HHS Public Access

Author manuscript

Nat Comput Sci. Author manuscript; available in PMC 2022 February 20.

Published in final edited form as:

Nat Comput Sci. 2021 August ; 1(8): 550–561. doi:10.1038/s43588-021-00113-z.

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

Muhammad Haseeb[1], **Fahad Saeed**[2,3,4,5]

[1]Knight Foundation School of Computing and Information Sciences, Florida International University, Miami, FL, USA.

[2]Knight Foundation School of Computing and Information Sciences, Florida International University, Miami, FL, USA.

[3]Biomolecular Sciences Institute (BSI), Florida International University, Miami, FL, USA.

[4]Department of Human and Molecular Genetics, Herbert Wertheim School of Medicine, Florida International University, Miami, FL, USA.

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

[5]Corresponding Author. fsaeed@fiu.edu.

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.

Competing Interests

The authors declare no competing interests.

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, meta-proteomics 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:

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

Or simply:

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

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

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

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: $\exp_n = (q, D, \delta M)$; 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 ±100Da. Note that the tuple does not contain the fragmention mass tolerance (δF) information as it is globally set to ±0.01 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 × 12 cores of Intel Xeon E5-2680v3 processor (Total: 24 cores), 2 NUMA nodes × 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 >48h (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 (δM=1Da) and three with open search ($\delta M \geq$100Da) 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 thoretical database of 18 million spectra at δM = 1Da. 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 δM = 1Da. 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 δM = 1Da. 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 δM = 200Da. 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 δM = 100Da. 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 δM = 200Da.

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 δM = 10Da and δM = 500Da 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 δM = 500Da. 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 δM = 10Da and δM = 500Da respectively. In the sixth (**f**) experiment, dataset S_4 was searched against the database: D_1 modified with variable Methionine oxidation, STY-phosphorylation and NQ-deamidation yielding a theoretical database of: 213 million at δM = 100Da. 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× 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× 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 δM = ±500Da and δF = ±0.01Da. In the second experiment, the dataset: S_4 was searched against a theoretical database of 1.59 billion spectra (1.7TB) at δM = ±500Da and δF = ±0.05Da. In the third experiment, the dataset: S_2 was searched against a theoretical database of 3.89 billion spectra (4TB) at δM = ±500Da and δF = ±0.01Da. HiCOPS completed the execution of the 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× slower). The second experiment was completed by MSFragger in 18 hours (72 nodes; 10.3× slower) and 35.5 days when using 1 node (494× 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: exp_1 = (0.3, 0.84, 0.1), exp_2 = (0.3, 0.84, 2), exp_3 = (3.89, 0.07, 5), exp_4 = (1.51, 2.13, 5), exp_5 = (6.1, 0.93, 5), exp_6 = (3.89, 7.66, 5), exp_7 = (1.51, 19.54, 5), exp_8 = (1.6, 38.89, 5), exp_9 = (3.89, 15.85, 5), exp_{10} = (3.89, 1.08, 5), exp_{11} = (1.58, 2.13, 1), and exp_{12} = (0.305, 0.847, 5). Note that the fragment-ion tolerance is set to δF = ±0.01Da 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 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 ≥ 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 exp_5 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 ≤10%, I/O costs remain ≤10%, and inter-task communication costs remain ≤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 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 ≤ 5% in most experiments (Figure 5d).

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 homogeneous (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.

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 ($T_{j,pi}$) and the generic overheads due to boilerplate code, OS delays, memory allocation etc. will be captured via (γ_{pi}). 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_{pi} cores) will be given as follows:

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

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 $c_{pi}/2$ cores each, the execution time will be given as:

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

For analysis purposes, if the time complexity of the algorithms used for step k_{jz} is known (say $O(.)$), 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:

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

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):

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

Cost Analysis: The first step generates the entire database of size (D) and separates out a local partition (of roughly the size $D/P = D_{pi}$) 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: $k_{12}(D_{pi})$. The third step constructs a fragmention 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: $q\beta/P = Q_{pi}$) using a data pre-processing algorithm such as [47], [5], [44] etc. in runtime: $k_{22}(Q_{pi})$. The third step may write the pre-processed data back to the file system in runtime: $k_{23}(Q_{pi})$. 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 subtasks 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:

