-K (K=100 or 150) data points by intensity from each experimental MS/MS spectrum similar to [KLA+17] and [HDDA21]. In the STA, all spectra in a batch are first concatenated in super-vector Q. Another vector, called tag-array T is also initialized where $T[i] = j$ where j is the spectrum number of each data point in the super-vector Q to preserve the spectrum positions in Q. Then, the *stable-sort-by-key* kernel [BH12] is applied to Q and T twice, first using Q as key and T as value and then

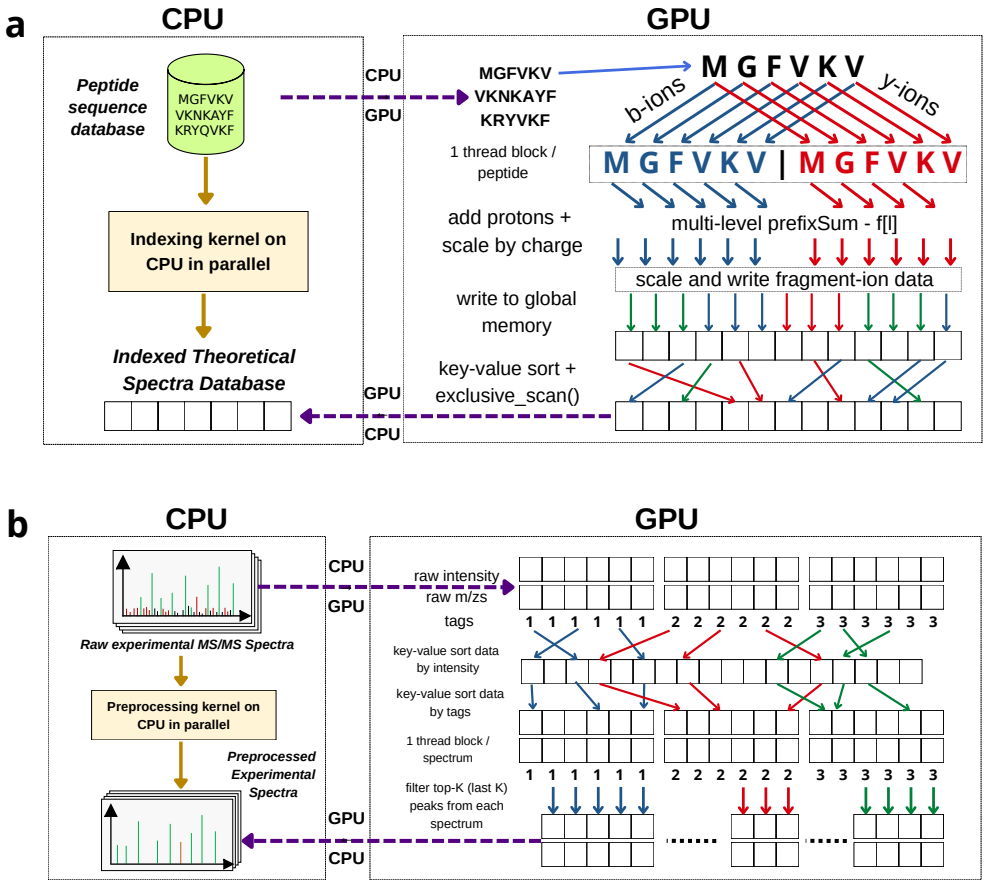

Figure 6.2: **(a)** CPU side communicates the peptide sequences to the GPU where the theoretical MS/MS spectra are simulated, indexed and communicated back to the CPU. **(b)** CPU side streams the experimental MS/MS spectra to the GPU where they are processed using the supplied algorithm and streamed back to the CPU.

using T as key and Q as value. The final result of this is that each spectrum $q \in Q$ is sorted and is extracted back from Q using T using another two-fold parallel GPU kernel. Figure 6.2b illustrates the GPU-algorithm and the data pipeline of this step. Figure 6.3 illustrates an example of the STA approach.

Runtime Cost: We will skip the GPU-side analysis for this step as it does not significantly contribute to the overall GiCOPS's execution. This is because the CPU-GPU pipeline naturally assigns almost all work units in this step to the CPU

side due to its memory-intensive nature. Further, the experimental MS/MS data are pre-processed only once and written to the disk to be used in the subsequent runs.

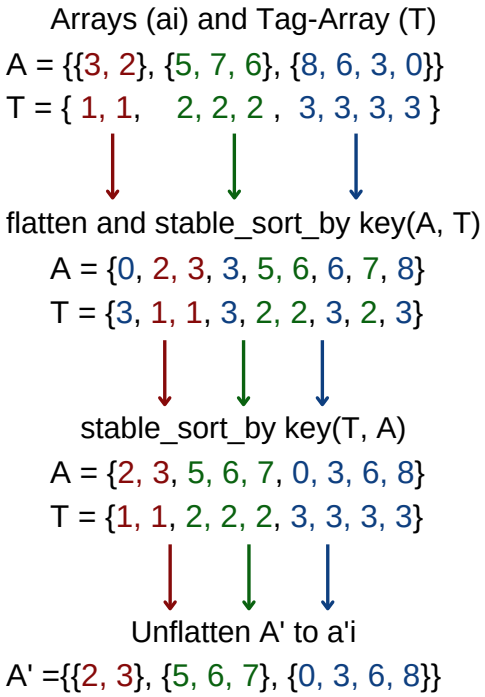

Figure 6.3: Illustration of the STA algorithm using example arrays A and T.

6.2.6 Step 3: Database Peptide Search

In this step, GiCOPS searches the batches of experimental MS/MS spectra data against the indexed (CFIR-Index). The CPU side reads the batches of (preprocessed) experimental MS/MS spectra and streams them to the GPU. The CFIR-Index index is also pre-communicated to the GPU. The GPU side them searches each batch against the CFIR-Index in a two-fold parallel model where each thread block searches an experimental spectrum against the CFIR-Index. The number of threads per block vary depending on the search mode. i.e., more threads per block

for open-search and vice versa. Multi-step CUDA warp-shuffle based ψ-ary reduction loops/trees [HDDA21] are employed to optimize the shared and global memory usage as well as to alleviate race conditions without using atomics, but at an additional $\log(\psi)$ cycles per 1 data-parallel (ψ) memory updates. The generic pattern of the optimized two-step ψ-ary reduction kernels in GiCOPS is illustrated in Algorithm 4. The database peptide search results are communicated to the CPU for the underlying HiCOPS framework to communicate and assemble across the parallel nodes. In case of single node GiCOPS application, these results are not communicated to the CPU at this step and are directly used to execute the fourth (result postprocessing) step and only the final postprocessed results are communicated to the GPU as shown in Figure 6.4. Figure 6.4 illustrates the GPU-algorithm and the data pipeline of this step. This process repeats for all batches of the experimental MS/MS spectra.

