

12-10-2021

High Performance Computing Framework for Tera-Scale Database Search of Mass Spectrometry Data

Muhammad Haseeb

Fahad Saeed

Follow this and additional works at: https://digitalcommons.fiu.edu/cs_fac

This work is brought to you for free and open access by the College of Engineering and Computing at FIU Digital Commons. It has been accepted for inclusion in School of Computing and Information Sciences by an authorized administrator of FIU Digital Commons. For more information, please contact dcc@fiu.edu.

High Performance Computing Framework for Tera-Scale Database Search of Mass Spectrometry Data

Muhammad Haseeb¹, and Fahad Saeed^{1,2,3,4}

¹*Knight Foundation School of Computing and Information
Sciences,*

²*Biomolecular Sciences Institute (BSI),*

³*Department of Human and Molecular Genetics,
Herbert Wertheim School of Medicine,*

Florida International University, Miami, FL, USA.

⁴*Corresponding Author. Email: fsaeed@fiu.edu*

December 10, 2021

Abstract

Database peptide search algorithms deduce peptides from mass spectrometry (MS) data. There has been substantial effort in improving their computational efficiency to achieve larger and more complex systems biology studies. However, modern serial and high-performance computing (HPC) algorithms exhibit sub-optimal performance mainly due to their ineffective parallel designs (low resource utilization), and high overhead costs.

We present an HPC framework, called HiCOPS, for efficient acceleration of the database peptide search algorithms on distributed-memory supercomputers. HiCOPS provides, on average, more than 10-fold improvement in speed, and superior parallel performance over several existing HPC database search software. We also formulate a mathematical model for performance analysis and optimization, and report near-optimal results for several key metrics including strong-scale efficiency, hardware utilization, load-balance, inter-process communication and I/O overheads. The core parallel design, techniques, and optimizations presented in HiCOPS are search-algorithm independent and can be extended to efficiently accelerate the existing and future algorithms and software.

Keywords— mass spectrometry, high performance computing, proteomics, peptide identification, bulk synchronous parallel

1 Introduction

Faster, and more efficient peptide identification algorithms [1], [2], [3] have been the cornerstone of computational research in shotgun mass spectrometry (MS) based proteomics for more than 30 years [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17]. Modern mass spectrometry technologies allow the generation of thousands of raw, noisy spectra in a span of few hours producing several gigabytes of data [18] (Supplementary Figure 1). Database peptide search is the most commonly employed computational approach to identify the peptides from the experimental spectra [19], [10], [2], [20]. In this approach, the experimental spectra are searched against an (indexed) database of theoretical spectra (or modeled-spectra) with the goal to find the best possible matches [1]. The theoretical spectra database (or simply theoretical database) is simulated by simulating in-silico digestion on a proteome sequence database (Supplementary Figure 2). The theoretical databases (and their indexed versions) expand exponentially in space (several giga to terabytes) as the post-translational modifications (PTMs) are added in the simulation [2], [21] (Supplementary Figure 3a, b). Consequently, the low computational arithmetic intensity (operations or instructions per byte) [22] inherent to database search algorithms [2], [23], [9] results in performance bottlenecks due to memory contention (parallel database query), out-of-core processing (database size > main memory), database management (data movement), and I/O.

As demonstrated by other scientific fields [24], these limitations can be alleviated through effective exploitation of architectural resources provided by modern high-performance computing (HPC) systems. However, most existing HPC database peptide search algorithms [25], [26], [27], [28], [29], [30], [31] employ unoptimized parallelization techniques, resulting in sub-optimal performance and limited application in the domain (Supplementary Section 1, Supplementary Section 2, Supplementary Figure 3c). The need for efficient parallel database peptide search software is driven by the computational demands of modern systems biology studies for proteomics, metaproteomics and proteogenomics, where peptide identification is often the first step in the analysis. These systems biology studies also have a direct impact on personalized nutrition, microbiome research [32], [33], and cancer therapeutics [34].

In this paper, we present an HPC framework for efficient acceleration of database peptide search algorithms on large-scale symmetric multiprocessor (SMP) distributed-memory supercomputers. HiCOPS provides orders-of-magnitude improvement in speed compared to several existing shared- and distributed-memory database peptide search tools allowing searching of several gigabyte experimental MS/MS data against terabytes of theoretical databases in a few minutes compared to several hours required using existing algorithms. The proposed HiCOPS parallel design implements an unconventional approach where the (massive) theoretical databases are distributed across parallel nodes in a load-balanced fashion followed by asynchronous parallel execution of the database peptide search. Upon completion, the locally computed results are merged into global results in a communication-optimal manner. This overhead cost-optimal design, along with several optimizations, allows HiCOPS to maximize resource utilization and alleviate the performance bottlenecks. We also formulate and perform a performance analysis to identify the overhead costs and discuss optimization techniques to minimize them. Finally, we implement a shared-peak counting coupled hyperscore-based search algorithm [11], [2], [35] in HiCOPS to demonstrate its parallel performance, but in essence, our framework is search-algorithm oblivious. i.e. the proposed parallel design, algorithms and optimizations can be extended or replaced to

accelerate most existing and future search algorithms.

Our comprehensive experimentation shows that HiCOPS outperforms several existing serial and parallel database peptide search tools by more than 10-folds on average while producing correct and consistent peptide identifications. Additionally, we demonstrate the application of HiCOPS in large-scale database search setting through multiple compute- and data-intensive experiments. Note that the HiCOPS framework does not propose a new database search algorithm and instead relies on the underlying (portable) search algorithmic workflow for peptide identification accuracy. Finally, we performed an extensive performance evaluation where we report between 70-80% strong-scale efficiency and less than 25% overall performance overheads (load imbalance, I/O, inter-process communication, pipeline halt); collectively depicting a near-optimal parallel performance.

2 Results

2.1 Methods Overview

HiCOPS constructs the parallel database peptide search workflow (task-graph) through *four* Single Program Multiple Data (SPMD) Bulk Synchronous Parallel (BSP) [36] supersteps. In the BSP model, a *superstep* [37] refers to a set of distinct algorithmic and data communications blocks, asynchronously executed by all parallel processes ($p_i \in P$). Synchronization between the processes is done at the end of each superstep, as needed. In the first HiCOPS’s superstep, the (massive) theoretical database is partitioned across parallel processes in a load balanced fashion, and locally indexed. In the second superstep, the experimental data are divided into batches and pre-processed, if required. In the third superstep, the parallel processes execute a local database peptide search, producing intermediate results. In the final superstep, the intermediate results are de-serialized, and assembled into complete (global) results. Supplementary Figure 4 provides an overview of the overall task-graph, and workload profile for each superstep (Methods). The current HiCOPS design allows in-core processing so the minimum number of nodes P_{min} required to run must be $\geq D/M$ where D is the theoretical database index size and M is the available main memory per node.

The total wall time (T_H) for executing the four supersteps is the sum of superstep execution times, given as:

$$T_H = T_1 + T_2 + T_3 + T_4$$

Where the execution time for a superstep (j) is the maximum time required by any parallel task ($p_i \in P$) to complete that superstep, given as:

$$T_j = \max(T_{j,p_1}, T_{j,p_2}, \dots, T_{j,p_P})$$

Or simply:

$$T_j = \max_{p_i}(T_{j,p_i})$$

Combining the above three equations, the total HiCOPS runtime is given as:

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

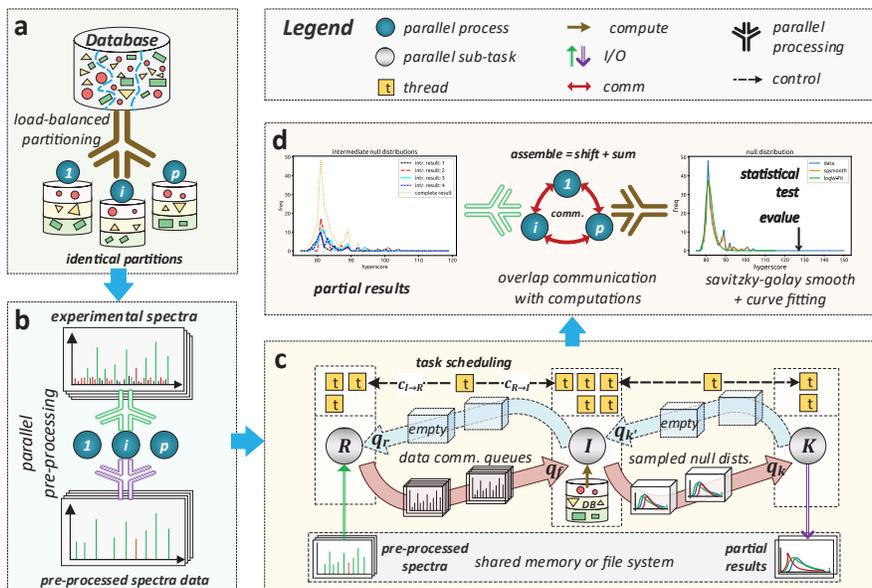


Figure 1: **Methods Overview.** (a) **Superstep 1:** The massive theoretical spectra database (spectra shown as shapes) is partitioned among parallel processes and locally indexed. Partitioning is done in a load balanced fashion (similar shapes clustered and scattered across processes). (b) **Superstep 2:** The experimental MS/MS spectra data are indexed, tagged, pre-processed and written back to a shared memory in data parallel. (c) **Superstep 3:** Asynchronous parallel database peptide search is executed by all processes. On each process, three parallel sub-tasks R , I and K work in a pipeline to load the pre-processed data, execute a local database search, and write the produced (sampled) local results to the shared memory respectively. Task scheduler manages the parallel threads between the pipeline tasks. (d) **Superstep 4:** Local/intermediate results are assembled followed by curve fitting and expected value (es) computation in data parallel fashion. Results with $es < 0.01$ are communicated to their origin processes.

2.2 Experimental Setup Overview

We constructed five custom datasets (S_i) by combining several Pride Archive (PXD) datasets (accession numbers: PXDxxxxxx) for our experimentation and evaluation. These five custom datasets are given as follows: S_1 : PXD009072, S_2 : PXD020590, S_3 : PXD015890, S_4 : PXD007871, 009072, 010023, 012463, 013074, 013332, 014802, 015391 combined, and S_5 : all above listed datasets combined. The datasets were searched against several theoretical databases constructed by adding combinations of post-translational modifications (PTMs) to the D_1 : UniProt Homo sapiens (UP000005640) and D_2 : UniProt SwissProt (reviewed) databases. Detailed discussion about the settings for database digestion, post-translational modifications, theoretical spectra generation etc. is provided in the (Methods) section. In the rest of the paper, we will

represent the workload size for each performed experiment (exp_n) as a tuple given as: $\text{exp}_n = (q, D, \delta M)$; where q is experimental MS/MS dataset size in 1 million spectra, D is theoretical database size in 100 million spectra and δM is the peptide precursor mass tolerance setting in $\pm 100\text{Da}$. Note that the tuple does not contain the fragmentation mass tolerance (δF) information as it is globally set to $\pm 0.01\text{ Da}$ unless specifically mentioned as the fourth element in an experiment tuple.

Runtime Environment: All experiments were run on the Extreme Science and Engineering Discovery Environment (XSEDE) [38] Comet cluster at the San Diego Supercomputer Center (SDSC). The Comet compute nodes are equipped with 2 sockets \times 12 cores of Intel Xeon E5-2680v3 processor (Total: 24 cores), 2 NUMA nodes \times 64GB (Total: 128GB) DRAM, 56 Gbps FDR InfiniBand interconnect and Lustre shared file system. The maximum number of nodes allowed per job is 72 and maximum allowed job time is 48 hours. Furthermore, the single-node experiments for Crux and X!Tandem tools requiring $>48\text{h}$ (XSEDE limit) execution time were run on a (comparable) local machine named *raptor*, equipped with Intel Xeon Gold 6152 processor (22 cores), 128GB DRAM and 6TB SSD HDD.

2.3 Correctness Analysis

We evaluated the HiCOPS’s correctness using a two-step approach. In the first step, we verified the consistency of results across parallel runs by searching all five datasets S_i against both protein sequence databases D_i using various settings and PTM combinations. The correctness was evaluated in terms of identified peptide sequences, and the corresponding hyperscores and expected values (expectscores) assigned (within 3 decimal points). A comparison of hyperscores and expectscores between the serial (x-axis) and parallel runs (y-axis), obtained by searching the dataset: S_1 against the database: D_1 with no PTMs is shown in Figure 2a, 2b respectively. The results show over 99.5% consistency in scores. A small error was observed in a negligible number of results due to the sampling and floating-point precision losses (Methods, Figure 1d).