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

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: $k_{31}(qD_{pi})$ and $k_{32}(q\beta\alpha_{pi})$ respectively. Here, the α_{pi} represents the average filtered database size filtered from the first step. The currently implemented scoring mechanism computes hyperscores [13] in runtime: $k_{33}(q\beta\sigma_{pi}) + k_{34}(q\alpha\gamma_{pi})$. Here, the σ_{pi} and μ_{pi} 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: $V_{pi}(q, D_{pi}, P)$. 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. $t_{pi}(r, \mid r \mid)$, is given as:

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

The runtime of sub-task *I*, i.e. $t_{pi}(i, \mid i \mid)$, is given as:

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

Or:

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

The runtime of sub-task *K*, i.e. $t_{pi}(k, \mid k \mid)$, is given as:

$$t_{p_i}(k, \mid k \mid) = k_{35}(q, \mid k \mid) \tag{9}$$

Combining equations 7, 8 and 9 we have:

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

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* mod $P = p_i$; $p_i \epsilon$ 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 $i = 0,1,.., P-1$. 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_{pi}) 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: $k_{41}(Q_{pi}, c_{pi}) + k_{42}(Q_{pi}, c_{pi}) + k_{43}(P, 1)$ 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: $k_{45}(Q_{pi})$. Collectively, the runtime for this superstep is given by equation 11.

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

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

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

Assuming that the network latency is denoted as (ω), bandwidth is denoted as (π) and ($16Q_{pi}$) 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:

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

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

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

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

and:

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

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:

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

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:

$$\begin{aligned} T_F = max(t_{p_i}(t, \mid r \mid), t_{p_i}(i, \mid i \mid), t_{p_i}(k, \mid k \mid)) + k_{41}(Q_{p_i}, c_{p_i}) + \\ k_{42}(Q_{p_i}, c_{p_i}) + + k_{45}(Q_{p_i}) + T_o \end{aligned} \tag{18}$$

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:

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

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 $t_{pi}(i, \mid i \mid)$ and $t_{pi}(k, \mid k \mid)$ allowing a safe removal from the above equation yielding:

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

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

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

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:

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

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 $c_{pi} + 2$ threads where c_{pi} is the number of available cores. In the first iteration, 2 threads are assigned to the sub-tasks R and K while the remaining $c_{pi} - 2$ 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:

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

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.

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

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: $k_{33}(q\beta\sigma_{pi}, \mid i \mid) + k_{34}(q\mu_{pi}, \mid i \mid)$. Here, the terms $q\beta$ and q are constants (experimental data size) whereas the terms σ_{pi} and μ_{pi} 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 resultc 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: ~20MB, 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-: $\delta M \sim \pm 500$Da and closed-search: $\delta M \leq \pm 10$Da scenarios. The closed-search criterion was set to a few Daltons (≤1Da in correctness analysis and ≤10Da 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 *Accession Num* 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].

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

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.

References

[1]. Nesvizhskii Alexey I. 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. [PubMed: 20816881]

[2]. Kong Andy T, Leprevost Felipe V, Avtonomov Dmitry M, Mellacheruvu Dattatreya, and Nesvizhskii Alexey I. Msfragger: ultrafast and comprehensive peptide identification in mass spectrometry-based proteomics. Nature methods, 14(5):513, 2017. [PubMed: 28394336]

[3]. McIlwain Sean, Tamura Kaipo, Kertesz-Farkas Attila, Grant Charles E, Diament Benjamin, Frewen Barbara, Jeffry Howbert J, Hoopmann Michael R, Kall Lukas, Eng Jimmy K, et al. Crux: rapid open source protein tandem mass spectrometry analysis. Journal of proteome research, 13(10):4488–4491, 2014. [PubMed: 25182276]

[4]. ei Yuan Zuo-F, Liu Chao, Wang Hai-Peng, Sun Rui-Xiang, Fu Yan, Zhang Jing-Fen, Wang Le-Heng, Chi Hao, Li You, Xiu Li-Yun, et al. pparse: A method for accurate determination of monoisotopic peaks in high-resolution mass spectra. Proteomics, 12(2):226–235, 2012. [PubMed: 22106041]

[5]. Deng Yamei, Ren Zhe, Pan Qingfei, Qi Da, Wen Bo, Ren Yan, Yang Huanming, Wu Lin, Chen Fei, and Liu Siqi. pclean: an algorithm to preprocess high-resolution tandem mass spectra for database searching. Journal of proteome research, 18(9):3235–3244, 2019. [PubMed: 31364357]