Runtime Cost: The GPU-side runtime cost of this step involves the communication costs of the CFIR-Index and b batches of experimental data in runtime: $O(b\omega + (Q + D)/\pi)$, the cost of database filtering by peptide precursor mass in runtime: $O(q\log(D)) + O(q\eta\alpha\log(\psi)) + O(q\mu\log(\psi)) + O(q\upsilon/\psi)$ where α is the average fragment-ion matches per experimental fragment, μ is the average database peptide candidates per experimental spectrum and υ is the average number of collisions per atomic null distribution (histogram) construction. Finally, in multi-node case, the communication of communicating intermediate results for q spectra is $O(b\omega + 2048q/\pi)$. Collectively, the cost of this step can be written as:

$$\begin{aligned} T_3 = k_{3,1}(b\omega + (Q + D)/\pi) + k_{3,2}(q\log(D)) + k_{3,3}(q\eta\alpha\log(\psi)) + \\ k_{3,4}(q\mu\log(\psi)) + k_{3,5}(q\upsilon/\psi) + k_{3,6}(b\omega + 2048q/\pi) \end{aligned} \tag{6.2}$$

Algorithm 4: Warp-Shuffle based Reduction kernels

Data: thread-local variable (v_l), commutative and associative reduction operation ($\otimes$), conditions $cond1$,$cond2$

Result: reduced global variable v_g

```
/* currently reduced values in v_l                                    */
/* compute the thread mask                                            */
1  mask ← __ballot_sync(activemask(), cond1);
/* reduce intra-warp                                                  */
2  for off ∈ warpSize/2, ··· , 1 do
3      tmp ← __shfl_down_sync(mask, v_l, off);
4      if cond2 then
5          v_l ← v_l ⊗ tmp;
6  __shared__ vals[32];
/* The laneId = 0 threads now have the reduced value of their warps
   */
7  if laneId = 0 then
8      vals[warpId] ← v_l;
/* read from vals and reduce the first warp only                      */
9  if warpId = 0 then
10     v_l ← vals[laneId];
       /* update thread mask                                          */
11     mask ← __ballot_sync(activemask(), warpId = 0);
12     for off ∈ warpSize/2, ··· , 1 do
13         tmp ← __shfl_down_sync(mask, v_l, off);
14         if cond2 then
15             v_l ← v_l ⊗ tmp;
/* tid = 0 will have the reduced value, broadcast                     */
16 if tid = 0 then
17     vals[0] ← v_l;
/* read and return the reduced value in v_g                           */
18 return v_g ← vals[0];
```

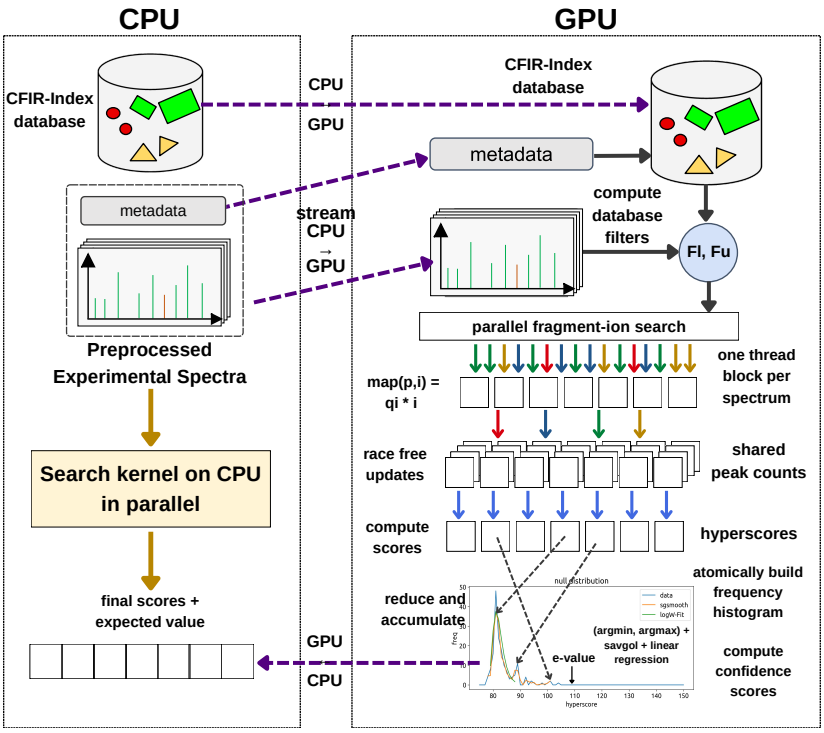

Figure 6.4: CPU side communicates the entire CFIR-Index and streams the preprocessed experimental spectra to the GPU, where they are searched using fragment-ion search method. The computed results are then post-processed before being communicated to the CPU. Multi-tier reduction trees are used to optimize each sub-kernel in these steps. In case of multi-node application, the intermediate results are assembled across the system by HiCOPS before post-processing.

6.2.7 Step 4: Results Postprocessing

In this step, GiCOPS postprocesses the results computed in the last step. The CPU-GPU pipeline of this step depends on the GiCOPS (parallel) application. In case of multi-node application, the partial or intermediate results at each node are first assembled into global results at the CPU side by HiCOPS as explained in Section 3.2.6. In case of single node, the results from the last step are directly used in this step as shown in Figure 6.4 (last steps). The GPU side computes the required confidence scores using optimized data parallel vector and reduction kernels implementing regressions and curve fitting algorithms. The final results from this

step are communicated to the CPU side and written to the file system (single-node application) or communicated to other nodes (multi-node application).

Runtime Cost: We will skip the GPU-side analysis for this step as it does not significantly contribute to the overall GiCOPS's execution. This is because the GPU kernels employed in this step are highly optimized for the SIMT architectures and also contribute less than 1% execution time in most real world application.

6.3 Performance Analysis

GiCOPS's GPU performance can be modeled by analyzing the performance of its steps.

$$T_G = T_1 + T_3$$

Inserting Equations 6.1 and 6.2 in above equation, we have:

$$\begin{aligned} T_G = k_{1,1}(D) + k_{1,2}(D\log D) + k_{1,3}(a\log(a)) + k_{1,4}(\omega + D/\pi) + \\ k_{3,1}(b\omega + (Q+D)/\pi) + k_{3,2}(q\log(D)) + k_{3,3}(q\eta\alpha\log(\psi)) + \\ k_{3,4}(q\mu\log(\psi)) + k_{3,6}(b\omega + 2048q/\pi) \end{aligned} \tag{6.3}$$

Splitting the Equation 6.3 into computational (T_P) and overhead (T_O) parts, we have.

$$\begin{aligned} T_P = k_{1,1}(D) + k_{1,2}(D\log D) + k_{1,3}(a\log(a)) + k_{3,2}(q\log(D)) + \\ k_{3,3}(q\eta\alpha\log(\psi)) + k_{3,4}(q\mu\log(\psi)) \end{aligned} \tag{6.4}$$

and

$$\begin{aligned} T_O = k_{1,4}(\omega + D/\pi) + k_{3,1}(b\omega + (Q+D)/\pi) + \\ k_{3,5}(q\upsilon/\psi) + k_{3,6}(b\omega + 2048q/\pi) \end{aligned} \tag{6.5}$$

From equations 6.4 and 6.5, it can be seen that increasing the database (D) or the experimental dataset size $Q = q\eta$ size directly positively affects the T_P.

However, the dominant quadratic factor that controls the amount of many-to-many search operations are controlled by the peptide (δM) and fragment-ion (δF) mass tolerances, seen as α and μ in Equation 6.4. Note that for small α and μ, the T_P diminishes to only the T_1 factors even with large D and Q, and the T_O may dominate the overall performance. Also notice that the database indexing T_1 is relatively unaffected by other factors and the GPU provides a decent speedup over CPU-only code even for small D.

6.4 Optimizations

In the following sub-sections, we will discuss the optimizations employed in GiCOPS to maximize the GPU throughput given by Equation 6.4.

6.4.1 Race Conditions in Fragment-Ion Search

The fragment-ion matches encountered in the data-parallel database search by GPU threads are written to a scorecard to be processed to compute the hyperscores later on. However, doing so may result in race conditions if multiple threads simultaneously try to update (fetch, update, write) the same scorecard entry. A simple solution could be to employ atomic intrinsics, which would severely hurt the performance in case of (unpredictable) considerable number of collisions. Therefore, to avoid this, GiCOPS implements a race condition and atomics free algorithm involving two main steps. The first step involves ensuring that the sorting algorithms in CFIR-Index are *stable* allowing natural clustering of the (colliding) GPU threads which are to write to the same scorecard location. Then, a warp-shuffle based two-step reduction is employed reducing these clusters in $O(\log(\psi))$ cost before writing to the scorecard.