In the second step, we verified the quality of the implemented search algorithm by comparing the HiCOPS-computed hyperscores with the MSFragger-computed ones as both frameworks employ a similar scoring algorithm. i.e. shared-peak counting coupled hyperscore. Note that the hyperscores computed by MSFragger and HiCOPS cannot be exactly identical as MSFragger uses several pre-processing and boosting features that affect the final scores. These features could not be replicated in HiCOPS as MSFragger is a proprietary software. We designed and executed six experiments, three with restricted search ($\delta M=1\text{Da}$) and three with open search ($\delta M \geq 100\text{Da}$) setting. The experimental MS/MS data pre-processing and database search settings were kept identical (and as minimal as possible) for both tools for fair comparisons. The details of the six experiments are as follows:

In the first experiment, a subset of 860 thousand spectra from the dataset: S_4 was searched against the database: D_1 modified with Methionine oxidation and NQ-deamidation as PTMs yielding a theoretical database of 18 million spectra at $\delta M=1\text{Da}$. In the second experiment, the dataset: S_3 was searched against the database: D_1 modified with Methionine oxidation and STY-phosphorylation yielding a theoretical database of 66 million spectra at $\delta M=1\text{Da}$. In the third experiment, the dataset: S_3 was searched against the database: D_2 modified with Methionine oxidation and Serine phosphorylation yielding a database of 80 million spectra at $\delta M=1\text{Da}$. In the fourth experiment: the entire dataset: S_3 was searched against the database: D_1 with Methionine oxidation and NQ-deamidation yielding a theoretical database of 18

million spectra at $\delta M=200\text{Da}$. In the fifth experiment, the S_3 was searched against the database: D_1 modified with Methionine oxidation and ST-phosphorylation yielding a theoretical database of 56 million spectra at $\delta M=100\text{Da}$. In the sixth experiment, dataset: S_3 was searched against the database: D_2 modified with Methionine oxidation and Serine phosphorylation yielding a database of 80 million spectra at $\delta M=200\text{Da}$.

For our comparisons, first, a correlation between the hyperscores assigned by both tools to commonly identified peptide to spectrum matches (PSMs) was computed (shown in Figures 2c to 2h). Then, the PSMs from both tools were filtered at 1% q-value (false discovery rate) and compared (shown in Supplementary Figure 5). Figures 2c, 2d, and 2e respectively depict a strong-correlation (pearson coefficient $R \geq 0.90$) between the hyperscores computed by both tools in the first three (restricted-search) experiments. However, the correlation between the hyperscores slightly drops between $0.70 \leq R \leq 0.90$ for the last three (open-search) experiments (Figures 2f, 2g, and 2h respectively). We suspect that the divergence in hyperscores may have stemmed from open-search specific spectral processing, reconstruction and/or score re-ranking algorithms implemented in MSFragger. Further, the results in Supplementary Figure 5 show about 50% overlap between the q-value filtered PSMs from HiCOPS and MSFragger. The results also show that the MSFragger’s scoring algorithm outperformed the underlying scoring algorithm in HiCOPS in identified peptides, as expected. Recall that the HiCOPS is designed as algorithm oblivious; meaning the underlying algorithms can be customized or ported with more sophisticated versions to improve the identification while delivering similar performance.

2.4 Speed Comparison Against Existing Algorithms

We compared the HiCOPS speed against many existing shared- and distributed-memory database peptide search algorithms including Tide/Crux v3.2 [3], Comet v2020.01 [40], MSFragger v3.0 [2], X! Tandem v17.2.1 [41], X!! Tandem v10.12.1 [26], and SW-Tandem [29]. Parallel versions of the shared-memory tools were also implemented and run through Python and Bash wrapper scripts executing the following workflow: run parallel instances of the tool on XSEDE Comet nodes with equal partitions (random partitioning) of the experimental MS/MS data files. This technique also indirectly simulated the workflows of cloud-based tools such as MS-PyCloud (via parallel MSGF+) and Bolt (via parallel MSFragger). Additionally, we tried to run the UltraQuant HPC tool which implements a parallel MaxQuant. However, it crashed with unhandled exceptions every time it was run on >1 node.

We designed six experiments listed as **(a)** to **(f)** in increasing order of their experimental workload sizes. (i.e. database and dataset sizes, and experimental settings). In the first two **(a, b)** experiment, a subset of 8000 spectra from dataset: S_3 (file: 7Sep18_Olson_WT24) was searched against the database: D_2 modified with variable Methionine oxidation, and Tyrosine Biotin-tyramide yielding a theoretical database of 93.5 million spectra at $\delta M=10\text{Da}$ and $\delta M=500\text{Da}$ respectively. In the third experiment **(c)**, the dataset: S_3 was searched against database D_1 modified with variable Methionine oxidation, and Tyrosine Biotin-tyramide as PTMs yielding a theoretical database of 7.1 million spectra at $\delta M=500\text{Da}$. In the fourth **(d)** and fifth **(e)** experiments, the entire dataset: S_3 was searched against the theoretical database of first two experiments (i.e. the 93.5 million spectra one) at $\delta M=10\text{Da}$ and $\delta M=500\text{Da}$ respectively. In the sixth **(f)** experiment, dataset S_4 was searched against the database: D_1 modified with variable Methionine oxidation, STY-phosphorylation

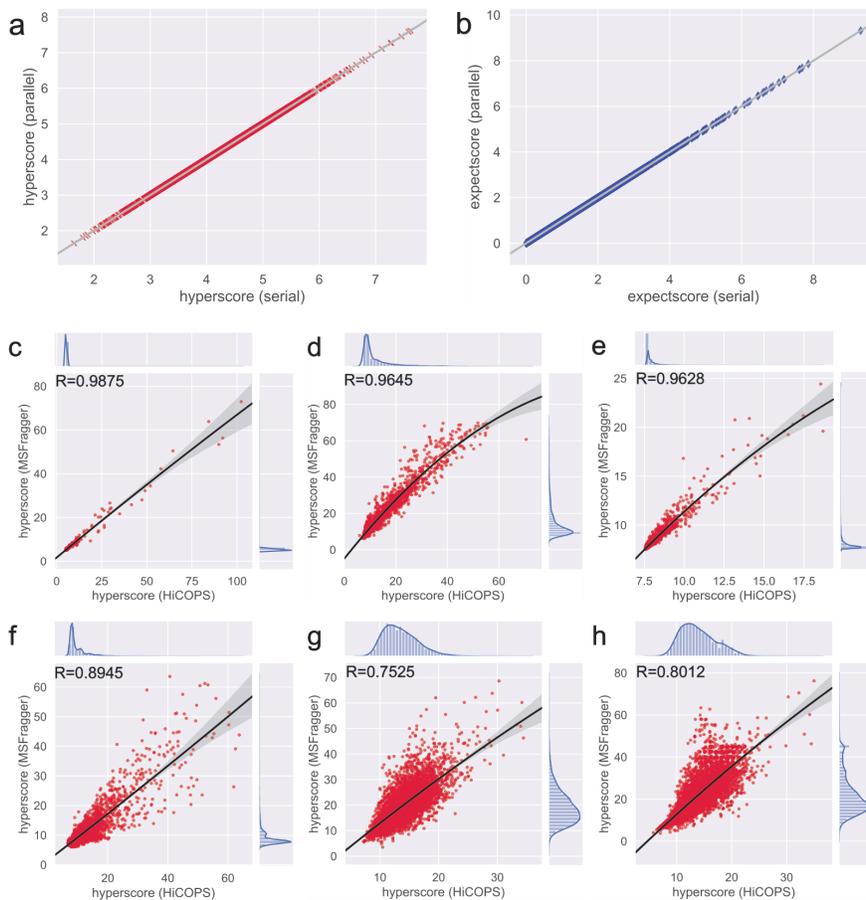


Figure 2: **Correctness Analysis.** (a,b) Comparison of 5K out of 251K data samples of hyperscores and expected values (expectcores), computed by HiCOPS in serial (x-axis) and parallel (y-axis) runs is shown. Note that all 251K samples depict the same consistency across parallel runs [39], only infeasible to plot. (c to h) Correlations between hyperscores computed by HiCOPS (x-axis) and MSFragger (y-axis) under restricted-search (c, d, e) and corresponding open-search (f, g, h) settings are shown along with Pearson correlation coefficients (R).

and NQ-deamidation yielding a theoretical database of: 213 million at $\delta M=100\text{Da}$. The slower tools such as Comet, MSGF+, Crux and X!Tandem variants were only run for smaller experiments due to XSEDE max job time limits.

The obtained wall time results (Figure 3a to 3f) show that the HiCOPS outperforms all other tools by $>10\times$ on average in speed, especially for experiments with larger workloads (Figure 3d, e, f). It can also be observed that the HiCOPS exhibits better strong-scale parallel efficiency compared to other tools as the experimental

workload size increases (a→f). For smaller workloads (Figure 3a, 3b, 3c) the parallel efficiency is limited by the Amdahl’s law. The scalability is shown as the deviation (+ve = sub-linear; -ve = hyper-linear) from the ideal speedup track (dotted gray lines in each experiment in Figure 3a to Figure 3f. The parallel efficiency results for MSFragger were particularly peculiar as it appears to be scaling super-linearly up to a certain number of parallel nodes, and then dropping to sub-linear. To explain this, the runtime components of MSFragger were further analyzed in detail. The results (Figure 3g and 3i) show that a large percentage of MSFragger’s runtime is composed of I/O, and load imbalance, resulting in low overhead/compute ratio (effective resource utilization). Comparatively, HiCOPS exhibits significantly improved memory performance (Figure 3h, 3j) resulting in lower run time even though the effective search time (useful compute time) for MSFragger and HiCOPS are comparable. The results (Figure 3a, 3b, 3c) show that the existing HPC tools including X!Tandem, SW-Tandem, parallel Comet and parallel MSGF+ (MS-PyCloud) are $> 100\times$ slow even for small-scale experiments. Finally, we observed zero parallel efficiency for SW-Tandem in all experiments, i.e. no speedups whatsoever (Supplementary Section 3).

2.5 Application in Tera-Scale Experimentation

Application of HiCOPS in tera-scale experiments was demonstrated using three additional experiments. In the first experiment, the dataset: S_3 was searched against a theoretical database of 766 million spectra (780GB) at $\delta M = \pm 500\text{Da}$ and $\delta F = \pm 0.01\text{Da}$. In the second experiment, the dataset: S_4 was searched against a theoretical database of 1.59 billion spectra (1.7TB) at $\delta M = \pm 500\text{Da}$ and $\delta F = \pm 0.05\text{Da}$. In the third experiment, the dataset: S_2 was searched against a theoretical database of 3.89 billion spectra (4TB) at $\delta M = \pm 500\text{Da}$ and $\delta F = \pm 0.01\text{Da}$. HiCOPS completed the execution of these three experiments in 14.55 minutes (64 nodes), 103.5 minutes (72 nodes) and 27.3 minutes (64 nodes) respectively. In contrast, MSFragger completed the execution of first experiment in 158.8 minutes (64 nodes; $10\times$ slower). The second experiment was completed by MSFragger in 18 hours (72 nodes; $10.3\times$ slower) and 35.5 days when using 1 node ($494\times$ slower). The other experiments were intentionally not run on MSFragger or other tools due to feasibility issues. The results for this set of experiments are summarized in Table 1.

2.6 Performance Evaluation

Twelve experiments of varying workload sizes were designed using combinations of aforementioned databases (D_i) and datasets (S_i), post-translational modifications, and precursor peptide mass tolerance windows (δM) for an extensive performance evaluation. These experimental workloads varied from extremely small to massive-scale covering a wide-range of application. The twelve experiment sets in the tuple form are listed as follows: $\text{exp}_1=(0.3, 0.84, 0.1)$, $\text{exp}_2=(0.3, 0.84, 2)$, $\text{exp}_3=(3.89, 0.07, 5)$, $\text{exp}_4=(1.51, 2.13, 5)$, $\text{exp}_5=(6.1, 0.93, 5)$, $\text{exp}_6=(3.89, 7.66, 5)$, $\text{exp}_7=(1.51, 19.54, 5)$, $\text{exp}_8=(1.6, 38.89, 5)$, $\text{exp}_9=(3.89, 15.85, 5)$, $\text{exp}_{10}=(3.89, 1.08, 5)$, $\text{exp}_{11}=(1.58, 2.13, 1)$, and $\text{exp}_{12}=(0.305, 0.847, 5)$. Note that the fragment-ion tolerance is set to $\delta F = \pm 0.01\text{Da}$ in all these experiments.