[6]. Degroeve Sven and Martens Lennart. Ms2pip: a tool for ms/ms peak intensity prediction. Bioinformatics, 29(24):3199–3203, 2013. [PubMed: 24078703]

[7]. Zhou Xie-Xuan, Zeng Wen-Feng, Chi Hao, Luo Chunjie, Liu Chao, Zhan Jianfeng, He Si-Min, and Zhang Zhifei. pdeep: predicting ms/ms spectra of peptides with deep learning. Analytical chemistry, 89(23):12690–12697, 2017. [PubMed: 29125736]

[8]. Zhang Jing, Xin Lei, Shan Baozhen, Chen Weiwu, Xie Mingjie, Yuen Denis, Zhang Weiming, Zhang Zefeng, Lajoie Gilles A, and Ma Bin. Peaks db: de novo sequencing assisted database search for sensitive and accurate peptide identification. Molecular & Cellular Proteomics, 11(4):M111–010587, 2012.

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

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

[11]. Bern Marshall, Cai Yuhan, and Goldberg David. 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. [PubMed: 17243770]

[12]. Eng Jimmy K, McCormack Ashley L, and Yates John R. 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. [PubMed: 24226387]

[13]. Craig Robertson and Beavis Ronald C. 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. [PubMed: 14558131]

[14]. Diament Benjamin J and Noble William Stafford. Faster sequest searching for peptide identification from tandem mass spectra. Journal of proteome research, 10(9):3871–3879, 2011. [PubMed: 21761931]

[15]. Eng Jimmy K, Fischer Bernd, Grossmann Jonas, and MacCoss Michael J. A fast sequest cross correlation algorithm. Journal of proteome research, 7(10):4598–4602, 2008. [PubMed: 18774840]

[16]. Park Christopher Y, Klammer Aaron A, Kall Lukas, MacCoss Michael J, and Noble William S. Rapid and accurate peptide identification from tandem mass spectra. Journal of proteome research, 7(7):3022–3027, 2008. [PubMed: 18505281]

[17]. Geer Lewis Y, Markey Sanford P, Kowalak Jeffrey A, Wagner Lukas, Xu Ming, Maynard Dawn M, Yang Xiaoyu, Shi Wenyao, and Bryant Stephen H. Open mass spectrometry search algorithm. Journal of proteome research, 3(5):958–964, 2004. [PubMed: 15473683]

[18]. Hebert Alexander S, Richards Alicia L, Bailey Derek J, Ulbrich Arne, Coughlin Emma E, Westphall Michael S, and Coon Joshua J. The one hour yeast proteome. Molecular & Cellular Proteomics, 13(1):339–347, 2014. [PubMed: 24143002]

[19]. Nesvizhskii Alexey I, Roos Franz F, Grossmann Jonas, Vogelzang Mathijs, Eddes James S, Gruissem Wilhelm, Baginsky Sacha, and Aebersold Ruedi. 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. [PubMed: 16352522]

[20]. Eng Jimmy K, Searle Brian C, Clauser Karl R, and Tabb David L. A face in the crowd: recognizing peptides through database search. Molecular & Cellular Proteomics, pages mcp-R111, 2011.

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

[23]. Chi Hao, He Kun, Yang Bing, Chen Zhen, Sun Rui-Xiang, Fan Sheng-Bo, Zhang Kun, Liu Chao, Yuan Zuo-Fei, Wang Quan-Hui, 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. [PubMed: 25979774]

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

[25]. Duncan Dexter T, Craig Robertson, and Link Andrew J. 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. [PubMed: 16212440]

[26]. Bjornson Robert D, Carriero Nicholas J, Colangelo Christopher, Shifman Mark, Cheung Kei-Hoi, Miller Perry L, and Williams Kenneth. 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. [PubMed: 17902638]

[27]. Pratt Brian Jeffry Howbert J, Tasman Natalie I, and Nilsson Erik J. Mr-tandem: parallel x! tandem using hadoop mapreduce on amazon web services. Bioinformatics, 28(1):136–137, 2011. [PubMed: 22072385]