6.4.2 Performance Tuning

We used the Nvidia Nsight Compute (NCU) software to profile the database peptide search kernel and fine tune several hyperparameters including the GPU thread grid size, shared memory usage, reduce bank faults, optimize register usage and improve the thread occupancy resulting in an additional 25% improvement in performance (incorporated in the reported results), speed of light performance (12.1% compute and 80.06% memory), occupancy factor (80% theoretical max), active blocks (79.6% theoretical max), and the shared memory bank conflicts ($< 0.1\%$ transactions).

6.4.3 Compile-Time Computations

Billions of hyperscore computations are performed in database peptide search experiments. The hyperscore between a pair of spectra ν and ξ with the number of shared b- and y-ions n_b and n_y respectively with corresponding intensities $i_{b,j}$ and $i_{y,j}$ from Equation 5.3 is rewritten as:

$$hyperscore(\nu, \xi) = \log(n_b!) + \log(n_y!) + \log(\sum_{j=1}^{n_b} i_{b,j}) + \log(\sum_{k=1}^{n_y} i_{y,k})$$

Notice that the first two terms in the above equation compute log of $factorial$ of n_b and n_y, which can be precomputed at compile time and reused to avoid billions of $O(n!)$ time computations. To do this, we compute a C++ `constexpr` data structure using dynamic programming memoizing: $\log(n!) = \log(n) + \log((n-1)!)$. This array is communicated to the GPU constant memory to be used at the GPU side as well.

6.5 Results

We employed the database D_1 and the five custom datasets E_1 to E_5 from HiCOPS experimentation to measure evaluate GiCOPS as well. We also used the same

experimental and data processing settings described in the Section 3.5.1.

Runtime Environment: All experiments were run on the *Dragon* scientific computing cluster at the Florida International University (FIU). The cluster consists of 6 compute nodes, each powered by 2 × Intel Xeon Gold 5215 (10 cores each), 2 × NVIDIA RTX A6000 GPU (84 SMs, 48GB DRAM), 2 NUMA nodes × 128GB DRAM, 3TB SSD local scratch space. The compute nodes are interconnected with each other and the storage nodes (18TB) via a 10Gbps Ethernet interconnect.

6.5.1 Correctness Analysis

We measured the correctness of GiCOPS by directly comparing its computed results (identified peptides and the scores) against HiCOPS for single- and multi-node runs. For this, we searched all five datasets against various increasing size databases in both open and closed search modes. The computed scores were directly one-to-one compared against the ones computed by HiCOPS. Figures 6.5a and 6.5b show a comparison of scores (100 samples out of 208K) computed by HiCOPS and GiCOPS from searching the dataset: E_1 against D_1 modified with methionine oxidation as PTM in both search modes (overlayed and duplicates removed).

Figures 6.5a and 6.5b depict that both GiCOPS and HiCOPS computes consistent and correct results irrespective of the degree of parallelism. We also observed a small number ($\geq$0.05%) of discrepancies in the GiCOPS's computed scores across runs. We investigated this and found that this was due to the order of the reduction loops and floating point precision errors. We fixed the first problem by modifying the reduction loop to prefer the peptides with lower index in reduction and slightly widening the δM boundaries in the GiCOPS experiments.

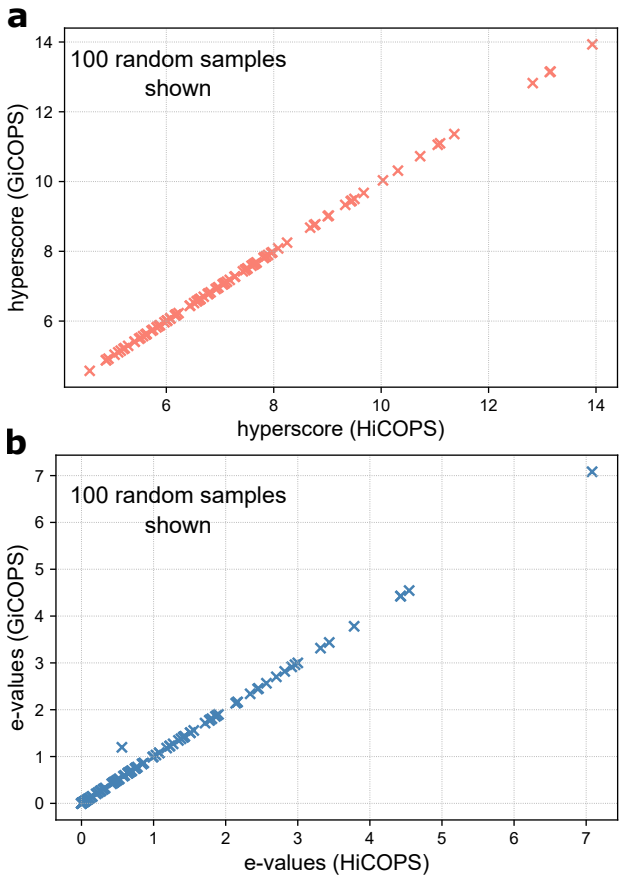

Figure 6.5: GiCOPS computes correct and consistent results for >99.5% samples as compared to HiCOPS across several experiments.

6.5.2 Speed Comparison Against HiCOPS

We measured the GiCOPS speed improvement against HiCOPS (its CPU-only version), by searching all five datasets against increasing size databases in both search modes. Our experimental results in Figures 6.6a to 6.6c show that the GiCOPS's speeds (labeled as g_o and g_c for open and closed search respectively) over HiCOPS (labeled as h_o and h_c for open and closed search respectively) depends on the dataset and database sizes and the search setting. It can be seen that the GiCOPS outperforms HiCOPS by 4-5 $\times$ in open-search and 1-2 $\times$ in closed-search setting. This reduction in speedup stems from the reduced computation-to-computation ratio in

GiCOPS from ∼85% to ∼50% in the database search step as shown in Figure 6.6d. Step by step speed analysis of GiCOPS shown in Figures 6.6e and 6.6f depicts that GiCOPS exhibits speed improvement for step 1 in all experiments while no speedup for step 2. Figures 6.6g and 6.6h show a similar trend in GiCOPS's speedup in multi-node application as the single-node case (in Figures 6.6a to 6.6c) relative to the compute workload per node.

6.5.3 Speed Comparison Against Existing Algorithms

We attempted to measure the GiCOPS speed improvement against several existing GPU based database peptide search software. These software include, Tempest [MFG12], Tide-for-PTM-search [KHUP18], GPUScorer [LXCC14], ProteinByGPU [LCXC14], MIC-Tandem [LLLL19], and PaSER [Bru23]. Unfortunately, we could only do so for Tide-for-PTM-search, referred to as GPU-Tide in the rest of the chapter, as the other software were either outdated, unavailable, incompatible, unusable, proprietary or faulty. To do this, we searched all five datasets, expect E_3 as it crashed for GPU-Tide, using GiCOPS, GPU-Tide, HiCOPS and MSFragger in closed and open-search modes. The open search setting was limited to $\delta M =$ 100Da due to the δM limit in GPU-Tide. In the first experiment, the datasets were searched against D_1 with M-oxidation (size: 3.89 million) as PTM. The experimental results depict that all tools outperform GPU-Tide by $> 100\times$ in open search (Figure 6.7a) and $> 50\times$ in closed search mode (Figure 6.7b). In the second experiment, the datasets were searched against D_1 with M-oxidation and NQ-deamidation as PTMs (size: 10.3 million). Similar results are seen for both the open (Figure 6.7c) and closed search (Figure 6.7d) modes.