Parallel Scalability: Strong-scale efficiency for all twelve experiments was measured and the results (Figure 4a, 4b) depict that the overall strong-scale efficiency ranges between 70-80% for sufficiently large experimental workloads. For smaller experiments, the parallel speedup quickly dampens as there is not enough parallel work

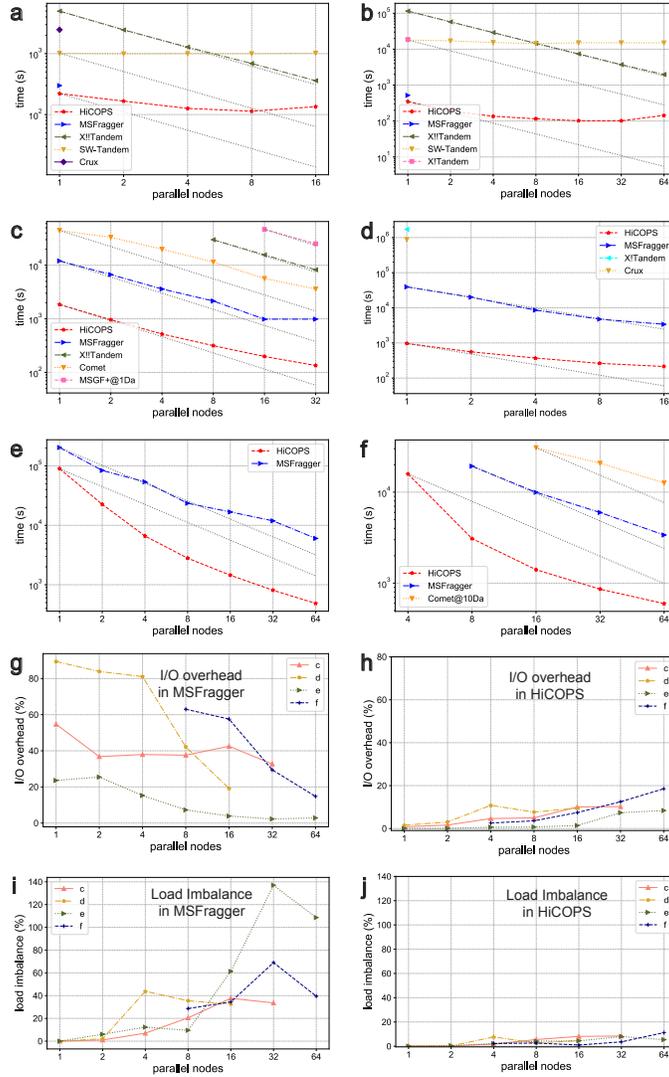


Figure 3: **Speed Comparisons.** (a to f) Speed comparison between HiCOPS and other tools with increasing number of parallel nodes is shown. The gray dotted line tracks the ideal speedup times for each tool (log-log scale) in experiments. The δM window for MSGF+ and Comet was further tightened in some experiments (indicated by '@' in labels) due to tool limitations. (g to i) The percentage I/O and load imbalance overheads exhibited by HiCOPS and MSFragger for experiments in sub-figures (c, d, e, and f), are shown with increasing number of parallel nodes.

Table 1: Summary of the execution times for three large-scale database search experiments using HiCOPS and MSFragger is shown. Peptide precursor mass tolerance and fragment-ion tolerance in Daltons (Da) are given as δM and δF respectively. Single node version of the second experiment using MSFragger (i.e. 2*) was run on the local (*raptor*) server. The third experiment was not run using MSFragger due to feasibility issues.

Experiment Number	Tool Name	Nodes	Dataset size (GB)	Database size (GB)	δM (Da)	δF (Da)	Runtime (min)
1	HiCOPS	64	20	780	500	0.01	14.55
1	MSFragger	64	20	780	500	0.01	158.8
2	HiCOPS	72	15	1692	500	0.05	103.5
2	MSFragger	72	15	1692	500	0.05	1074.45
2*	MSFragger	1	15	1692	500	0.05	51130
3	HiCOPS	64	41	4000	500	0.01	27.3

to be done (Amdahl’s Law). Superstep-level dissection of the speedup results in Supplementary Figure 6 further confirm that the most significant fraction of the overall runtime is constituted by the superstep 3 indicating its importance in optimizations. Note that the minimum number of parallel nodes (P_{min}) required by HiCOPS for each experiment must be $P_{min} \geq D/M$; where M is main memory per node. Therefore, the speedup and efficiency calculations were done using the runtime for the experiment with minimum nodes as the base case. The serial time (T_s) was first computed using the base case experiment runtime ($T_{P_{min}}$) as: $T_s = P_{min} \times T_{P_{min}}$. Then, the speedups and efficiency for experiments with nodes $\geq P_{min}$ were computed relative to $T_{P_{min}}$ using the computed T_s . Essentially, the speedups are relative to the base case runtime which may not be the 1-node time depending on the P_{min} (limitation of HiCOPS). Furthermore, super-linear speedups were observed in several experiments with larger workloads. To explain this, the following hardware counters-based metrics were also recorded for all experiments: instructions per cycle (*ipc*), last level cache misses (LLC) per all cache level misses (*lpc*), and the cycles stalled due to writes per total stalled cycles (*wps*). The results (Figure 4c, 4d, 4e) show that the CPU, cache, and memory bandwidth utilization improves as the workload per node (wf/P) increases reaching to an optimum point after which it saturates due to memory bandwidth contention since the database search algorithms employed (and also in general) are highly memory intensive. Beyond this saturation point, increasing the number of parallel nodes for the same experimental workload resulted in a substantial improvement (super-linear) in performance as the workload per node (wf/P) reduced to the normal (optimal) range. For instance, the experiment set exp₅ depicts super-linear speedups (Figure 4a) which can be correlated to the hardware performance surge in Figure 4c.

Performance Overheads: Several metrics including load imbalance, communication, I/O, and pipeline halt costs were also measured to identify and quantify the performance overheads. The obtained results (Figure 5a, 5b, 5c) depict that the load imbalance costs remain $\leq 10\%$, I/O costs remain $\leq 10\%$, and inter-task communication costs remain $\leq 5\%$ in most experiments. Note that the load imbalance is a direct measure of synchronization cost. Figure 5e shows a time series of the per-batch

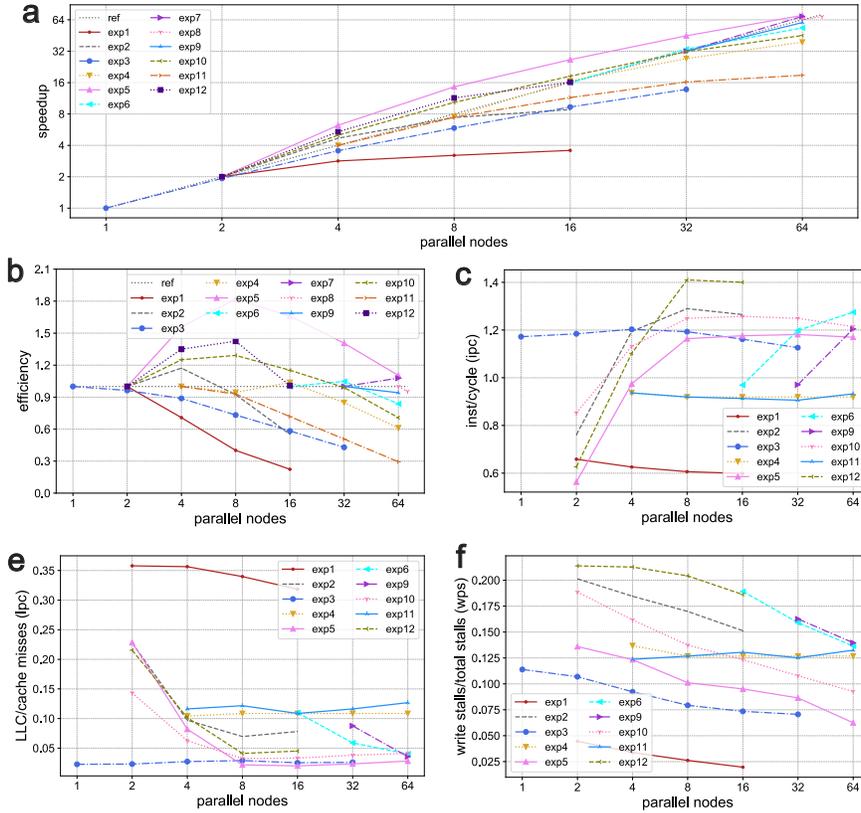


Figure 4: **Performance Metrics.** Performance metrics including (a) parallel speedup, (b) strong-scale efficiency, (c) instructions per cycle (*ipc*), (d) last level cache miss per total cache misses (*lpc*), and (e) write stalls per total stalls (*wps*) respectively are shown with increasing parallel nodes for all performance evaluation experiments (labeled as tuples: exp_n in section: Performance Evaluation). The black dotted lines (*ref*) show the ideal speedup and efficiency in (a) and (b) respectively.

producer-consumer pipeline halt time (see Superstep 3 in Methods) when searching three datasets of increasing size. The *wait time* is the time when any of the pipeline sub-tasks wait for a batch of data from its predecessor. The results (Figure 5e) show that our task-scheduling algorithm actively performs counter measures (reallocates threads) as soon as a pipeline-stall is detected due to speed mismatches between parallel sub-tasks keeping the total cost to $\leq 5\%$ in most experiments (Figure 5d).

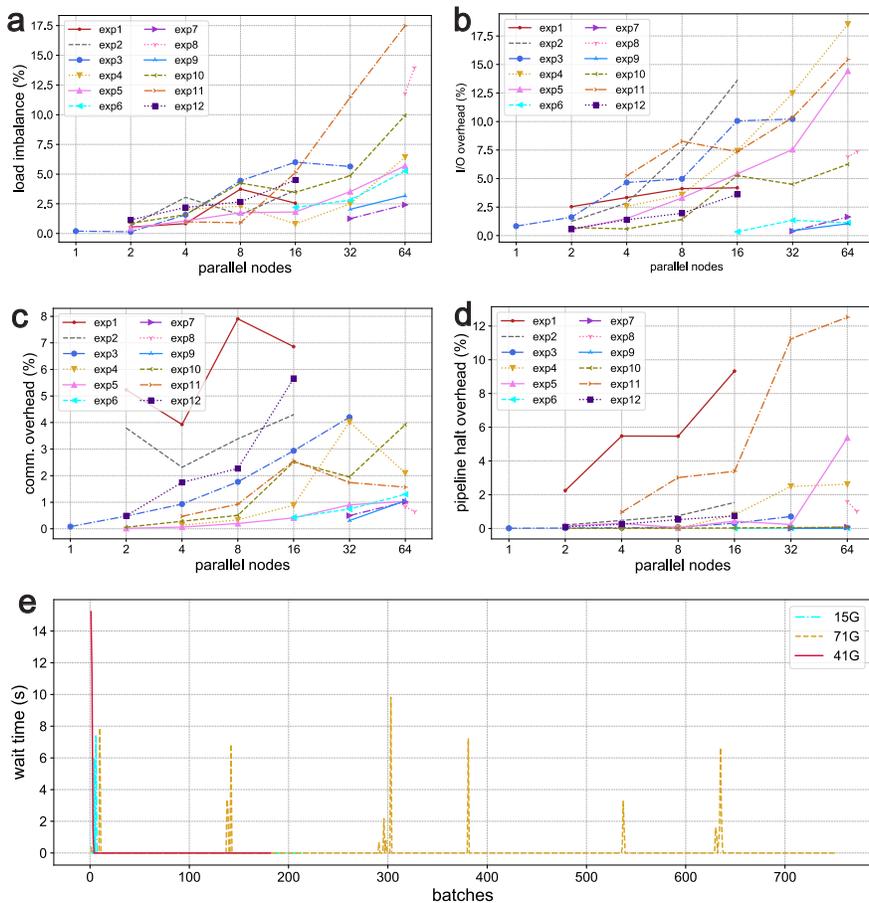


Figure 5: **Overhead Analysis.** Overhead costs including (a) load imbalance, (b) I/O, (c) communication, and (d) pipeline halt time are shown with increasing parallel nodes for all performance evaluation experiments (labeled as tuples: exp_n in section: Performance Evaluation). (e) The time series shows the per-batch sub-task pipeline halt time (scheduling performance) in Superstep 3 when searching datasets of sizes 15GB, 41GB, and 71GB in open-search using 64 nodes. The *wait time* shows the time the pipeline sub-tasks in Superstep 3 waited for corresponding data batches.

3 Discussion

Recent trends in high-performance computing (HPC) have shifted towards heterogeneous architectures [42] as several top-500 supercomputers combine CPUs with Graphical Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs) to deliver petascale (and in near future, exascale [43]) computing powers. However, the presented SPMD-BSP based HiCOPS design limits its application to only the homo-

geneous (CPU-only) parallel nodes in a supercomputer. This technological shift in HPC drives our future efforts that include a GPU-accelerated design for HiCOPS.

Peptide identification rates achieved by HiCOPS are limited by the underlying data processing, scoring and statistical modeling algorithms it executes. In our current design, we implement a basic shared-peak coupled hyperscoring algorithm [2] without making an explicit effort to improve these algorithms. Further, in some cases, searching against smaller databases (on single nodes) results in better performance (smaller workloads) and search quality (high-confidence separation of true positives from false positives). Although the proposed parallel design is algorithm-independent; i.e. underlying algorithms can be trivially ported and updated, we focus our future efforts on implementing (heterogeneous) HPC versions of several modern algorithms, and machine- and deep-learning models [44], [45], [9] within HiCOPS.

Finally, we believe that the computational tools are the enablers of new and more exciting science – science that one might not envision today because of the limitations of the infrastructure that is at our disposal. Therefore, we are confident that our current and future efforts will make a useful advance in enabling scientific investigations in this application domain.

4 Acknowledgments

This work used the NSF Extreme Science and Engineering Discovery Environment (XSEDE) Supercomputers through allocations: TG-CCR150017 (F.S.) and TG-ASC200004 (F.S.). This research was supported by the NIGMS of the National Institutes of Health (NIH) under award number: R01GM134384 (F.S.). The authors were further supported by the National Science Foundations (NSF) under the award number: NSF CAREER OAC-1925960 (F.S.). The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health and/or National Science Foundation.

5 Author Contributions

M.H. and F.S. designed the parallel computational framework. M.H. implemented the software. M.H. and F.S. designed and performed the experiments, performed calculations, analyzed the data and results, and wrote the manuscript.

6 Competing Interests

The authors declare no competing interests.