[28]. Li Chuang, Li Kenli, Li Keqin, and Lin Feng. Mctandem: an efficient tool for large-scale peptide identification on many integrated core (mic) architecture. BMC bioinformatics, 20(1):397, 2019. [PubMed: 31315562]

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

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

[31]. Prakash Amol, Ahmad Shadab, Majumder Swetaketu, Jenkins Conor, and Orsburn Ben. 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. [PubMed: 31452088]

[32]. Kaiser Patricia, Bode Maya, Cornils Astrid, Hagen Wilhelm, Arbizu Pedro Martínez, Auel Holger, and Laakmann Silke. 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]. Rossel S and Martínez Arbizu P. 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. [PubMed: 30626917]

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

[35]. Beyter Doruk, Lin Miin S, Yu Yanbao, Pieper Rembert, and Bafna Vineet. Proteostorm: An ultrafast metaproteomics database search framework. Cell systems, 7(4):463–467, 2018. [PubMed: 30268435]

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

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

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

[39]. Haseeb Muhammad and Saeed Fahad. Source Data: High Performance Computing Framework for Tera-Scale Database Search of Mass Spectrometry Data. , 7 2021.

[40]. Eng Jimmy K, Jahan Tahmina A, and Hoopmann Michael R. Comet: an open-source ms/ms sequence database search tool. Proteomics, 13(1):22–24, 2013. [PubMed: 23148064]

[41]. Craig Robertson and Beavis Ronald C. Tandem: matching proteins with tandem mass spectra. Bioinformatics, 20(9):1466–1467, 2004. [PubMed: 14976030]

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

[43]. Stevens Rick, Ramprakash Jini, Messina Paul, Papka Michael, and Riley Katherine. Aurora: Argonne's next-generation exascale supercomputer. Technical report, ANL (Argonne National Laboratory (ANL), Argonne, IL (United States)), 2019.

[44]. Liu Kaiyuan, Li Sujun, Wang Lei, Ye Yuzhen, and Tang Haixu. Full-spectrum prediction of peptides tandem mass spectra using deep neural network. Analytical chemistry, 92(6):4275–4283, 2020. [PubMed: 32053352]

[45]. Lin Yang-Ming, Chen Ching-Tai, and Chang Jia-Ming. Ms2cnn: predicting ms/ms spectrum based on protein sequence using deep convolutional neural networks. BMC genomics, 20(9):1–10, 2019. [PubMed: 30606130]

[46]. Haseeb Muhammad, Afzali Fatima, and Saeed Fahad. 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]. Ding Jiarui, Shi Jinhong, Poirier Guy G, and Wu Fang-Xiang. A novel approach to denoising ion trap tandem mass spectra. Proteome Science, 7(1):9, 2009. [PubMed: 19292921]

[48]. Fenyö David and Beavis Ronald C. A method for assessing the statistical significance of mass spectrometry-based protein identifications using general scoring schemes. Analytical chemistry, 75(4):768–774, 2003. [PubMed: 12622365]

[49]. LaViola Joseph J. 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]. Haseeb Muhammad and Saeed Fahad. hicops software code v1.0.0 – 1st public release, 7 2021.

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

[52]. Kulkarni Gaurav, Kalyanaraman Ananth, Cannon William R, and Baxter Douglas. 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.