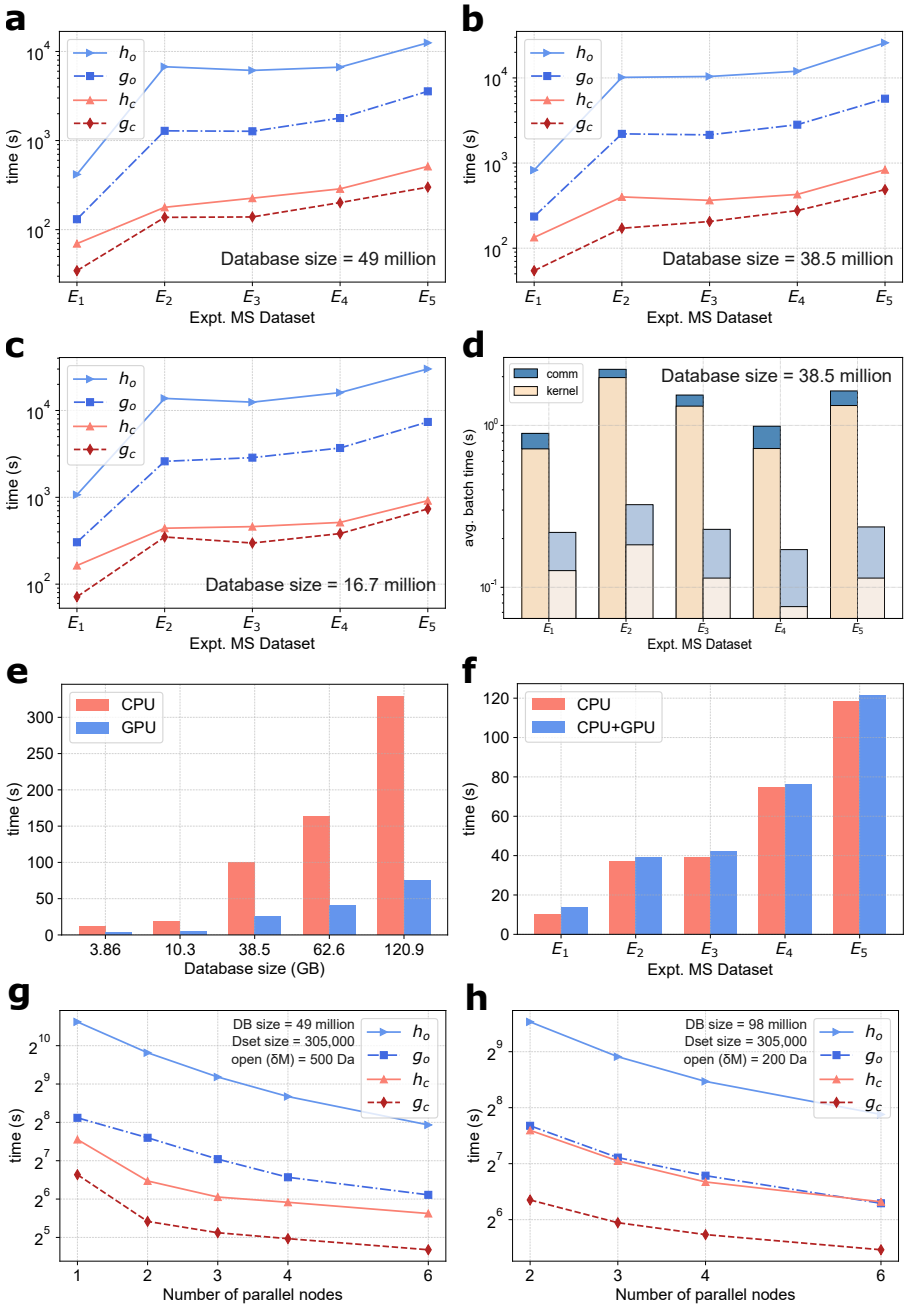

Figure 6.6: **(a to c)** GiCOPS depicts about 4-5× and 1-2× speedup over HiCOPS in open- and closed search modes respectively. **(d)** The reduced speedup in closed-search mode is due to the lower compute to communication ratio. **(e, f)** GiCOPS shows speedup in step 1 for all experiments and almost no speedup in step 2. **(g, h)** Similar speedup results are seen for multi-node GiCOPS vs HiCOPS case with respect to compute load per node as single-node case in **(a to c)**.

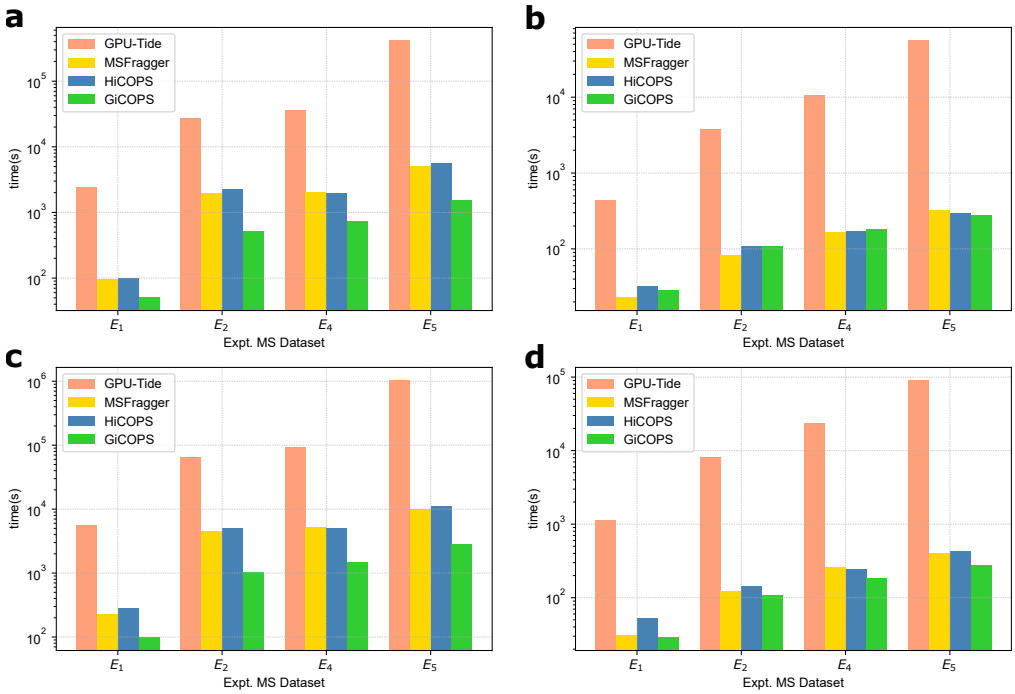

Figure 6.7: GiCOPS and other CPU-only database peptide search tools outperform GPU-Tide by $> 100\times$ in open search **(a, c)** and by $> 50\times$ in closed search **(b, d)** mode.

6.5.4 Performance Evaluation

We measured the GPU throughput of GiCOPS using the Instruction Roofline Model [DW19]. The Instruction Roofline Model, like the traditional Roofline Model [WWP09] is an intuitive model to visualize the performance for non-floating-point kernels. For our case, we profiled and rooflined the database peptide search kernel as it contributes more than 85% of the compute time in sufficiently large experiments. We measured our GPU's (NVIDIA RTX A6000) maximum throughputs using the Empirical Roofline Toolkit (ERT) [LWS+14] and then plotted the database peptide kernel's performance by searching a batch 10000 experimental MS spectra against the Homo sapiens database incorporating M-oxidation, NQ-deamidation, and STY-phosphorylation as PTMs (size: 36 Million) in open- and closed-search modes.

The Roofline results in Figure 6.8a depicts that the GiCOPS achieves a through-

put of 143.608 warp Giga Instructions per second (warp GIPS) in open-search (shown as squares) and 64.642 warp GIPS in closed-search mode (shown as diamonds). The achieved throughputs correspond to 48% and 21.5% of theoretical integer instruction peak on the NVIDIA RTX A6000 GPU. Figure 6.8a also shows that the effect of branch divergence is minimum in GiCOPS as well as the DRAM performance is between stride-1 and stride-8 due to the heavy usage of the scorecard data structure (12 bytes). Figure 6.8b depicts a nearly bank-conflict-free shared-memory performance of GiCOPS's database search for open- (square) and closed-search (diamond) modes.