7 Methods

Notations and Symbols

For the rest of the paper, we will denote the number of peptide sequences in the database as (ζ), average number of post-translational modifications (PTMs) per peptide sequence as (m), the total theoretical database index size as ($\zeta(2^m) = D$), the number of parallel nodes/processes as (P), number of cores per parallel process as (c_{p_i}), size of experimental MS/MS dataset (i.e. number of experimental/query spectra) as (q), average length of query spectrum as (β), and the total dataset size as ($q\beta$). The runtime of executing the superstep (j) by parallel task (p_i) will be denoted as (T_{j,p_i}) and the generic overheads due to boilerplate code, OS delays, memory allocation etc. will be captured via (γ_{p_i}). Note that we shall refer the theoretical database as simply the database in the rest of the paper.

7.1 Runtime Cost Model

Since the HiCOPS parallel processes run in SPMD fashion, the cost analysis for any parallel process (with variable input size) is applicable for the entire system. Also, the runtime cost for a parallel process ($p_i \in P$) to execute superstep (j) can be modeled by only its local input size (i.e. database and dataset sizes) and available resources (i.e. number of cores, memory bandwidth). The parallel processes may execute the algorithmic work in a data parallel, task parallel or a hybrid task and data parallel model. As an example, the execution runtime (cost) for a parallel process p_i to execute superstep (j) which first generates D model-spectra using algorithm k_1 and then sorts them using algorithm k_2 in data parallel fashion (using all c_{p_i} cores) will be given as follows:

$$T_{j,p_i} = k_{j1}(D) + k_{j2}(D) + \gamma_{p_i} \quad (2)$$

Similarly, if the above steps k_z are performed in a hybrid task and data parallel fashion, the number of cores allocated to each (k_{jz}) must also be considered. For instance, in the above example, if the two algorithmic steps are executed in sub-task parallel fashion with $c_{p_i}/2$ cores each, the execution time will be given as:

$$T_{j,p_i} = \max(k_{j1}(D, c_{p_i}/2), k_{j2}(D, c_{p_i}/2)) + \gamma_{p_i} \quad (3)$$

For analysis purposes, if the time complexity of the algorithms used for step k_{jz} is known (say $O(\cdot)$), we will convert it into a linear function k'_{jz} with its input data size multiplied by its runtime complexity. This conversion will allow better quantification of serial and parallel runtime portions as seen in later sections. As an example, if it is known that the sorting algorithms used for k_{j2} have time complexity: $O(N \log N)$, the equation 2 can be modified to:

$$T_{j,p_i} = k_{j1}(D) + k'_{j2}(D \log D) + \gamma_{p_i} \quad (4)$$

Remarks: The formulated model will be used to analyze the runtime cost for each superstep, quantify the serial, parallel and overhead costs in the overall design, and optimize the overheads.

7.2 Superstep 1: Database Partitioning

In this superstep, the HiCOPS parallel processes construct a local database partition through the following three algorithmic data parallel steps (Figure 1): **1**) Generate and

extract a (balanced) local partition of the (peptides + PTM variants) database. **2)** Generate the theoretical spectra data. **3)** Index the local peptide and model-spectra to build the theoretical database index (suffix array and the fragment-ion index).

The database partitions are constructed using the LBE algorithm [46] (illustrated in Supplementary Figure 7). The LBE algorithm first clusters similar model-spectra in the database which are then scattered across parallel nodes cluster by cluster to achieve the balance [46] as also depicted in Supplementary Algorithm 1. In this work, we supplement the LBE algorithm with a new additional distance metric for clustering. We call this metric as the *Mod Distance* (Δm) which allows better separation of database spectral-pairs which cannot be separated by the normalized Edit Distance (Δe) metric introduced in the LBE algorithm (See Supplementary Section 5 for more information on Mod Distance). Consequently, the new distance metric allows better load balance between the database partitions as corroborated by our experimental results. To the best of our knowledge, LBE is the only existing technique for efficient theoretical database partitioning. *Mod Distance* (Δm) proposed in this paper is defined as follows:

Mod Distance: For a pair of model-spectra in the database (x, y), assuming the sum of unedited amino acid sequence lengths from both peptide sequence termini is (a), the Mod Distance (Δm) is given as follows (See Supplementary Section 5):

$$\Delta m(x, y) = 2 - \frac{a}{\max(\text{len}(x), \text{len}(y))}$$

Cost Analysis: The first step generates the entire database of size (D) and separates out a local partition (of roughly the size $D/P = D_{p_i}$) in runtime: $k_{11}(D)$. The second step generates the model-spectra from the partitioned database using the standard simulation model [12], [40] in runtime: $k_{12}(D_{p_i})$. The third step constructs a fragment-ion index similar to [2], [23], [21] in runtime: $O(N \log N)$. In our implementation, we employed the CFIR-Index [21] algorithm due to its smaller memory footprint resulting in runtime: $k'_{13}(D_{p_i} \log D_{p_i})$. Collective runtime for this superstep is given by Equation 5.

$$T_1 = \max_{p_i} (k_{11}(D) + k_{12}(D_{p_i}) + k'_{13}(D_{p_i} \log D_{p_i}) + \gamma_{p_i}) \quad (5)$$

Remarks: Equation 5 depicts that the serial execution time i.e. $k_{11}(D)$ bottlenecks the parallel efficiency.

7.3 Superstep 2: Experimental MS/MS Data Pre-processing

In this superstep, the HiCOPS parallel processes pre-process a partition of experimental MS/MS spectra data through the following three algorithmic data parallel steps (Figure 1): **1)** Read the dataset files, create a batch index and initialize internal structures. **2)** Pre-process (i.e. normalize, clear noise, reconstruct etc.) a partition of experimental MS/MS data. **3)** Write-back the pre-processed data.

The experimental spectra are split into batches such that a reasonable parallel granularity is achieved when these batches are searched against the database. By default, the maximum batch size is set to 10K spectra and the minimum number of batches per dataset is set to P . The batch information is indexed using a queue and a pointer stack to allow quick access to the pre-processed experimental data in the superstep 3.

Cost Analysis: The first step for reads the entire dataset (size: $q\beta$) and creates a batch index in runtime: $k_{21}(q\beta)$. The second step may pre-process a partition of the

dataset (of roughly the size: $q\beta/P = Q_{p_i}$) using a data pre-processing algorithm such as [47], [5], [44] etc. in runtime: $k_{22}(Q_{p_i})$. The third step may write the pre-processed data back to the file system in runtime: $k_{23}(Q_{p_i})$. Note that the second and third steps may altogether be skipped in subsequent runs when the input data are already pre-processed. Collective runtime for this superstep is given by Equation 6.

$$T_2 = \max_{p_i}(k_{21}(q\beta) + k_{22}(Q_{p_i}) + k_{23}(Q_{p_i}) + \gamma_{p_i}) \quad (6)$$

Remarks: Equation 6 depicts that the parallel efficiency of superstep 2 is highly limited by its dominant serial portion i.e. $k_{21}(q\beta)$. Moreover, this superstep is sensitive to the file system bandwidth since large volumes of data may be communicated to/from the shared file system.

7.4 Superstep 3: Database Peptide Search

This is the most important superstep in HiCOPS workflow and is responsible for 80-90% of the total algorithmic workload. In this superstep, the HiCOPS parallel processes search the pre-processed experimental spectra against their local database partitions through the following three algorithmic hybrid task and data parallel fashion steps (Figure 1, Supplementary Figure 4): **1**) Load the pre-processed experimental MS/MS data batches into memory. **2**) Search the loaded spectra batches against the (local) database partition and produce intermediate results. **3**) Serialize and write the intermediate results to the shared file system assigning them unique tags.

Three parallel sub-tasks are created, namely R , I and K , that work in a producer-consumer pipeline to execute the algorithmic work of this superstep (Figure 1c). The data flow between the sub-tasks is handled through queues to create a buffer between the producers and consumers. The first sub-task (R) loads batches of the pre-processed experimental spectra data and puts them in queue (q_f) as depicted in Supplementary Algorithm 2. The sub-task R may also perform *minimal* computations on the experimental spectra before putting them in queue. e.g. peak selection and/or intensity normalization. The parallel cores assigned to sub-task R are given by: $|r|$. The second sub-task (I) extracts batches from (q_f), performs the database peptide search (currently: shared-peak counting coupled hyperscore computation) against its local database partition and puts the produced intermediate (local) results in queue (q_k) as depicted in Supplementary Algorithm 3. The parallel cores assigned to sub-task I are given by: $|i|$. The sub-task I also recycles the memory buffers back to sub-task R using the queue (q_r). The third sub-task (K) serializes and writes the intermediate results to a shared memory using $|k|$ cores. Given an experimental spectrum (φ), a database peptide (χ), the number of shared b-ions between them (n_b) with intensities ($i_{b,j}$), and the number of shared y-ions between them (n_y) with intensities ($i_{y,k}$), the hyperscore between them is given as:

$$\text{hyperscore}(\varphi, \chi) = \log(n_b!) + \log(n_y!) + \log\left(\sum_{j=1}^{n_b} i_{b,j}\right) + \log\left(\sum_{k=1}^{n_y} i_{y,k}\right)$$

Cost Analysis: The sub-task (R) reads the experimental data batches in runtime: $k_{30}(q\beta)$. The sub-task (I) iteratively filters the database and computes spectral comparisons against the database (scoring step). Most commonly, the database peptide search algorithms use two or three database filtration steps, most commonly, peptide precursor mass tolerance [3], [29], shared fragment-ions [2], [23] and sequence tags [10] [9]. In current implementation, we use the first two filtration methods which execute in

runtime: $k_{31}(qD_{p_i})$ and $k_{32}(q\beta\alpha_{p_i})$ respectively. Here, the α_{p_i} represents the average filtered database size filtered from the first step. The currently implemented scoring mechanism computes hyperscores [13] in runtime: $k_{33}(q\beta\sigma_{p_i}) + k_{34}(q\mu_{p_i})$. Here, the σ_{p_i} and μ_{p_i} represent the average number of filtered shared-ions and model-spectra per experimental spectrum. Note that the scoring algorithm in this superstep is portable as the parallel design does not depend on it. Finally, the sub-task K writes the intermediate results to the shared file system in runtime: $k_{35}(q)$.

Overhead Costs: Overhead factors stemming from load imbalance, producer-consumer pipeline halt, file system bandwidth congestion affect the performance of this superstep. Therefore, we capture them using an additional runtime cost: $V_{p_i}(q, D_{p_i}, P)$. Several optimizations including buffering, task scheduling, load balancing and data sampling (discussed in later sections) were implemented to alleviate the overhead costs. Collective runtime for this superstep is given by Equation 10.

The runtime of sub-task R , i.e. $t_{p_i}(r, | r |)$, is given as:

$$t_{p_i}(r, | r |) = k_{30}(q\beta, | r |) \quad (7)$$

The runtime of sub-task I , i.e. $t_{p_i}(i, | i |)$, is given as:

$$t_{p_i}(i, | i |) = k_{31}(qD_{p_i}, | i |) + k_{32}(q\beta\alpha_{p_i}, | i |) + k_{33}(q\beta\sigma_{p_i}) + k_{34}(q\mu_{p_i}, | i |)$$

Or:

$$t_{p_i}(i, | i |) = k'_{31}(q \log(D_{p_i}), | i |) + k'_{32}(q\beta \log(\alpha_{p_i}), | i |) + k_{33}(q\beta\sigma_{p_i}, | i |) + k_{34}(q\mu_{p_i}, | i |) \quad (8)$$

The runtime of sub-task K , i.e. $t_{p_i}(k, | k |)$, is given as:

$$t_{p_i}(k, | k |) = k_{35}(q, | k |) \quad (9)$$

Combining equations 7, 8 and 9 we have:

$$T_3 = \max_{p_i}(\max(t_{p_i}(r, | r |), t_{p_i}(i, | i |), t_{p_i}(k, | k |)) + V_{p_i}(q, D_{p_i}, P) + \gamma_{p_i}) \quad (10)$$

Remarks: Equations 7, 8, 9 and 10 depict that the parallel runtime portion of this superstep grows quadratically superseding the serial portion if the experimental load is sufficient.

7.5 Superstep 4: Result Assembly

In this superstep, the HiCOPS parallel processes assemble the intermediate results from the last superstep into complete results through the following hybrid task and data parallel algorithmic steps (Figure 1d): **1)** Read a set of intermediate result batches, assemble them into complete results, and send the assembled results to their parent processes. **2)** Receive complete results from other parallel processes and synchronize communication. **3)** Write the complete results to the file system.

Two parallel sub-tasks are created to execute the algorithmic steps in this superstep. The first sub-task reads sets of intermediate results from the shared file system (or shared memory) (satisfying: $tag \bmod P = p_i; p_i \in \text{MPI ranks}$), de-serializes them and assembles the complete results. The expectation scores are then computed and communicated to their origin processes. For example, the process with MPI rank 4

will process the all intermediate result batches with tag $0x8_i$ where $i = 0, 1, \dots, P - 1$. The assembly process is done through signal addition and shift operations (Figure 1d). The expected values (expectscores (es)) are computed by first smoothing the assembled data through Savitzky-Golay filter and then applying the null test through either the Linear-Tail Fit [48] or log-Weibull (Gumbel) Fit method (Figure 1d). The computed es along with additional information (total: 16 bytes) are queued to be sent to the HiCOPS process that recorded the most significant database hit (origin). The final results are stored in a map data structure for collective communication at the end of all batches. All available cores (c_{p_i}) are assigned to this sub-task. Supplementary Algorithm 4 depicts the algorithmic work performed by this sub-task.