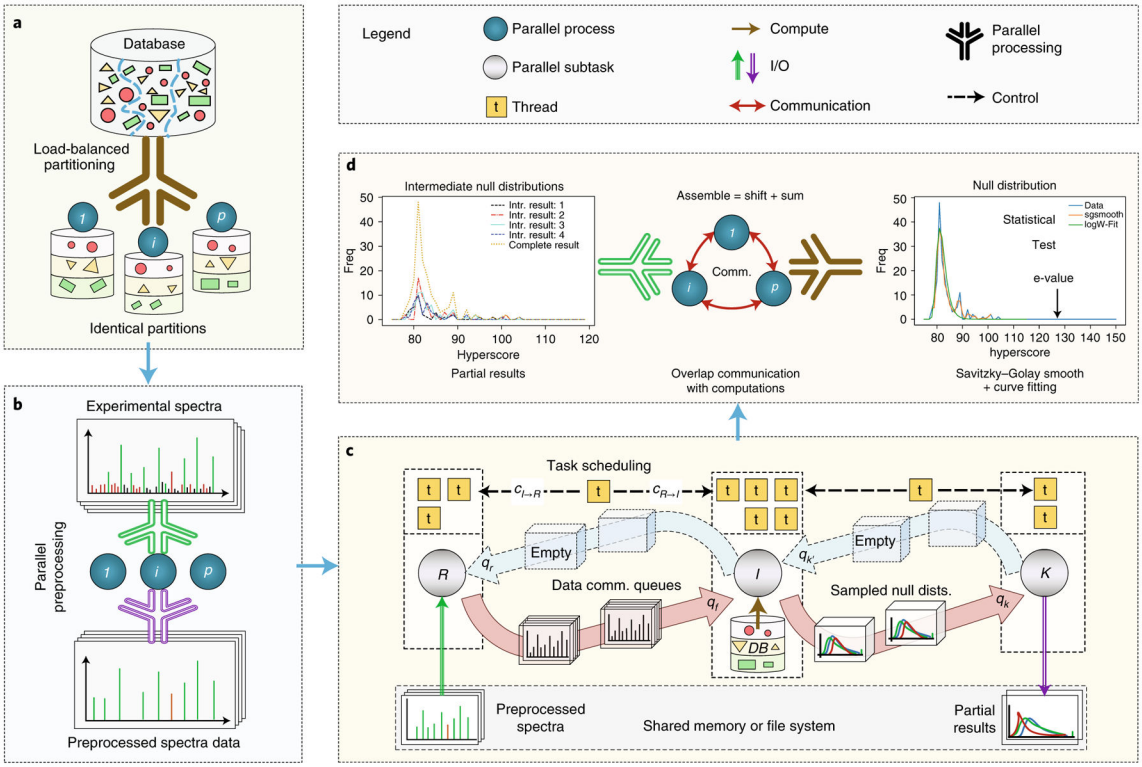

Figure 1: Methods Overview.

(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.