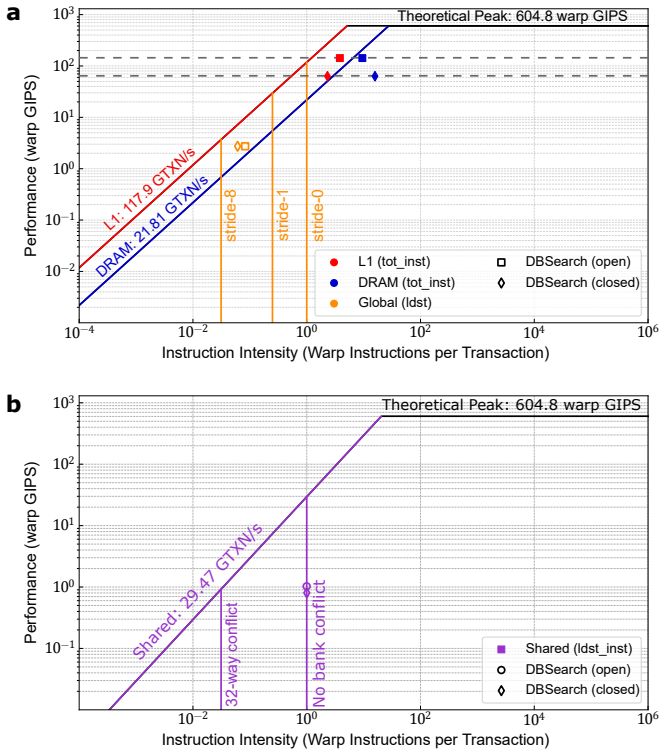

Figure 6.8: **(a)** GiCOPS exhibits a throughput of 143.6 and 64.6 warp GIPS in open- (squares) and closed- (diamonds) search modes corresponding to the 48% and 21.5% of the theoretical integer peak throughput respectively. **(b)** Shared memory performance of GiCOPS depicts almost no bank faults indicating near-ideal usage.

6.6 Summary

Existing GPU-accelerated database peptide search algorithms are based on closed-search (index-free) algorithms and are outperformed by $> 100\times$ newer indexing-based CPU-only algorithms with only a few cores especially for the open-search application. This renders their application in the application domain almost negligible. We present a new-age GPU-accelerated framework or algorithms and optimizations, called GiCOPS, to efficiently accelerate the database peptide search algorithms. Our experiments show that the GiCOPS provides a speed improvement of 4-5$\times$ for open-search application compared to its CPU-only predecessor, HiCOPS, as well as over $100\times$ compared to GPU-Tide, the only existing (open-source) usable GPU-accelerated database peptide search software. Our comprehensive performance evaluation reveals that the GiCOPS provides an average throughput of 143.6 warp GIPS in open-search experiments corresponding to a 95% theoretical maximum integer-instruction throughput on a NVIDIA RTX A6000 GPU. We also discuss the application of the presented algorithms, GPU kernels and optimizations in GiCOPS to build and GPU-accelerate existing and new database peptide search algorithms.

CHAPTER 7

CURRENT AND FUTURE WORK

In this chapter, we will discuss our accomplishments in high performance computing (HPC) for accelerated peptide identification in large-scale systems biology application. We will also discuss the current and future aspects of the research work in this domain.

To date, most research works on database peptide search have focused on improving its accuracy and consistency. While there have been significant algorithmic advancements, the computational efficiency of these state-of-the-art algorithms has barely improved. This is primarily because the designed algorithms are implemented for either serial or multicore computers. While multicore computing scales well for purely numerical algorithms, the complex computations, graph traversals, dynamic programming algorithms employed in modern day database peptide search algorithms render them unscalable (strong-scale efficiency less than 50%) on the shared-memory architectures. This is primarily because the performance gap between the CPU and memory throughputs in the post-Moore computers is significant and is increasing, making data patterns and memory accesses the system bottleneck [SHI22].

Our research work targets this problem by employing distributed-memory and GPU-accelerated supercomputers to alleviate these bottlenecks by implementing novel compute and memory-optimal parallelization methods, optimizations, and data pipelines, which synergistically efficiently accelerate the database peptide search workflows on top-500 supercomputers providing more 10× speedup over the existing infrastructure. The algorithms and optimizations implemented in our HiCOPS and GiCOPS frameworks are designed to be algorithm-oblivious, meaning they can be integrated and extended to accelerate many new and existing algorithms.

Recent advances in machine and deep learning (ML/DL) algorithms for database peptide search have demonstrated modest gains in peptide identification accuracy as well as consistent validation. These ML/DL algorithms include SpeCollate [TS21], yHydra [AMR21], AlphaPeptDeep [ZZW+22], Prosit [GSZ+19]. While these improvements in peptide discovery are significant, these ML/DL algorithms pose even more complex computational profiles and demand astronomical resources to train and deploy their neural networks. On the other hand, recent advances in computing infrastructure including cloud computing, unified memory architectures, Tensor Computing Units (TPUs), Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs) demonstrate impressive accelerations for neural networks. Therefore, there is an urgent need for the development and expansion of the HPC methods discussed in this dissertation to efficiently harness the new architectures to meet the demands of ML/DL algorithms for database peptide search.

The ultimate (future) goal of this dissertation work is the development, expansion and integration of the HPC and ML/DL methods for database peptide search algorithms in a way that HPC harnesses the modern supercomputers to train and ML/DL methods for accurate and consistent peptide identifications as well as the application of machine and deep learning in recognizing the compute, data and control flow patterns aiding HPC methods to better harness the computational resources at hand.

BIBLIOGRAPHY

[ABD+08] Krste Asanovic, Ras Bodik, James Demmel, Tony Keaveny, Kurt Keutzer, John D Kubiatowicz, Edward A Lee, Nelson Morgan, George Necula, David A Patterson, et al. The parallel computing laboratory at uc berkeley: A research agenda based on the berkeley view. *EECS Department, University of California, Berkeley, Tech. Rep*, 2008.

[AMR21] Tom Altenburg, Thilo Muth, and Bernhard Y Renard. yhydra: Deep learning enables an ultra fast open search by jointly embedding ms/ms spectra and peptides of mass spectrometry-based proteomics. *bioRxiv*, pages 2021–12, 2021.

[AS16] 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.

[BCC+07] Robert D Bjornson, Nicholas J Carriero, Christopher Colangelo, Mark Shifman, Kei-Hoi Cheung, Perry L Miller, and Kenneth Williams. X!! tandem, an improved method for running x! tandem in parallel on collections of commodity computers. *The Journal of Proteome Research*, 7(1):293–299, 2007.

[BG08] Aydin Buluc and John R Gilbert. On the representation and multiplication of hypersparse matrices. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–11. IEEE, 2008.

[BH12] Nathan Bell and Jared Hoberock. Thrust: A productivity-oriented library for cuda. In *GPU computing gems Jade edition*, pages 359–371. Elsevier, 2012.

[BLY+18] Doruk Beyter, Miin S Lin, Yanbao Yu, Rembert Pieper, and Vineet Bafna. Proteostorm: An ultrafast metaproteomics database search framework. *Cell systems*, 7(4):463–467, 2018.

[BMNL18] Wout Bittremieux, Pieter Meysman, William Stafford Noble, and Kris Laukens. Fast open modification spectral library searching through approximate nearest neighbor indexing. *Journal of proteome research*, 17(10):3463–3474, 2018.

[Bru23] Bruker. Run & done with paser, 2023.

[CB03] Robertson Craig and Ronald C Beavis. A method for reducing the time required to match protein sequences with tandem mass spectra. *Rapid communications in mass spectrometry*, 17(20):2310–2316, 2003.