The second sub-task runs waits for $P - 1$ packets of results from other HiCOPS processes. This task runs asynchronously using an over-subscribed thread and only activates when incoming data is detected. Finally, once the two sub-tasks complete (join), the complete results are written to the file system in data parallel fashion using all available threads.

Cost Analysis: The first sub-task reads the intermediate results, performs regression and sends computed results to other processes in runtime: $k_{41}(Q_{p_i}, c_{p_i}) + k_{42}(Q_{p_i}, c_{p_i}) + k_{43}(P, 1)$ time. The second sub-task receives complete results from other tasks in runtime: $k_{44}(P, 1)$. Finally, the complete results are written in runtime: $k_{45}(Q_{p_i})$. Collectively, the runtime for this superstep is given by equation 11.

$$T_4 = \max_{p_i}(\max(k_{41}(Q_{p_i}, c_{p_i}) + k_{42}(Q_{p_i}, c_{p_i}) + k_{43}(P, 1), k_{44}(P, 1)) + k_{45}(Q_{p_i}) + \gamma_{p_i}) \quad (11)$$

To simplify equation 11, we can re-write it as a sum of computation costs plus the communication overheads ($k_{com}(P, 1)$) as:

$$T_4 = \max_{p_i}(k_{41}(Q_{p_i}, c_{p_i}) + k_{42}(Q_{p_i}, c_{p_i}) + k_{com}(P, 1) + k_{45}(Q_{p_i}) + \gamma_{p_i}) \quad (12)$$

Assuming that the network latency is denoted as (ω), bandwidth is denoted as (π) and ($16Q_{p_i}$) is the average data packet size in bytes, the inter-process communication overhead cost ($k_{com}(P, 1)$) in seconds is estimated to be:

$$k_{com}(P, 1) \approx 2(P - 1)(\omega + 16Q_{p_i}/\pi)$$

Remarks: As the communication per process are limited to only one data exchange between any pair of processes, the overall runtime given by equation 12 is highly scalable. The effective communication cost depends on the amount of overlap with computations and the network parameters at the time of experiment.

7.6 Performance Analysis

To quantify the parallel performance, we decompose the total HiCOPS time T_H (Eq. 1) into three runtime components. i.e. parallel runtime (T_p), serial runtime (T_s) and overheads runtime (T_o) given as:

$$T_H = \sum_{j=1}^4 \max_{p_i}(T_{j,p_i}) = T_o + T_s + T_p \quad (13)$$

Using equations 1, 5, 6, 10, and 12, we separate the three runtime components as:

$$T_o = V_{p_i}(q, D_{p_i}, P) + \gamma_{p_i} \quad (14)$$

$$T_s = k_{11}(D) + k_{21}(q\beta) + k_{com}(P, 1) \quad (15)$$

and:

$$T_p = k_{12}(D_{p_i}) + k'_{13}(D_{p_i} \log D_{p_i}) + k_{22}(Q_{p_i}) + k_{23}(Q_{p_i}) + \max(t_{p_i}(t, | r |), t_{p_i}(i, | i |), t_{p_i}(k, | k |)) + k_{41}(Q_{p_i}, c_{p_i}) + k_{42}(Q_{p_i}, c_{p_i}) + k_{45}(Q_{p_i}) \quad (16)$$

T_s is the minimum serial time required for HiCOPS execution and cannot be further reduced. Therefore, we will focus on optimizing the remaining runtime: $T_F = T_p + T_o$. Using equations 14 and 16, we have:

$$T_F = k_{12}(D_{p_i}) + k'_{13}(D_{p_i} \log D_{p_i}) + k_{22}(Q_{p_i}) + k_{23}(Q_{p_i}) + \max(t_{p_i}(t, | r |), t_{p_i}(i, | i |), t_{p_i}(k, | k |)) + k_{41}(Q_{p_i}, c_{p_i}) + k_{42}(Q_{p_i}, c_{p_i}) + k_{45}(Q_{p_i}) + T_o \quad (17)$$

Since the HiCOPS parallel processes divide the database and experimental dataset roughly fairly in supersteps 1 and 2, the first four and the sixth term in T_p are already almost optimized, so we can prune them from T_F :

$$T_F = \max(t_{p_i}(t, | r |), t_{p_i}(i, | i |), t_{p_i}(k, | k |)) + k_{41}(Q_{p_i}, c_{p_i}) + k_{42}(Q_{p_i}, c_{p_i}) + k_{45}(Q_{p_i}) + T_o \quad (18)$$

Recall that the superstep 4 runtime is optimized for maximum parallelism (and least inter-process communication) and that the superstep 3 performs the largest fraction of overall algorithmic workload. Thus, we can also remove the superstep 4 terms from T_F to simplify analysis:

$$T_F = \max(t_{p_i}(t, | r |), t_{p_i}(i, | i |), t_{p_i}(k, | k |)) + T_o$$

Further, as that the superstep 3 is executed using the producer-consumer pipeline (Figure 1c) where the sub-task R must produce all data before it can be consumed by I meaning its runtime must also be smaller than $t_{p_i}(i, | i |)$ and $t_{p_i}(k, | k |)$ allowing a safe removal from the above equation yielding:

$$T_F = \max(t_{p_i}(i, | i |), t_{p_i}(k, | k |)) + T_o$$

In above equation, we can rewrite the $\max(\cdot)$ term as the time to complete sub-task I : ($t_{p_i}(i, | i |)$) plus the extra time to complete sub-task K (the last consumer): $t_x(k)$. Therefore, using equation 9 we have:

$$T_F = k'_{31}(q \log(D_{p_i}), | i |) + k'_{32}(q\beta \log(\alpha_{p_i}), | i |) + k_{33}(q\beta\sigma_{p_i}, | i |) + k_{34}(q\mu_{p_i}, | i |) + t_x(k) + T_o \quad (19)$$

We can prune the first two terms in the equation 19 as well since their runtime contribution: $O(\log N)$ will be relatively very small. Finally, using equation 14 in 19, we have:

$$T_F = k_{33}(q\beta\sigma_{p_i}, | i |) + k_{34}(q\mu_{p_i}, | i |) + t_x(k) + V_{p_i}(q, D_{p_i}, P) + \gamma_{p_i} \quad (20)$$

7.7 Optimizations

The following sections discuss the optimization techniques employed to alleviate the overhead costs in Equation 20.

7.7.1 Buffering

Four queues, the forward queue (q_f), recycle queue (q_r) and result queues (q_k, q'_k) are initialized and routed between the producer-consumer sub-tasks in the superstep 3 (Figure 1c) as: $R \rightarrow I, R \leftarrow I, I \rightarrow K$ and $I \leftarrow K$ respectively. The q_r is initialized with (default: 20) empty buffers for the sub-task R to fill the pre-processed experimental data batches and push in q_f . The sub-task I removes a buffer from q_f , consumes it (searches it) and pushes back to q_r for re-use. The results are pushed to q_k which are consumed by sub-task K and pushed back to q'_k for re-use. Three regions are defined for the queue q_f based on the number of data buffers it contains at any time. i.e. $w_1 : (len(q_f) < 5)$, $w_2 : (5 \leq len(q_f) < 15)$ and $w_3 : (len(q_f) \geq 15)$. These regions (w_l) are used by the task-scheduling algorithm discussed in the following section.

7.7.2 Task Scheduling

The task scheduling algorithm is used to maintain a synergy between the producer-consumer (sub-task) pipeline in the superstep 3. The algorithm initializes a thread pool of $c_{p_i} + 2$ threads where c_{p_i} is the number of available cores. In the first iteration, 2 threads are assigned to the sub-tasks R and K while the remaining $c_{p_i} - 2$ threads are assigned to sub-task I . Then, in each iteration, the q_f region: w_l , and the $q_f.pop()$ time for I , given by: t_{wait} , are monitored. A time series is built to forecast the next t_{wait} (i.e. t_{fct}) using double exponential smoothing [49]. The t_{wait} is also accumulated into t_{cum} . Two thresholds are defined: minimum wait (t_{min}) and maximum cumulative wait (t_{max}). Using all this information, a thread is removed from sub-task I and added to R if the following conditions are satisfied:

$$c_{I \rightarrow R} = (t_{wait} \geq t_{min} \wedge (t_{cum} + t_{fct}) > t_{max}) \vee (w_l = w_1 \wedge |r| = 0)$$

The t_{cum} is set to 0 every time a thread is added to R . Similarly, a thread is removed from sub-task R and added to I if the following conditions are satisfied. All threads are removed from R if the queue q_f becomes full or there is no more experimental MS/MS data left to be loaded.

$$c_{R \rightarrow I} = (w_l = w_3 \wedge |r| > 1) \vee q_f.full()$$

The sub-task K uses its 2 over-subscribed threads to perform the overlapped I/O operations concurrently (Figure 1c).

7.7.3 Load Balancing

The algorithmic workload in equation 20 is given by: $k_{33}(q\beta\sigma_{p_i}, |i|) + k_{34}(q\mu_{p_i}, |i|)$. Here, the terms $q\beta$ and q are constants (experimental data size) whereas the terms σ_{p_i} and μ_{p_i} are variable. The variable terms represent the filtered database size for a parallel HiCOPS process (p_i) and thus, must be balanced across processes. We do this statically by constructing balanced database partitions (hence a balanced workload) using the LBE algorithm supplemented with our new *Mod Distance* metric in Superstep 1 (Methods, Figure 1a, Supplementary Figure 6). The correctness of the LBE algorithm for load balancing is proven in Supplementary Section 6. In future, we plan to devise and develop dynamic load balancing techniques for better results.

7.7.4 Sampling

Sampling is used to reduce the inter-process communication required in result assembly (superstep 4) without compromising on the assembly accuracy. For each experimental spectrum, the HiCOPS processes (p_i) produce a local result consisting: number of local hits, hyperscore for the top hits etc. (12 bytes), and the local null distribution histogram of hyperscores (2048 bytes). Communicating this, the size of each data packet (1 per batch) will be: $\sim 20\text{MB}$, which can result in serious overheads. It has been shown that the null distribution hyperscore (and several other scoring algorithms) in database peptide search follow a log-Weibull or Gumbel curve [41]. This means that most of the data are localized around the mean. We exploit this information to reduce the communication footprint as follows: We first locate the mean of the local null distribution and sample most intense non-zero data points around it. If the total number of non-zero samples exceed $s(=120$ default), we prioritize the samples towards the head of the distribution as we can reconstruct the tail fairly accurately through curve fitting. The sampled data are further encoded into `unsigned short` instead of `int` to fit inside a buffer of 256 bytes resulting in a 1.5MB data packet size which is instantly written/read from the shared file system reducing the overhead costs including $t_x(k)$ (see Equation 20). Supplementary Figure 7 illustrates an example of sampling.

7.8 Detailed Experimental Setup

The two databases (i.e. D_1 and D_2) were digested in-silico using Trypsin as enzyme (fully tryptic) with 2 allowed missed cleavages, peptide lengths between 6 and 46 and peptide masses between 500 and 5000Da. The pseudo-spectra were simulated by generating b- and y-ions up to +3 charge with zero isotope error and no decoys. Cysteine carbamidomethylation was set as fixed modification for all experiments whereas the variable modifications were chosen from the combinations of Methionine oxidation, Arginine and Glutamine deamidation, Serine, Threonine and Tyrosine phosphorylation, Cysteine and Lysine gly-gly adducts, and Tyrosine Biotin-tyramide across experiments. The maximum number of allowed modified residues (amino acid letters) per peptide was set to 5. The number and type of PTMs used in database expansion, and the search settings including peptide precursor mass tolerance (δM) were varied across experiments to cover both the open-: $\delta M \sim \pm 500\text{Da}$ and closed-search: $\delta M \leq \pm 10\text{Da}$ scenarios. The closed-search criterion was set to a few Daltons ($\leq 1\text{Da}$ in correctness analysis and $\leq 10\text{Da}$ in performance evaluation) instead of 10-20ppms to cover the differences in calculated peptide precursor masses due to monoisotopic or average masses and isotopic masses across search tools. The four experimental MS/MS datasets were converted to MS2 format before use. The experimental MS/MS spectra pre-processing settings for all tools were set to minimal so that all tools execute a nearly identical algorithmic work (fairness). Some of these settings are listed as follows: allowed precursor masses: 500 to 5000Da, precursor charges: +1 to +4, min matched peaks for PSM candidacy: 4, min database hits for statistical scoring: 4, de-noising: only top 100 peaks picked (by intensity), peak transformations: none, mass calibration: no, precursor peak removal: no, partial spectrum re-construction: no, clip n-term M: no.

8 Code Availability

The HiCOPS software has been implemented using object-oriented C++17, MPI, OpenMP, Python, Bash and CMake. Instrumentation interface is implemented via Timemory [42] for performance analysis. Command-line tools for MPI task mapping (Supplementary Section 7), database processing, file format conversion and result post-processing are also distributed with the software. HiCOPS is under active development and all documentation updates, source code releases etc. will be updated on the same web page. The source code is available open-source at <https://doi.org/10.5281/zenodo.5094072> [50] and <https://github.com/hicops/hicops>. Please refer to the software web page: <https://hicops.github.io> for detailed documentation, licensing and future software updates.