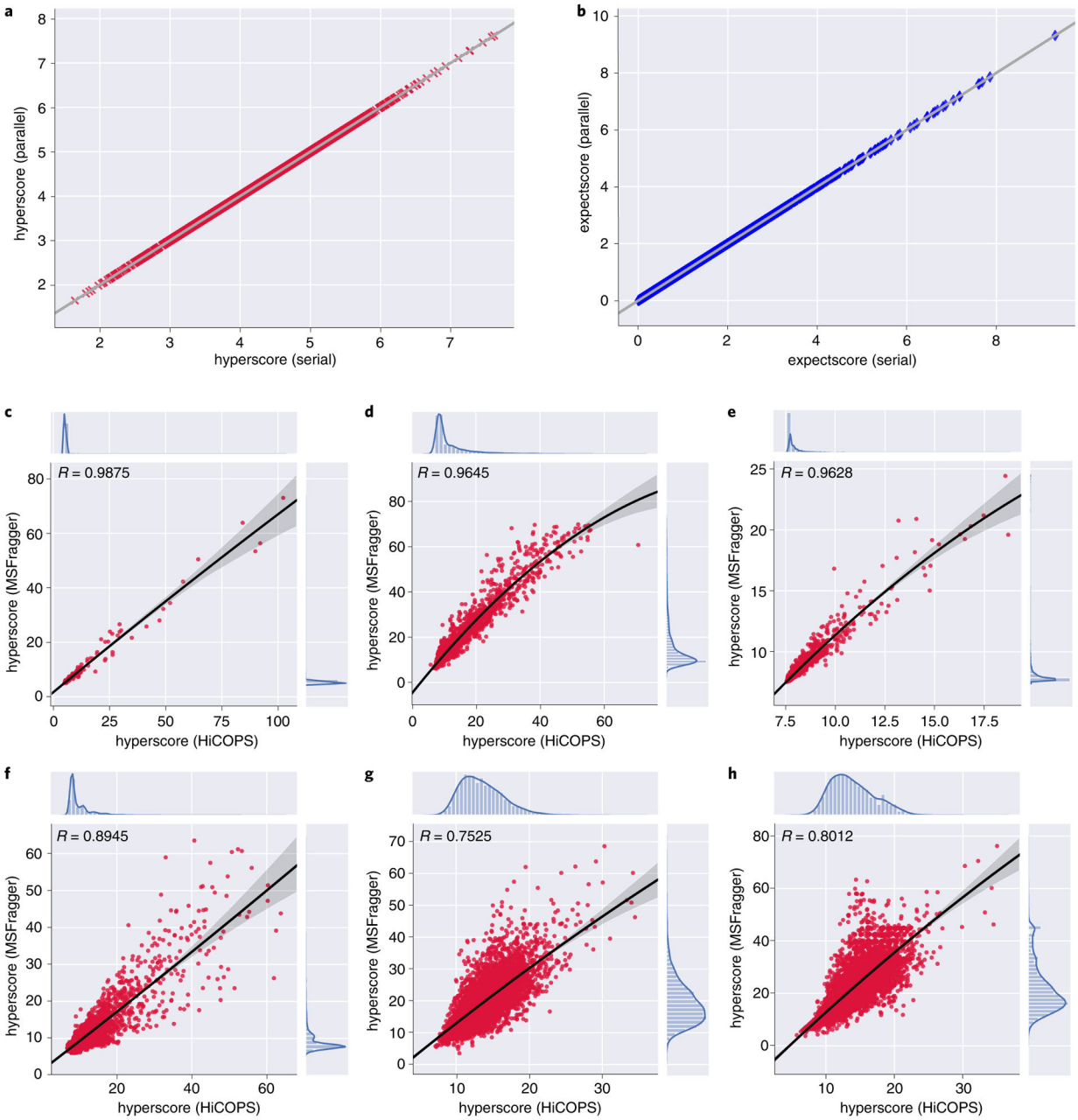

Figure 2: Correctness Analysis.

(a,b) Comparison of 5K out of 251K data samples of hyperscores and expected values (expectscores), 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*).

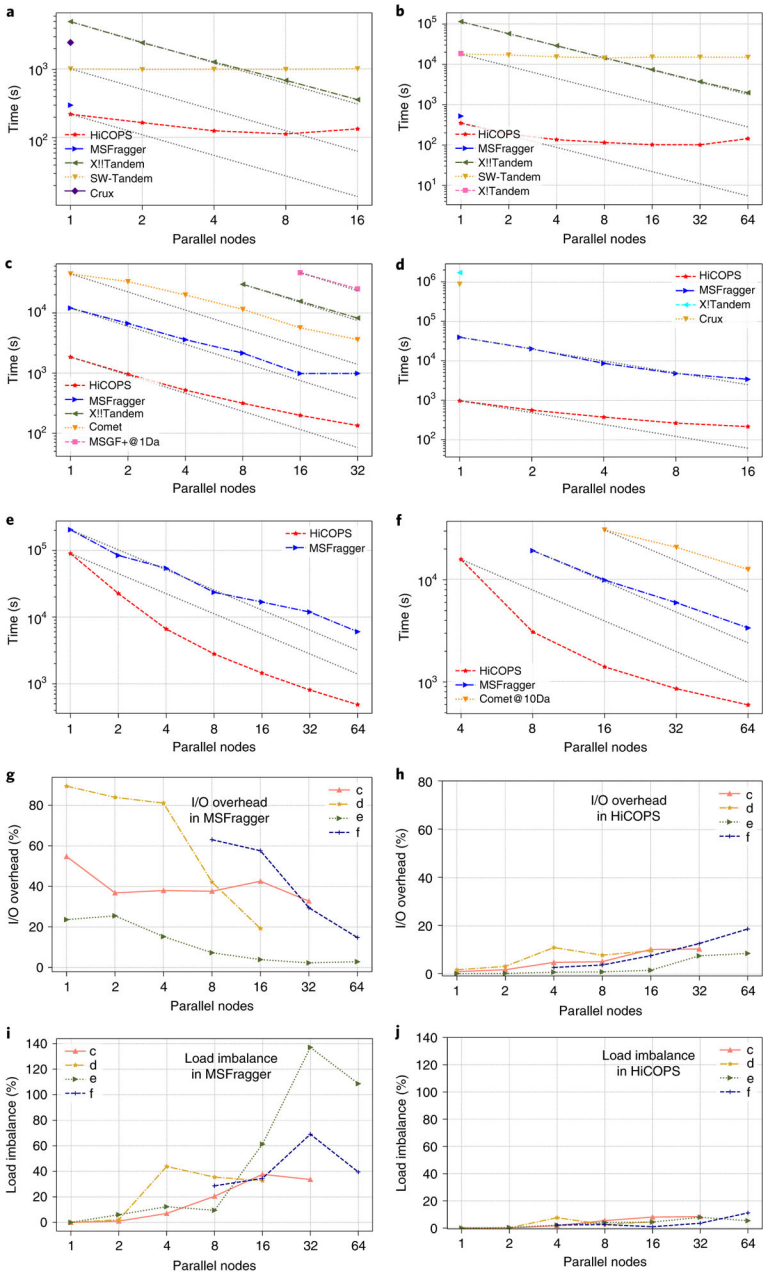

Figure 3: Speed Comparisons.