[CB04] Robertson Craig and Ronald C Beavis. Tandem: matching proteins with tandem mass spectra. *Bioinformatics*, 20(9):1466–1467, 2004.

[CHY+15] Hao Chi, Kun He, Bing Yang, Zhen Chen, Rui-Xiang Sun, Sheng-Bo Fan, Kun Zhang, Chao Liu, Zuo-Fei Yuan, Quan-Hui Wang, et al. pfind–alioth: A novel unrestricted database search algorithm to improve the interpretation of high-resolution ms/ms data. *Journal of proteomics*, 125:89–97, 2015.

[CKN+15] 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, 2015.

[CLY+18] Hao Chi, Chao Liu, Hao Yang, Wen-Feng Zeng, Long Wu, Wen-Jing Zhou, Xiu-Nan Niu, Yue-He Ding, Yao Zhang, Rui-Min Wang, et al. Open-pfind enables precise, comprehensive and rapid peptide identification in shotgun proteomics. *bioRxiv*, page 285395, 2018.

[CM08] Jürgen Cox and Matthias Mann. Maxquant enables high peptide identification rates, individualized ppb-range mass accuracies and proteome-wide protein quantification. *Nature biotechnology*, 26(12):1367–1372, 2008.

[CNM+11] Jurgen Cox, Nadin Neuhauser, Annette Michalski, Richard A Scheltema, Jesper V Olsen, and Matthias Mann. Andromeda: a peptide search engine integrated into the maxquant environment. *Journal of proteome research*, 10(4):1794–1805, 2011.

[CZS+18] Li Chen, Bai Zhang, Michael Schnaubelt, Punit Shah, Paul Aiyetan, Daniel Chan, Hui Zhang, and Zhen Zhang. Ms-pycloud: An open-source, cloud computing-based pipeline for lc-ms/ms data analysis. *bioRxiv*, page 320887, 2018.

[DCL05] Dexter T Duncan, Robertson Craig, and Andrew J Link. Parallel tandem: a program for parallel processing of tandem mass spectra using

pvm or mpi and x! tandem. *Journal of proteome research*, 4(5):1842–1847, 2005.

[DLZ+19] Arun Devabhaktuni, Sarah Lin, Lichao Zhang, Kavya Swaminathan, Carlos G Gonzalez, Niclas Olsson, Samuel M Pearlman, Keith Rawson, and Joshua E Elias. Taggraph reveals vast protein modification landscapes from large tandem mass spectrometry datasets. *Nature biotechnology*, page 1, 2019.

[DM13] Sven Degroeve and Lennart Martens. Ms2pip: a tool for ms/ms peak intensity prediction. *Bioinformatics*, 29(24):3199–3203, 2013.

[DRP+19] Yamei Deng, Zhe Ren, Qingfei Pan, Da Qi, Bo Wen, Yan Ren, Huanming Yang, Lin Wu, Fei Chen, and Siqi Liu. pclean: an algorithm to preprocess high-resolution tandem mass spectra for database searching. *Journal of proteome research*, 18(9):3235–3244, 2019.

[DSPW09] Jiarui Ding, Jinhong Shi, Guy G Poirier, and Fang-Xiang Wu. A novel approach to denoising ion trap tandem mass spectra. *Proteome Science*, 7(1):9, 2009.

[DW19] Nan Ding and Samuel Williams. An instruction roofline model for gpus. In *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 7–18, 2019.

[EJH13] Jimmy K Eng, Tahmina A Jahan, and Michael R Hoopmann. Comet: an open-source ms/ms sequence database search tool. *Proteomics*, 13(1):22–24, 2013.

[EMY94] Jimmy K Eng, Ashley L McCormack, and John R Yates. An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. *Journal of the American Society for Mass Spectrometry*, 5(11):976–989, 1994.

[FB03] David Fenyö and Ronald C Beavis. A method for assessing the statistical significance of mass spectrometry-based protein identifications using general scoring schemes. *Analytical chemistry*, 75(4):768–774, 2003.

[GMK+04] Lewis Y Geer, Sanford P Markey, Jeffrey A Kowalak, Lukas Wagner, Ming Xu, Dawn M Maynard, Xiaoyu Yang, Wenyao Shi, and Stephen H

Bryant. Open mass spectrometry search algorithm. *Journal of proteome research*, 3(5):958–964, 2004.

[GSG+22] Siegfried Gessulat, Tobias Schmidt, Michael Graber, Samia Ben Fredj, Lizi Mamisashvili, Patroklos Samaras, Florian Seefried, Magnus Rathke-Kuhnert, Daniel P Zolg, and Martin Frejno. An end-to-end machine learning workflow for ms-based proteomics. In *International Mass Spectrometry Conference (IMSC)*, 2022.

[GSZ+19] Siegfried Gessulat, Tobias Schmidt, Daniel Paul Zolg, Patroklos Samaras, Karsten Schnatbaum, Johannes Zerweck, Tobias Knaute, Julia Rechenberger, Bernard Delanghe, Andreas Huhmer, et al. Prosit: proteome-wide prediction of peptide tandem mass spectra by deep learning. *Nature methods*, 16(6):509–518, 2019.

[HAS19] Muhammad Haseeb, Fatima Afzali, and Fahad Saeed. Lbe: A computational load balancing algorithm for speeding up parallel peptide search in mass-spectrometry based proteomics. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 191–198. IEEE, 2019.

[HDDA21] Muhammad Haseeb, Nan Ding, Jack Deslippe, and Muaaz Awan. Evaluating performance and portability of a core bioinformatics kernel on multiple vendor gpus. In *2021 International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pages 68–78. IEEE, 2021.

[HL15] Pinjie He and Kenli Li. Mic-tandem: parallel x! tandem using mic on tandem mass spectrometry based proteomics data. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 717–720. IEEE, 2015.

[HRB+14] Alexander S Hebert, Alicia L Richards, Derek J Bailey, Arne Ulbrich, Emma E Coughlin, Michael S Westphall, and Joshua J Coon. The one hour yeast proteome. *Molecular & Cellular Proteomics*, 13(1):339–347, 2014.

[HS19] Muhammad Haseeb and Fahad Saeed. Efficient shared peak counting in database peptide search using compact data structure for fragmention index. In *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 275–278. IEEE, 2019.

[HS21] Muhammad Haseeb and Fahad Saeed. High performance computing framework for tera-scale database search of mass spectrometry data. *Nature Computational Science*, 1(8):550–561, 2021.

[KBC+18] Patricia Kaiser, Maya Bode, Astrid Cornils, Wilhelm Hagen, Pedro Martínez Arbizu, Holger Auel, and Silke Laakmann. High-resolution community analysis of deep-sea copepods using maldi-tof protein fingerprinting. *Deep Sea Research Part I: Oceanographic Research Papers*, 138:122–130, 2018.

[KHUP18] Hyunwoo Kim, Sunggeun Han, Jung-Ho Um, and Kyongseok Park. Accelerating a cross-correlation score function to search modifications using a single gpu. *BMC bioinformatics*, 19(1):1–5, 2018.

[KKCB09] Gaurav Kulkarni, Ananth Kalyanaraman, William R Cannon, and Douglas Baxter. A scalable parallel approach for peptide identification from large-scale mass spectrometry data. In *2009 International Conference on Parallel Processing Workshops*, pages 423–430. IEEE, 2009.

[KLA+17] Andy T Kong, Felipe V Leprevost, Dmitry M Avtonomov, Dattatreya Mellacheruvu, and Alexey I Nesvizhskii. Msfragger: ultrafast and comprehensive peptide identification in mass spectrometry–based proteomics. *Nature methods*, 14(5):513, 2017.

[KP14] Sangtae Kim and Pavel A Pevzner. Ms-gf+ makes progress towards a universal database search tool for proteomics. *Nature communications*, 5:5277, 2014.