9 Data Availability

All datasets used in this study are publicly available from Pride Archive and can be accessed via <https://www.ebi.ac.uk/pride/archive/projects/<AccessionNum>> where *AccessionNum* is the accession number for each dataset mentioned in the text. For example, to access the dataset S_1 : PXD009072, use the link: <https://www.ebi.ac.uk/pride/archive/projects/PXD009072>. Homo sapiens protein sequence database can be downloaded from UniProtKB using the link: <https://www.uniprot.org/proteomes/UP000005640>. The UniProt SwissProt (reviewed) database can be downloading using the link: <https://www.uniprot.org/uniprot/?query=reviewed:yes>. Source data for Figures 2, 3, 4, 5 are also available with this manuscript as well as on [39].

References

- [1] 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.
- [2] Andy T Kong, Felipe V Leprevost, Dmitry M Avtonomov, Dattatreya Melchacheruvu, and Alexey I Nesvizhskii. Msfragger: ultrafast and comprehensive peptide identification in mass spectrometry-based proteomics. *Nature methods*, 14(5):513, 2017.
- [3] Sean McIlwain, Kaipo Tamura, Attila Kertesz-Farkas, Charles E Grant, Benjamin Diamant, 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.
- [4] Zuo-F ei Yuan, Chao Liu, Hai-Peng Wang, Rui-Xiang Sun, Yan Fu, Jing-Fen Zhang, Le-Heng Wang, Hao Chi, You Li, Li-Yun Xiu, et al. pparse: A method for accurate determination of monoisotopic peaks in high-resolution mass spectra. *Proteomics*, 12(2):226–235, 2012.
- [5] 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.

- [6] Sven Degroeve and Lennart Martens. Ms2pip: a tool for ms/ms peak intensity prediction. *Bioinformatics*, 29(24):3199–3203, 2013.
- [7] Xie-Xuan Zhou, Wen-Feng Zeng, Hao Chi, Chunjie Luo, Chao Liu, Jianfeng Zhan, Si-Min He, and Zhifei Zhang. pdeep: predicting ms/ms spectra of peptides with deep learning. *Analytical chemistry*, 89(23):12690–12697, 2017.
- [8] 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.
- [9] 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.
- [10] Hao Chi, Chao Liu, Hao Yang, Wen-Feng Zeng, Long Wu, Wen-Jing Zhou, Xiun-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.
- [11] Marshall Bern, Yuhan Cai, and David Goldberg. Lookup peaks: a hybrid of de novo sequencing and database search for protein identification by tandem mass spectrometry. *Analytical chemistry*, 79(4):1393–1400, 2007.
- [12] 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.
- [13] 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.
- [14] Benjamin J Diament and William Stafford Noble. Faster sequest searching for peptide identification from tandem mass spectra. *Journal of proteome research*, 10(9):3871–3879, 2011.
- [15] Jimmy K Eng, Bernd Fischer, Jonas Grossmann, and Michael J MacCoss. A fast sequest cross correlation algorithm. *Journal of proteome research*, 7(10):4598–4602, 2008.
- [16] Christopher Y Park, Aaron A Klammer, Lukas Kall, Michael J MacCoss, and William S Noble. Rapid and accurate peptide identification from tandem mass spectra. *Journal of proteome research*, 7(7):3022–3027, 2008.
- [17] 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.
- [18] 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.

- [19] 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.
- [20] Jimmy K Eng, Brian C Searle, Karl R Clauser, and David L Tabb. A face in the crowd: recognizing peptides through database search. *Molecular & Cellular Proteomics*, pages mcp–R111, 2011.
- [21] Muhammad Haseeb and Fahad Saeed. Efficient shared peak counting in database peptide search using compact data structure for fragment-ion index. In *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 275–278. IEEE, 2019.
- [22] 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.
- [23] 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.
- [24] Vivien Marx. Biology: The big challenges of big data, 2013.
- [25] 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.
- [26] 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.
- [27] 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.
- [28] 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.
- [29] 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.
- [30] 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.

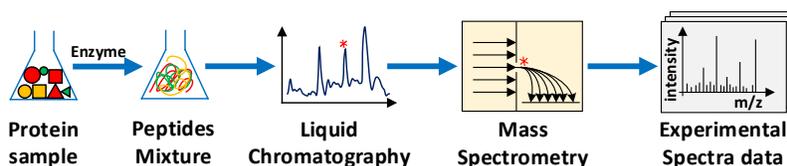
- [31] Amol Prakash, Shadab Ahmad, Swetaketu Majumder, Conor Jenkins, and Ben Orsburn. Bolt: A new age peptide search engine for comprehensive ms/ms sequencing through vast protein databases in minutes. *Journal of The American Society for Mass Spectrometry*, 30(11):2408–2418, 2019.
- [32] Patricia Kaiser, 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.
- [33] 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.
- [34] John R Yates III. Proteomics of communities: metaproteomics, 2019.
- [35] 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.
- [36] Leslie G Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.
- [37] Alexander Tiskin. *BSP (Bulk Synchronous Parallelism)*, pages 192–199. Springer US, Boston, MA, 2011.
- [38] John Towns, Timothy Cockerill, Maytal Dahan, Ian Foster, Kelly Gaiher, 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.
- [39] Muhammad Haseeb and Fahad Saeed. Source Data: High Performance Computing Framework for Tera- Scale Database Search of Mass Spectrometry Data. <https://doi.org/10.5281/zenodo.5076575>, July 2021.
- [40] 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.
- [41] Robertson Craig and Ronald C Beavis. Tandem: matching proteins with tandem mass spectra. *Bioinformatics*, 20(9):1466–1467, 2004.
- [42] Jonathan R Madsen, Muaaz G Awan, Hugo Brunie, Jack Deslippe, Rahul Gayatri, Leonid Olikier, 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.
- [43] Rick Stevens, Jini Ramprakash, Paul Messina, Michael Papka, and Katherine Riley. Aurora: Argonne’s next-generation exascale supercomputer. Technical report, ANL (Argonne National Laboratory (ANL), Argonne, IL (United States)), 2019.
- [44] Kaiyuan Liu, Sujun Li, Lei Wang, Yuzhen Ye, and Haixu Tang. Full-spectrum prediction of peptides tandem mass spectra using deep neural network. *Analytical chemistry*, 92(6):4275–4283, 2020.

- [45] Yang-Ming Lin, Ching-Tai Chen, and Jia-Ming Chang. Ms2cnn: predicting ms/ms spectrum based on protein sequence using deep convolutional neural networks. *BMC genomics*, 20(9):1–10, 2019.
- [46] 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.
- [47] 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.
- [48] 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.
- [49] 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.
- [50] Muhammad Haseeb and Fahad Saeed. <https://doi.org/10.5281/zenodo.5094072> hicops software code v1.0.0 - 1st public release, July 2021.
- [51] Sangtae Kim and Pavel A Pevzner. Ms-gf+ makes progress towards a universal database search tool for proteomics. *Nature communications*, 5:5277, 2014.
- [52] 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.

1 Supplementary Figures

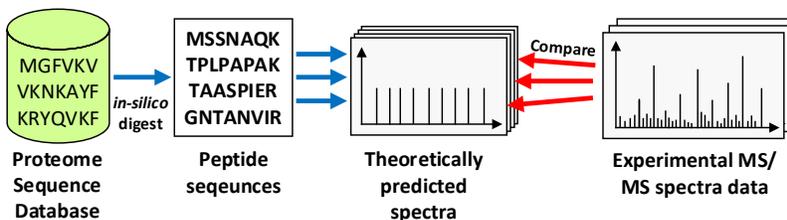
1.1 Supplementary Figure 1

Shotgun Mass-Spectrometry Proteomics. Proteins in a biological sample are proteolyzed into peptides by digesting them through an enzyme, typically Trypsin. The resultant peptide mixture is fed to an automated liquid chromatography (LC) coupled two-staged MS/MS pipeline (LC-MS/MS) which yields an experimental MS/MS spectrum per peptide. Thousands of experimental spectra (several GBs data) can be generated in a span of few hours.



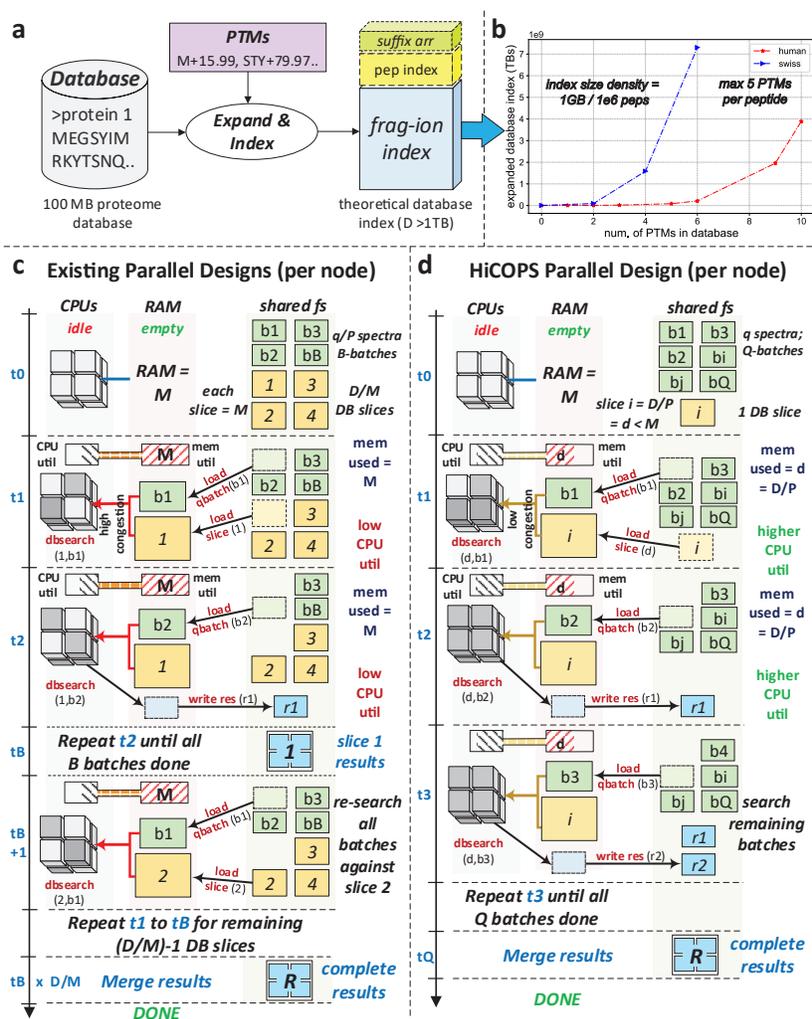
1.2 Supplementary Figure 2

Generic Database Peptide Search Workflow. Experimental MS/MS spectra are compared against a database of theoretically predicted spectra. These theoretical spectra are generated by first in-silico digesting a protein sequence database followed by spectral predicted using a probabilistic (or machine-learned) model. Post-translational modifications (PTMs) are also added in the digestion process to also generate and include the (modified) peptide variants in the theoretical spectra database. Adding PTMs to the database results in a combinatorial expansion to: $O(D_0 \times 2^m)$; where D_0 is the database size before PTMs and m is the average number of PTMs added per peptide.



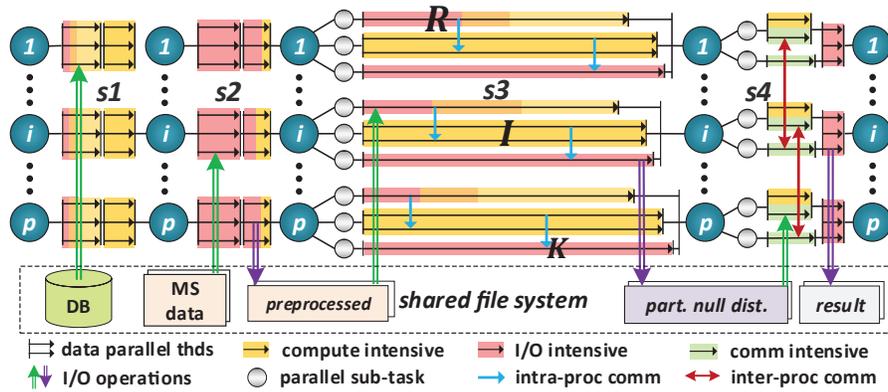
1.3 Supplementary Figure 3

Comparison of the Parallel Methods (a) The theoretical database (peptide and fragment-ion index) expands combinatorially when PTMs are added. (b) Homo sapiens and Swiss databases expand to >1B peptides and variants (indexed size: > 1TB) as a few commonly used PTMs are added. (c) Each parallel node in existing designs search a portion of the dataset against the entire replicated database. This results in exhibit poor performance due to memory contention and other overheads (low CPU utilization) when the theoretical database size (D) \gg RAM (M). (d) Each node in the proposed design searches the entire dataset against a small portion of database in a pipelined stream coupled with scheduling, sampling and other optimizations for efficient HPC resource utilization.