(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 MS-Fragger for experiments in sub-figures (**c**, **d**, **e**, and **f**), are shown with increasing number of parallel nodes.

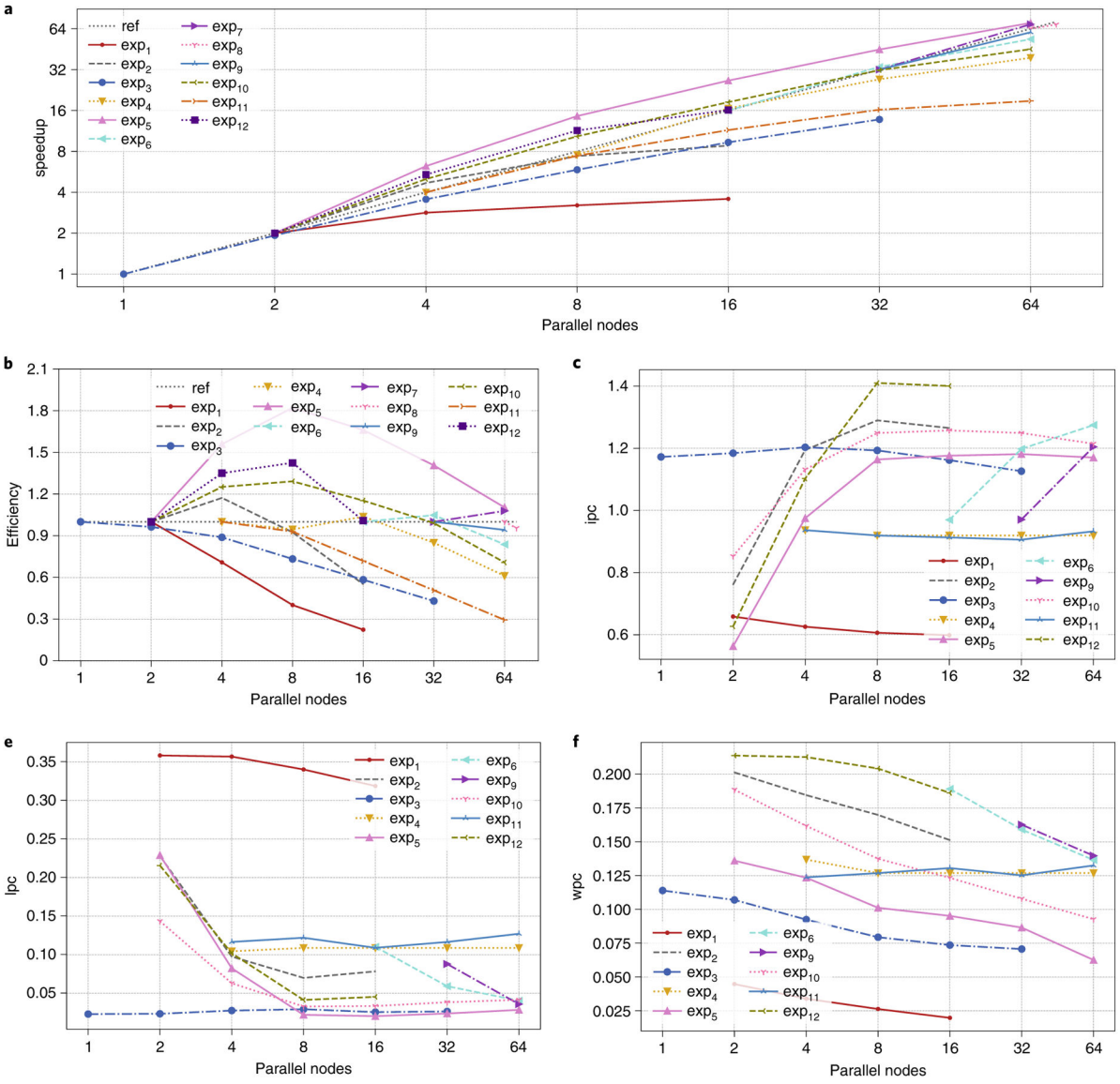

Figure 4: Performance Metrics.