[LA10] Henry Lam and Ruedi Aebersold. Spectral library searching for peptide identification via tandem ms. In *Proteome Bioinformatics*, pages 95–103. Springer, 2010.

[LaV03] Joseph J LaViola. Double exponential smoothing: an alternative to kalman filter-based predictive tracking. In *Proceedings of the workshop on Virtual environments 2003*, pages 199–206, 2003.

[LC12] You Li and Xiaowen Chu. Speeding up scoring module of mass spectrometry based protein identification by gpu. In *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*, pages 1315–1320. IEEE, 2012.

[LCW+10] You Li, Hao Chi, Le-Heng Wang, Hai-Peng Wang, Yan Fu, Zuo-Fei Yuan, Su-Jun Li, Yan-Sheng Liu, Rui-Xiang Sun, Rong Zeng, et al. Speeding up tandem mass spectrometry based database searching by peptide and spectrum indexing. *Rapid Communications in Mass Spectrometry*, 24(6):807–814, 2010.

[LCXC14] You Li, Hao Chi, Leihao Xia, and Xiaowen Chu. Accelerating the scoring module of mass spectrometry-based peptide identification using gpus. *BMC bioinformatics*, 15(1):1–11, 2014.

[LDE+06] Henry Lam, Eric Deutsch, James Eddes, Jimmy Eng, Nichole King, Sara Yang, Jeri Roth, Lisa Kilpatrick, Pedatsur Neta, Steve Stein, et al. Spectrast: An open-source ms/ms spectramatching library search tool for targeted proteomics. In *Poster at 54th ASMS Conference on Mass Spectrometry*, pages 1–2, 2006.

[LLC+19] C Li, K Li, T Chen, Y Zhu, and Q He. Sw-tandem: A highly efficient tool for large-scale peptide sequencing with parallel spectrum dot product on sunway taihulight. *Bioinformatics (Oxford, England)*, 2019.

[LLLL19] Chuang Li, Kenli Li, Keqin Li, and Feng Lin. Mctandem: an efficient tool for large-scale peptide identification on many integrated core (mic) architecture. *BMC bioinformatics*, 20(1):397, 2019.

[LWS+14] Yu Jung Lo, Samuel Williams, Brian Van Straalen, Terry J Ligocki, Matthew J Cordery, Nicholas J Wright, Mary W Hall, and Leonid Oliker. Roofline model toolkit: A practical tool for architectural and program analysis. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, pages 129–148. Springer, 2014.

[LXCC14] You Li, Leihao Xia, Hao Chi, and Xiaowen Chu. Accelerating mass spectrometry-based protein identification using gpus. *BMC Bioinformatics*, 2014.

[MAB+20] Jonathan R Madsen, Muaaz G Awan, Hugo Brunie, Jack Deslippe, Rahul Gayatri, Leonid Oliker, Yunsong Wang, Charlene Yang, and Samuel Williams. Timemory: Modular performance analysis for hpc. In *International Conference on High Performance Computing*, pages 434–452. Springer, 2020.

[Mar13] Vivien Marx. Biology: The big challenges of big data, 2013.

[MBDH99] Philip J Mucci, Shirley Browne, Christine Deane, and George Ho. Papi: A portable interface to hardware performance counters. In *Proceedings of the department of defense HPCMP users group conference*, volume 710, 1999.

[MDWC07] Lijuan Mo, Debojyoti Dutta, Yunhu Wan, and Ting Chen. Msnovo: a dynamic programming algorithm for de novo peptide sequencing via tandem mass spectrometry. *Analytical chemistry*, 79(13):4870–4878, 2007.

[MFG12] Jeffrey A Milloy, Brendan K Faherty, and Scott A Gerber. Tempest: Gpu-cpu computing for high-throughput database spectral matching. *Journal of proteome research*, 11(7):3581–3591, 2012.

[MTKF+14] Sean McIlwain, Kaipo Tamura, Attila Kertesz-Farkas, Charles E Grant, Benjamin Diament, Barbara Frewen, J Jeffry Howbert, Michael R Hoopmann, Lukas Kall, Jimmy K Eng, et al. Crux: rapid open source protein tandem mass spectrometry analysis. *Journal of proteome research*, 13(10):4488–4491, 2014.

[MVN12] Kelvin Ma, Olga Vitek, and Alexey I Nesvizhskii. A statistical model-building perspective to identification of ms/ms spectra with peptideprophet. *BMC bioinformatics*, 13(16):S1, 2012.

[Nes10] Alexey I Nesvizhskii. A survey of computational methods and error rate estimation procedures for peptide and protein identification in shotgun proteomics. *Journal of proteomics*, 73(11):2092–2123, 2010.

[NRG+06] Alexey I Nesvizhskii, Franz F Roos, Jonas Grossmann, Mathijs Vogelzang, James S Eddes, Wilhelm Gruissem, Sacha Baginsky, and Ruedi Aebersold. Dynamic spectrum quality assessment and iterative computational analysis of shotgun proteomic data toward more efficient identification of post-translational modifications, sequence polymorphisms, and novel peptides. *Molecular & Cellular Proteomics*, 5(4):652–670, 2006.

[PHTN11] Brian Pratt, J Jeffry Howbert, Natalie I Tasman, and Erik J Nilsson. Mr-tandem: parallel x! tandem using hadoop mapreduce on amazon web services. *Bioinformatics*, 28(1):136–137, 2011.

[PPCC99] 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: An International Journal*, 20(18):3551–3567, 1999.

[QTL+19] Rui Qiao, Ngoc Hieu Tran, Ming Li, Lei Xin, Baozhen Shan, and Ali Ghodsi. Deepnovov2: Better de novo peptide sequencing with deep learning. *arXiv preprint arXiv:1904.08514*, 2019.

[RA19] S Rossel and P Martínez Arbizu. Revealing higher than expected diversity of harpacticoida (crustacea: Copepoda) in the north sea using maldi-tof ms and molecular barcoding. *Scientific reports*, 9(1):1–14, 2019.

[RSSK14] V Srinivasa Rao, K Srinivas, GN Sujini, and GN Kumar. Protein-protein interaction detection: methods and analysis. *International journal of proteomics*, 2014, 2014.

[SHI22] Fahad Saeed, Muhammad Haseeb, and SS Iyengar. Communication lower-bounds for distributed-memory computations for mass spectrometry based omics data. *Journal of Parallel and Distributed Computing*, 161:37–47, 2022.

[SK15] Owen S Skinner and Neil L Kelleher. Illuminating the dark matter of shotgun proteomics. *Nature biotechnology*, 33(7):717, 2015.

[TCD+14] John Towns, Timothy Cockerill, Maytal Dahan, Ian Foster, Kelly Gaither, Andrew Grimshaw, Victor Hazlewood, Scott Lathrop, Dave Lifka, Gregory D Peterson, et al. Xsede: accelerating scientific discovery. *Computing in Science & Engineering*, 16(5):62–74, 2014.

[THA+20] Muhammad Usman Tariq, Muhammad Haseeb, Mohammed Aledhari, Rehma Razzak, Reza M Parizi, and Fahad Saeed. Methods for proteogenomics data analysis, challenges, and scalability bottlenecks: a survey. *IEEE Access*, 9:5497–5516, 2020.

[THS+05] Wilfred H Tang, Benjamin R Halpern, Ignat V Shilov, Sean L Seymour, Sean P Keating, Alex Loboda, Alpesh A Patel, Daniel A Schaeffer, and Lydia M Nuwaysir. Discovering known and unanticipated protein modifications using ms/ms database searching. *Analytical Chemistry*, 77(13):3931–3946, 2005.

[Tis11] Alexander Tiskin. *BSP (Bulk Synchronous Parallelism)*, pages 192–199. Springer US, Boston, MA, 2011.