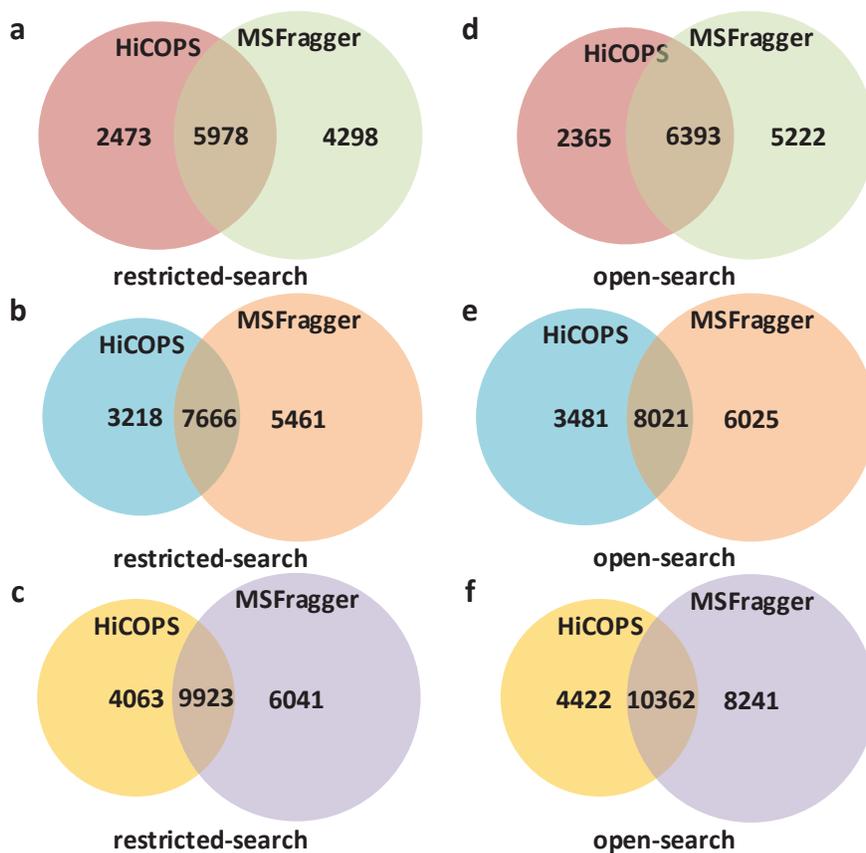
1.4 Supplementary Figure 4

Workload Profile. Supersteps 1 and 2 are designed as data parallel. Supersteps 3 and 4 are designed as hybrid task and data parallel. The workload executed by the four respective supersteps are compute intensive, I/O intensive, mixed (compute and I/O), and mixed (compute and comm.). In the last two supersteps, the compute workload may supersede the communication and/or I/O, given that the associated overhead costs are overlapped or minimized.



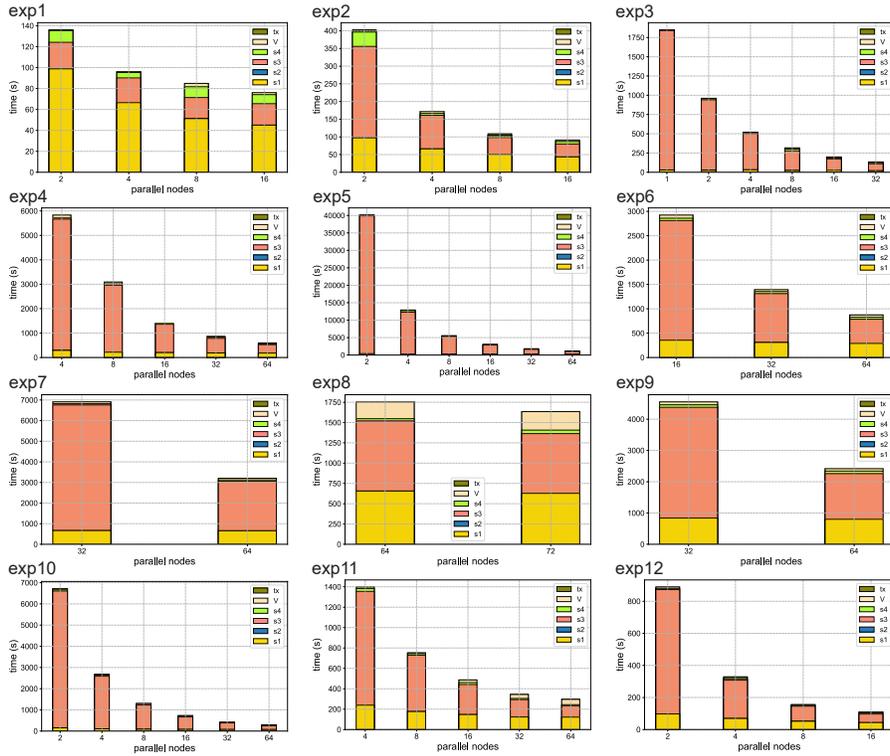
1.5 Supplementary Figure 5

Comparison of Filtered Peptide Identifications. Comparison of the peptide to spectrum matches (PSMs) obtained from HiCOPS and MSFragger after false discovery rate analysis (1% q-value) is shown for the *six* experiments discussed in Figures 2c to 2h (see Main text). (a,b,c) show more than 50% overlap between the two tools for closed-search experiments. (d, e, f) show that MSFragger outperforms the HiCOPS's underlying algorithm in open-search scenario. Nonetheless, HiCOPS is designed as algorithm-oblivious and the search quality can be improved using different sets of algorithms.



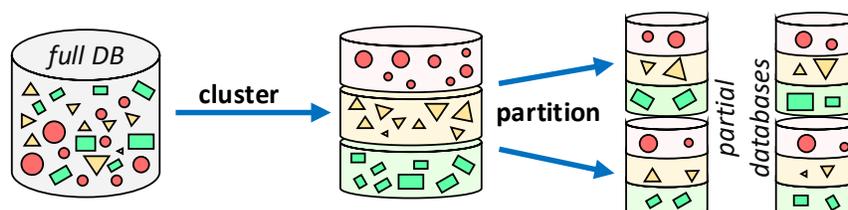
1.6 Supplementary Figure 6

Extended HiCOPS Performance Results Decomposition of HiCOPS’s total runtime (tot) into individual supersteps (s_j), producer-consumer pipeline overhead time (t_x), and remaining overheads (V) (see Equation 20 in Methods) is shown for the 12 experiments described in Figure 4 (see Main text). Note that the minimum number of parallel nodes (P_{min}) required for each experiment is $= D/M$ where M is the available main memory per node. The sub-figures show near ideal speedups as the workload (database, dataset and search filter) size increases.



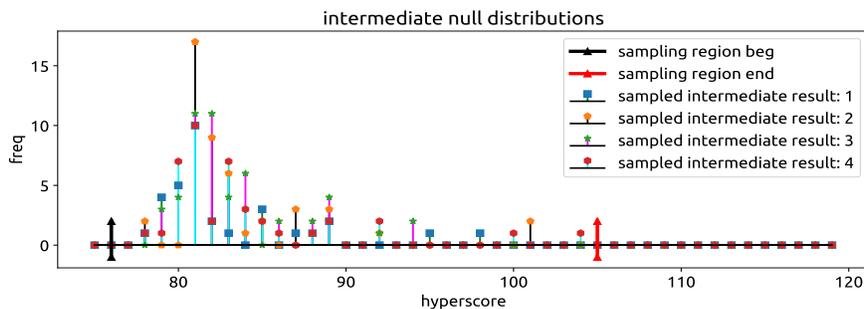
1.7 Supplementary Figure 7

Summary of the LBE Algorithm. The improved LBE method used in the superstep 1 clusters the theoretical database entries (shown as shapes) using two distance metrics: Edit Distance (Δe) and Mod Distance (Δm) (explained in Supplementary Section 5). The obtained database clusters (shown as highlighted pieces) are then finely and evenly scattered across database partitions at parallel HiCOPS processes in either round robin or random fashion.



1.8 Supplementary Figure 8

Example of the Sampling Technique. An example of the sampling technique used to minimize the inter-process communication load in superstep 4 is shown. The mean is computed roughly by averaging the locations of three most intense samples in the distribution ($\mu \approx 81$ in the figure). Then, a maximum of $s = 120$ most intense data points around the mean are sampled and the others are discarded. The discarding prefers pruning samples from the tail first the tail is usually easily recovered by fitting a log-Weibull curve into the data.



2 Supplementary Sections

2.1 Supplementary Section 1

Related Work. The distributed memory parallel database peptide search algorithms emerged with the Parallel Tandem [25], which is a variant of the X!Tandem [41] tool. Parallel Tandem achieves parallelism by spawning multiple instances of the original X!Tandem using MPI or PVM, where each instance processes a chunk of experimental dataset files. X!!Tandem [26] is another variant of the X!Tandem which implements an internal (but similar) parallel technique for computational and synchronization steps. The experimental dataset files in the case of X!!Tandem are shuffled among the MPI processes to achieve better load balance. MR-Tandem [27] follows a strategy similar to X!!Tandem however by breaking computations into small Map and Reduce tasks (Map-Reduce model) exhibiting better parallel efficiency than the Parallel Tandem and X!!Tandem. MCtandem [28] and SW-Tandem [29] implement the same parallel design but offload the X!Tandem’s expensive Spectral Dot Product (SDP) computations over Intel Many Integrated Core (MIC) co-processor and Haswell AVX2 vector instructions respectively. Both algorithms also implement optimization techniques including double buffering, pre-fetching, overlapped communication and computations and a task-distributor for better performance. Bolt [31] implements a cloud-computing based parallel solution using a MSFragger-like [2] algorithm where each parallel instance constructs a full theoretical database and processes a chunk of experimental data. MS-PyCloud [30] runs parallel instances of MS-GF+ [51] in cloud-computing setting. Similarly, UltraQuant achieves parallelization by spawning instances of MaxQuant37 with a partition of experimental dataset.

2.2 Supplementary Section 2

Limitations in existing HPC methods. Most existing HPC database peptide search algorithms achieve parallelization through naive methods - spawn (P) instances of the original serial code on memory-distributed parallel nodes, each searching a partition of the experimental MS/MS data against a local replica of the entire database. Database replication inflates the space complexity of these HPC algorithms to $= O(N)$; where $= O(N)$ is the space complexity of their serial counterparts. Consequently, these methods, if applied to modern (high memory footprint) database search algorithms will suffer from memory bandwidth contention, memory swaps, page faults, I/O from the shared file-system, load imbalance and out-of-core processing yielding non-scalable performance. Moreover, adding PTMs to the database will aggravate memory and bandwidth exhaustion at an exponential rate (Supplementary Figure 3b).

To workaround this bottleneck, Bolt [31] limits its application to the use case where each database replica must fit within the (150GB) main memory available on system nodes. On the other hand, X!Tandem’s parallel variants [25], [26], [28], [29], MS-PyCloud, UltraQuant and others parallelize the classical database search algorithms with focus on boosting the scoring arithmetic operations. However, these algorithms require enormous amounts of on-the-fly floating point computations making them comparatively slower. For instance, the SW-Tandem running on 6 parallel nodes performs slower than single node MSFragger run and even beyond 6 nodes the speedups are not significant [29]. Consequently, the invention of sophisticated parallel methods capable of leveraging all available (HPC) hardware resources as well as the existing and future algorithmic advancements are critical in furthering research in this domain.

There have been some investigation efforts by Kulkarni et al.[52] towards parallel designs capable of optimizing the database search compute and memory load per node to $= O(N/P)$ via theoretical database (and corresponding index) partitioning instead of the other way around. These designs employ stream based workflows where each parallel process receives a batch of experimental data, executes a partial search, and passes on the results to the next process in the stream. However, this scheme still required significant amounts of on-the-fly computations and frequent data communication between parallel nodes leading to high compute times showing promising, albeit limited ($\sim 50\%$) parallel performance [52].

2.3 Supplementary Section 3

SW-Tandem. The SW-Tandem binaries were obtained from its GitHub repository: <https://github.com/Logic09/SW-Tandem> and were run on XSEDE Comet system with increasing number of parallel nodes using MPI but no speedups in runtime were observed. We ran the Linux utility `ldd` on both binaries attached at the SW-Tandem GitHub repository and found that they do not link to any parallel library (MPI, PVM, UPCXX etc.) - hence no speedup. We also ran the `diff` utility on both binaries and found that both of them were exactly the same. We contacted the corresponding author via Email and GitHub issues (<https://github.com/Logic09/SW-Tandem/issues>) but did not receive a response as of the acceptance date of this paper.