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.

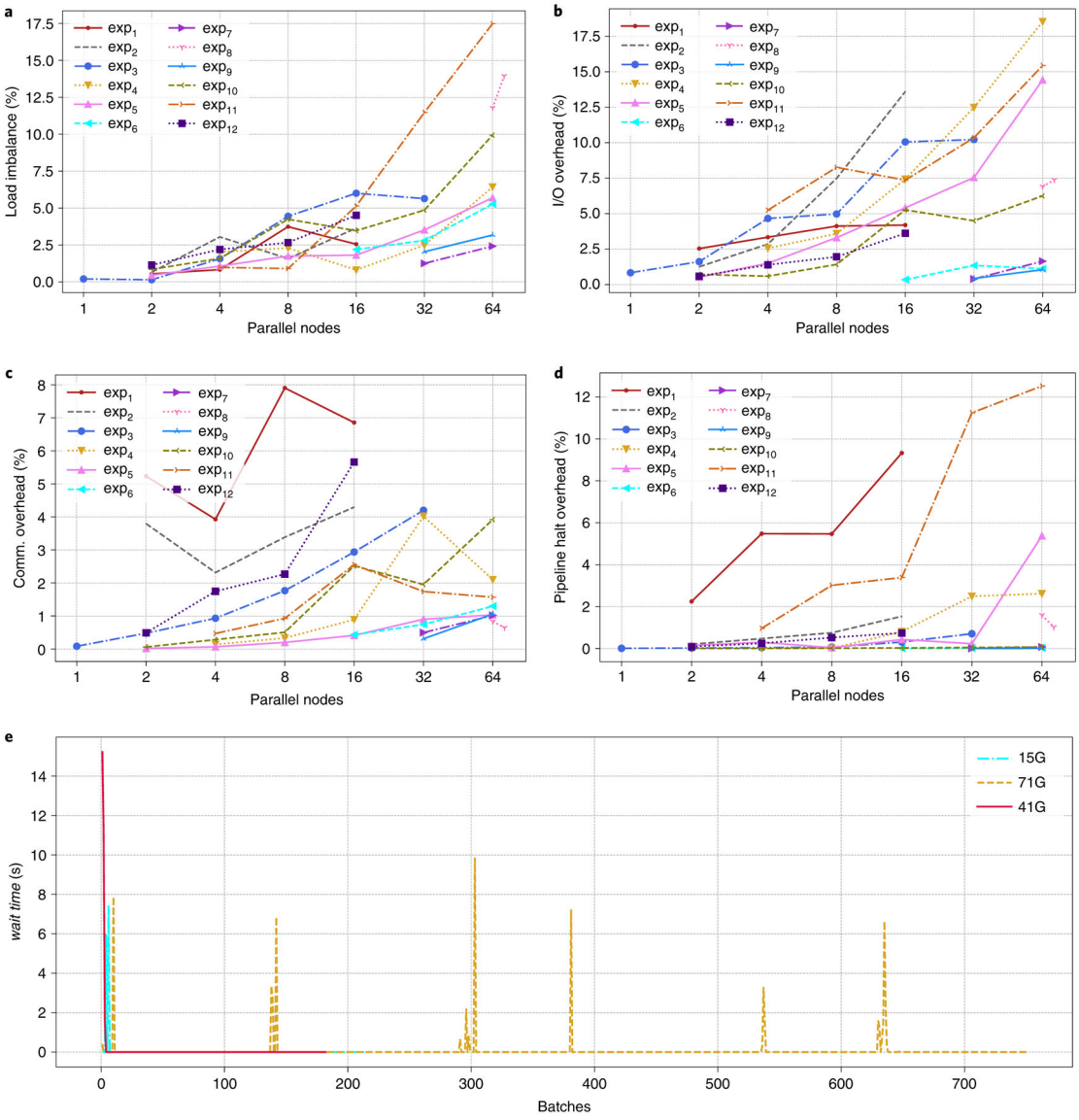

Figure 5: Overhead Analysis.

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.

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