[TS21] Muhammad Usman Tariq and Fahad Saeed. Specollate: Deep cross-modal similarity network for mass spectrometry data based peptide deductions. *PloS one*, 16(10):e0259349, 2021.

[TSY03] David L Tabb, Anita Saraf, and John R Yates. Gutentag: high-throughput sequence tagging via an empirically derived fragmentation model. *Analytical chemistry*, 75(23):6415–6421, 2003.

[Val90] Leslie G Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[WTE07] Xue Wu, Chau-Wen Tseng, and Nathan Edwards. Hmmatch: peptide identification by spectral matching of tandem mass spectra using hidden markov models. *Journal of Computational biology*, 14(8):1025–1043, 2007.

[WWP09] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[XPV+15] TPSK Xu, SK Park, JD Venable, JA Wohlschlegel, JK Diedrich, D Cociorva, B Lu, L Liao, J Hewel, X Han, et al. Prolucid: An improved sequest-like algorithm with enhanced sensitivity and specificity. *Journal of proteomics*, 129:16–24, 2015.

[YCZ+17] Hao Yang, Hao Chi, Wen-Jing Zhou, Wen-Feng Zeng, Kun He, Chao Liu, Rui-Xiang Sun, and Si-Min He. Open-pnovo: de novo peptide sequencing with thousands of protein modifications. *Journal of proteome research*, 16(2):645–654, 2017.

[YFS+10] Ding Ye, Yan Fu, Rui-Xiang Sun, Hai-Peng Wang, Zuo-Fei Yuan, Hao Chi, and Si-Min He. Open ms/ms spectral library search to identify unanticipated post-translational modifications and increase spectral identification rate. *Bioinformatics*, 26(12):i399–i406, 2010.

[YI19] John R Yates III. Proteomics of communities: metaproteomics, 2019.

[ZXS+12] Jing Zhang, Lei Xin, Baozhen Shan, Weiwu Chen, Mingjie Xie, Denis Yuen, Weiming Zhang, Zefeng Zhang, Gilles A Lajoie, and Bin Ma.

Peaks db: de novo sequencing assisted database search for sensitive and accurate peptide identification. *Molecular & Cellular Proteomics*, 11(4):M111–010587, 2012.

[ZZW+22] Wen-Feng Zeng, Xie-Xuan Zhou, Sander Willems, Constantin Ammar, Maria Wahle, Isabell Bludau, Eugenia Voytik, Maximillian T Strauss, and Matthias Mann. Alphapeptdeep: a modular deep learning framework to predict peptide properties for proteomics. *Nature communications*, 13(1):1–14, 2022.

APPENDIX A: CODE AVAILABILITY

All the software frameworks discussed in this dissertation are available as open-source.

HiCOPS and GiCOPS

Implemented using MPI, C++17, CUDA, OpenMP, Python, Bash, Timemory [MAB+20], PAPI [MBDH99] and CMake, and is available and maintained at: https://github.com/pcdslab/gicops. The legacy HiCOPS code repository and webpage are also available at https://github.com/hicops/hicops and https://hicops.github.io respectively.

LBE and CFIR-Index

Implemented using C++17, Python and Bash and available as a part of the HiCOPS/GiCOPS framework.

APPENDIX B: DATA AVAILABILITY

All datasets and databases used in this work are publicly available.

Datasets

All datasets used in this dissertation are available from the public Pride Archive repository using the link: https://www.ebi.ac.uk/pride/archive/projects/<AN> where <AN> is the dataset's accession number. For example, the dataset with the accession number: PXD012345, can be accessed via the link: https://www.ebi.ac.uk/pride/archive/projects/PXD012345.

Databases

The proteome sequence databases used in this dissertation are publicly available from UniProt via: https://www.uniprot.org/proteomes/<ID>, where <ID> is the Proteome ID of the species. For example, the Homo sapiens proteome sequence database with Proteome ID UP000005640 can be accessed via the link: https://www.uniprot.org/proteomes/UP000005640 and the SwissProt database is available at: https://www.uniprot.org/uniprot/?query=reviewed:yes.

APPENDIX C: FUNDINGS AND COMPUTE RESOURCES

This work used the NSF Extreme Science and Engineering Discovery Environment (XSEDE) Supercomputers - now called Advanced Cyberinfrastructure Coordination Ecosystem: Services and Support (ACCESS) - through allocations: TG-CCR150017 and TG-ASC200004, as well as the Dragon High-Performance Computing cluster resources at the Florida International University supported by the NIH Supplemental Award: 3R01GM134384-02S1. The research work was partially supported by the NIGMS of the National Institutes of Health (NIH) under award number: R01GM134384. Additionally this research work was partially supported by National Science Foundations (NSF) under the award number: NSF CAREER OAC-1925960. The content in this dissertation is solely the responsibility of the author(s) and does not necessarily represent the official views of the National Institutes of Health and/or National Science Foundation.

VITA

MUHAMMAD HASEEB

	Born, Faisalabad, Pakistan
2015	BS, Electrical Engineering University of Engineering and Technology Lahore, Pakistan
2015	Software Engineer (L1) Mentor Graphics Corporation Lahore, Pakistan
2016	Software Engineer (L2) Mentor Graphics Corporation Lahore, Pakistan
2017	Graduate Research Assistant Western Michigan University Kalamazoo, Michigan
2018	Graduate Research Assistant Florida International University Miami, Florida
2020	Application Performance Intern Lawrence Berkeley National Laboratory Berkeley, California
2021	GPU Performance Intern Lawrence Berkeley National Laboratory Berkeley, California
2023	PhD, Computer Science Florida International University Miami, Florida
2023	Application Performance Postdoc Lawrence Berkeley National Laboratory Berkeley, California

PUBLICATIONS AND PRESENTATIONS

Muhammad Haseeb, and Fahad Saeed., (2022) *GPU-Acceleration of the Distributed-Memory Database Peptide Search on Supercomputers.* ASMS Conference on Mass Spectrometry and Allied Topics.

Fahad Saeed, and Muhammad Haseeb., (2022) *High-Performance Algorithms for Mass Spectrometry-Based Omics.*, ISBN 9783031019593, Springer Nature Switzerland AG.

Muhammad Haseeb, and Fahad Saeed., (2022) *Systems and Methods for Peptide Identification.* U.S. Patent 11,309,061.

Muhammad Haseeb, Nan Ding, Jack Deslippe, and Muaaz Awan., (2021) *Evaluating Performance and Portability of a core bioinformatics kernel on multiple vendor GPUs.* International Workshop on Performance, Portability and Productivity in HPC (P3HPC).

Fahad Saeed, Muhammad Haseeb, and S. S. Iyengar., (2021) *Communication lower-bounds for distributed-memory computations for mass-spectrometry based omics data.* Journal of Parallel and Distributed Computing.

Muhammad Haseeb, and Fahad Saeed., (2021) *High performance computing framework for tera-scale database search of mass spectrometry data.* Nature Computational Science, 1 (8): 550-561

Muhammad Usman Tariq, Muhammad Haseeb, Mohammed Aledhari, Rehma Razzak, Reza M. Parizi, and Fahad Saeed., (2020) *Methods for Proteogenomics Data Analysis, Challenges, and Scalability Bottlenecks: A Survey.* IEEE Access.

Fahad Saeed, and Muhammad Haseeb., (2020) *Methods and systems for compressing data.* U.S. Patent 10,810,180.

Muhammad Haseeb and Fahad Saeed., (2019) *Efficient Shared Peak Counting in Database Peptide Search Using Compact Data Structure for Fragment-Ion Index.* IEEE International Conference on Bioinformatics and Biomedicine, 275-278.

Muhammad Haseeb, Fatima Afzali, and Fahad Saeed., (2019) *LBE: A Computational Load Balancing Algorithm for Speeding up Parallel Peptide Search in Mass-Spectrometry based Proteomics.* IEEE International Parallel and Distributed Processing Symposium Workshops, 191-198.