2.4 Supplementary Section 4

Existing parallel methods vs HiCOPS. *The HiCOPS achieves significant speed improvement over existing methods due to efficient resource utilization and overhead minimization.*

Proof. Let the experimental dataset size be (q) spectra, available main memory per node be (M) , theoretical database index size be $(D \gg M)$, number of parallel nodes be (P) , average number of database comparisons per experimental spectrum be (k) , spectrum I/O time be (t_{io}) , indexing time per database slice of size (d) be $(t_{i,d})$, and the time to perform one (database query + hyperscore) given database slice of size (d) be $(t_{s,d})$, then:

The algorithmic runtime per node using *existing* parallel techniques (w_{ex}) is the sum of I/O time for q/P spectra D/M times, indexing D/M database slices of size M , and searching the q/P spectra against those slices, given as:

$$w_{ex} = \frac{q}{P} \frac{D}{M} \times t_{io} + \frac{D}{M} \times t_{i,M} + \frac{q}{P} \frac{D}{M} \times M \times k \times t_{s,M}$$

Similarly, the algorithmic runtime per node using the *proposed* (HiCOPS) framework (w_H) is the sum of I/O for q spectra 1 time, indexing 1 slice of size D/P and searching q spectra against that slice, given as:

$$w_H = q \times t_{io} + t_{i,D/P} + q \frac{D}{P} \times k \times t_{s,D/P}$$

Since our proposed design performs I/O in parallel with database search (Online Methods, Superstep 3), the I/O overhead term can be pruned (better resource utilization). Further, as P increases, the database slice becomes smaller (already smaller

than M to fit inside RAM) making it safe to remove the indexing term as well.

$$w_H = q \frac{D}{P} \times k \times t_{s,D/P} < w_{ex}$$

Even if somehow all overhead terms in (w_{ex}) are ignored, the unit database search time $(t_{s,M})$ remains constant with increasing nodes (indifferent to available system memory). On the other hand, the unit time for HiCOPS $(t_{s,D/P})$ reduces with increasing P (see Figure 4 in Main text) or mathematically:

$$\frac{q}{P} D \times k \times t_{s,M} > q \frac{D}{P} \times k \times t_{s,D/P}$$

Since:

$$t_{s,M} > t_{s,D/P} ; \text{ since } M > D/P$$

Note the load balance factor has not been considered in the above analysis, which also contributes to the speed improvement achieved by HiCOPS over unbalanced tools. \square

2.5 Supplementary Section 5

Mod Distance. The proposed *Mod Distance* (Δm) is used as a supplementary metric in peptide database clustering in superstep 1 (the improved LBE method). The application of this metric can be best understood through an example. Consider three database peptide sequences p : MEGSYIRK, q : ME*GSYI*RK and r : MEGS*Y*IRK. The blue letters represent the normal amino acids in the peptide and the red letters with (*) represent the modified amino acids. Now, we can see that the Edit Distance between the pairs $\Delta e(p, q) = \Delta e(p, r) = 2$ (cannot differentiate). Now let us apply the *Mod Distance* on this scenario which considers the shared peaks between the peptide pairs to further separate them. For example, the shared (b- and y-) ions (or peaks) between p and q are: ME*GSYI*RK = 3, yielding $\Delta m(p, q) = 1.625$ and the peaks shared between p and r are: MEGS*Y*IRK = 6, yielding $\Delta m(p, r) = 1.25$. This indicates that the entries p and r should be located at relatively nearby database indices. The *Mod Distance* can be easily generalized for other ion-series such as: a-, c-, x-, z-ions and immonium ions as well.

2.6 Supplementary Section 6

Correctness of LBE. Let the peptide precursor m/z distribution of any given database is $g(m)$ and that of any given dataset is $f(m)$, then the LBE algorithm statically results in fairly balanced workloads at all parallel nodes.

Proof. The algorithmic workload $w(f, g)$ for database peptide search can be represented as the cost of performing the total number of scoring operations to search the dataset $f(m)$ against the database $g(m)$ using precursor mass tolerance of δM , fragment-ion tolerance of δF and shared peaks $\geq k$, mathematically:

$$w(f, g) = \text{cost} \left(\sum_{m=0}^{\infty} f(m) \sum_{z=-\delta M}^{\delta M} \text{filter}(f(m), g(m+z), \delta F, k) \right)$$

where $\text{filter}(f, g, \delta F, k)$ in our algorithm is:

$$\text{filter}(f, g, \delta F, k) = \text{count}(\text{shared_peaks}(f, g, \delta F) \geq k)$$

The above equations imply that the database distribution i.e. $\sum filter(f(m), g(m+z), \delta F, k)$ must be similar at all parallel nodes in order to achieve system-wide load balance. The LBE algorithm achieves this by localizing (by δM and shared peaks) the database entries and then finely scattering them across parallel nodes (Supplementary Figure 7) producing similar (distinct) local database distributions $g_{loc}(m)$ at parallel nodes thereby, similar workloads. This theorem can also be extended to incorporate *sequence-tag* based filtration methods in a straightforward manner. \square

2.7 Supplementary Section 7

Task Mapping. The parallel HiCOPS tasks are configured and deployed on system nodes based on the available resources, user parameters and the database size. The presented algorithm assumes a Linux based homogeneous multicore nodes cluster where the interconnected nodes have multicores, local shared memory and optionally a local storage as well. This is the most common architecture in modern supercomputers including XSEDE Comet, NERSC Cori etc. The resource information is read using Linux’s `lscpu` utility. Specifically, the information about shared memory per node (λ), NUMA nodes per node (u), cores per NUMA node (c_u), number of sockets per node (s) and cores per socket (c_s) is read. The total size of database (D) is then estimated using protein sequence database and user parameters. Assuming the total number of system nodes to be P , the parameters: number of MPI tasks per node (t_n) and the number of parallel cores per MPI task (t_c) and MPI task binding level (t_{bi}) are optimized as depicted in Supplementary Algorithm 5. The optimizations ensure that: 1) System resources are efficiently utilized 2) The MPI tasks have sufficient resources to process the database and 3) The MPI tasks have an exclusive access to a disjoint partition of local compute and memory resources.

Note that in Supplementary Algorithm 5, the lines 8 to 14 iteratively reduce the cores per MPI task while increasing the number of MPI tasks until the database size per MPI task is less than 48 million (empirically set for XSEDE Comet nodes). This was done to cap the memory usage per MPI process for superior performance. The while loop may be removed or modified depending on the database search algorithms and machine parameters.

2.8 Supplementary Section 8: Software Details

Minimum Environment

- SMP HPC cluster with Linux OS
- GCC/Intel/LLVM compiler with C++17.
- MPI with thread support, OpenMP.
- Python 3.7+ with common packages.
- CMake 3.18+ and other common packages.

Install

Comprehensive details about the required packages, supported environments and step by step installation of packages and HiCOPS are documented at: hicops.github.io/installation. This link will be updated as the development progresses.

Getting Started

The instructions for setting up the peptide database, experimental MS/MS dataset and running HiCOPS are documented at: hicops.github.io/getting_started.

Compatibility

The current version of HiCOPS accepts the experimental MS/MS data in the MS2 format and produces a peptide-to-spectrum matches (PSM) report in either tab delimited values (TSV) or Excel format. The future versions of HiCOPS will have native support for more formats including HDF5 based formats. For now, the input files can be converted to MS2 format using the provided `raw2ms2` tool and the output files can be converted to other formats such as pepXML, mzIdentML, idXML using either the `IDFileConverter` or `psm-convert` tools.

Integrating with HiCOPS framework

The details on integrating the existing and new algorithms with the HiCOPS parallel core library are documented at: hicops.github.io/getting_started/integrate. Currently, the integration is done via the provided functional interface (and data structures). Meanwhile, the integration is being redesigned with C++ template meta-programming interfaces. The documentation will be updated accordingly.

Command-line tools

Several command-line tools are distributed as a part of HiCOPS software. These tools provide support for runtime interface, preparation of database, dataset, and post-processing final results. A brief summary of each tool is documented at: hicops.github.io/tools.

3 Supplementary Algorithms

3.1 Supplementary Algorithm 1

Algorithm 1: Database Partitioning in Superstep 1

Data: peptide sequences (ϵ)
Result: database index partitions (D_i)

```
/* generate theoretical database entries */
1 for  $s$  in  $\epsilon$  do in parallel
2   for  $v$  in  $2^m$  do
3      $e \leftarrow \text{gen\_entry}(v)$ ;
4     /* add to local database if mine */
5     if  $\text{is\_mine}(e_v)$  then
6        $E.\text{append}(e)$ ;
7   /* generate model-spectra data */
8   for  $s$  in  $D_i$  do in parallel
9      $S.\text{append}(\text{model\_spectrum}(s))$ ;
10  /* index the database in parallel */
11   $D_i \leftarrow \text{map}(\text{parallel\_sort}(E), \text{parallel\_index}(S))$ ;
12  /* return the indexed parital database */
13  return  $D_i$ ;
```

3.2 Supplementary Algorithm 2

Algorithm 2: Data load (per thread) by sub-task R (Superstep 3)

Data: forward queue (q_f), recycle queue (q_r), pointer stack (s_d), batch index (i_d)

```
/* loop unless  $q_f$  full, preempted or no more batches */
1 while  $\sim q_f.full()$  do
  /* check pointer stack */
  2 if  $\sim dp$  then
  3    $dp \leftarrow s_d.pop()$ ;
  /* if stack is empty, get a new pointer */
  4 if  $\sim dp$  then
  5    $dp \leftarrow i_d.pop()$ ;
  /* no more experimental data batches - exit */
  6 if  $\sim dp$  then
  7   break;
  /* check preemption state and  $q_r$  status */
  8 if  $\sim preempt()$  or  $\sim q_r.empty()$  then
  9    $s_d.push(dp)$ ;
 10  break;
 11 else
 12   /* else get a buffer from  $q_r$  */
 13    $bp \leftarrow q_r.pop()$ ;
  /* read a batch of expt data */
 14   $dp.read\_batch(bp)$ ;
  /* push the buffer to  $q_f$  */
   $q_f.push(bp)$ ;
```

3.3 Supplementary Algorithm 3

Algorithm 3: Partial DB search by sub-task R (Superstep 3)

Data: forward queue (q_f), recycle queue (q_r), database partition (D_{p_i}),
result queue (q_k)

```
/* extract a batch from queue */
1  $b \leftarrow q_f.pop()$ ;
/* data parallel search */
2 for  $e$  in  $b$  do in parallel
    /* apply the precursor mass filter */
    3  $\sigma_{p_i} \leftarrow filter_1(D_{p_i}, e)$ ;
    4 if  $\sigma_{p_i}$  then
        5 for  $\beta$  in  $e$  do
            /* apply the shared peaks filter */
            6  $\mu_{p_i}.append(filter_2(\sigma_{p_i}, \beta))$ ;
            /* score against the filtered database */
            7 for  $h$  in  $\mu_{p_i}$  do
                8  $heap.push(k \leftarrow score(h, e))$ ;
            /* append to a batch of intermediate results */
            9  $res_i.append(heap)$ ;
        /* recycle the buffer back to  $q_r$  */
    10  $q_r.push(b)$ ;
    /* push the intermediate results batch to  $q_k$  */
    11  $q_k.push(res_i)$ ;
```

3.4 Supplementary Algorithm 4

Algorithm 4: Result Assembly in Superstep 4

Data: rank p_i , Intermediate Result batches (r_i)
Result: expect scores (ev)

```

/* extract a batch from queue */
1  $b \leftarrow q_f.pop()$ ;
/* get batches that satisfy the condition */
2 for  $b$  in ( $b \bmod p_i = 0$ ) do
3    $l.append(b)$ ;
/* data parallel assembling of results for each batch */
4 for  $s$  in  $l$  do in parallel
5   /* assemble the null distribution */
    $dist \leftarrow assemble(s)$ ;
6   /* max heapify the scores */
    $heap \leftarrow make\_heap(s)$ ;
7   /* use either fitting method */
    $fit \leftarrow logWeibullFit(dist)$ ;
8    $fit \leftarrow TailFit(dist)$ ;
9   /* get the top hit from heap */
    $g_{max} \leftarrow heap.pop().value()$ ;
10  /* compute the expect score */
    $ev \leftarrow (fit.w \times g_{max} + fit.b) \times heap.size()$ ;
11  /* push results to a map structure */
    $map.push(key = g_{max}.key(), val = ev)$ ;
/* asynchronous scatter complete result data */
12 for  $p_i$  in  $P$  do in parallel
13    $isend(map.data(key = p_i), dst = p_i)$ ;
/* synchronize using barrier */
14  $barrier()$ ;
/* write the results to the file system */
15  $write(map.data(key = rank))$ ;

```

3.5 Supplementary Algorithm 5

Algorithm 5: Task Mapping

Data: number of nodes (n), node parameters (λ, u, c_u, s, c_s) and database size (D)

Result: number of MPI tasks per node (t_n), cores per MPI task (t_c) and MPI binding level (t_{bl})

```
/* ensure enough memory for database */
1 if  $D_{p_i} \leftarrow D/P > 0.70\lambda$  then
2   | return err;
   /* set MPI binding level */
3  $t_{bl} \leftarrow \max\{u, c\}$ ;
   /* set MPI binding policy */
4  $t_{bp} \leftarrow \text{scatter}$ ;
   /* set cores per MPI task */
5  $t_c \leftarrow \min\{c_u, c_s\}$ ;
   /* set number of MPI tasks per node */
6  $t_n \leftarrow \max\{u, c\}$ ;
7  $t_{max} \leftarrow t_c$ ;
   /* Optional: optimize for memory bandwidth */
8 while ( $D/t_n > 48 \times 10^6$ ) do
   /* Choose the next highest factor of  $t_{max}$  */
9   |  $n_{poss} \leftarrow \text{factorize}(t_{max})$ ;
10  | if  $n_{poss} \geq t_{max}/2$  then
11  |   |  $t_n \leftarrow t_n \times t_{max}/n_{poss}$ ;
12  |   |  $t_c \leftarrow n_{poss}$ ;
13  |   else
14  |     | break;
15 return  $t_n, t_c, t_{bl}, t_{bp}$ ;
```
