ARTICLES
https://doi.org/10.1038/s43588-021-00113-z

nature computational science

Check for updates

High performance computing framework for tera-scale database search of mass spectrometry data

Muhammad Haseeb[1]✉ and Fahad Saeed[1,2,3]✉

Database peptide search algorithms deduce peptides from mass spectrometry 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 suboptimal 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 tenfold 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.

Faster and more efficient peptide identification algorithms[1–3] have been the cornerstone of computational research in shotgun mass spectrometry-based proteomics for more than 30 years[2–17]. Modern mass spectrometry technologies allow the generation of thousands of raw, noisy spectra in the span of a few hours, producing several gigabytes of data[18] (Supplementary Fig. 1). Database peptide search is the most commonly employed computational approach to identify peptides from experimental spectra[2,10,19,20]. In this approach, the experimental spectra are searched against an (indexed) database of theoretical spectra (or modeled spectra) with the goal of finding the best-possible matches[1]. The theoretical spectra database (or simply the theoretical database) is constructed by simulating in silico digestion on a proteome sequence database (Supplementary Fig. 2). The theoretical databases (and their indexed versions) expand exponentially in space (several gigabytes to terabytes) as the post-translational modifications (PTMs) are added in the simulation[2,21] (Supplementary Fig. 3a,b). Consequently, the low computational arithmetic intensity (operations or instructions per byte[22]) inherent to database search algorithms[2,9,23] 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[25–31] employ unoptimized parallelization techniques that lead to suboptimal performance and limited application in the domain (Supplementary Sections 1 and 2, and Supplementary Fig. 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 distributed-memory supercomputers. HiCOPS exhibits orders-of-magnitude improvement in speed compared with several existing shared- and distributed-memory database peptide search tools, allowing several gigabytes of experimental MS/MS data to be searched against terabytes of theoretical databases in a few minutes compared with the several hours required by existing algorithms. The proposed HiCOPS parallel design implements an unconventional approach in which the (massive) theoretical databases are distributed across parallel nodes in a load-balanced fashion followed by asynchronous parallel execution of the database peptide search. On 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 performance bottlenecks[35]. 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[2,11,36] in HiCOPS to demonstrate its parallel performance, but in essence, our framework is search-algorithm oblivious, that is, 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 tenfold on average while producing correct and consistent peptide identifications. Furthermore, we demonstrate the application of HiCOPS in a 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 in

[1]Knight Foundation School of Computing and Information Sciences, Florida International University, Miami, FL, USA. [2]Biomolecular Sciences Institute (BSI), Florida International University, Miami, FL, USA. [3]Department of Human and Molecular Genetics, Herbert Wertheim School of Medicine, Florida International University, Miami, FL, USA. ✉e-mail: mhase003@fiu.edu; fsaeed@fiu.edu

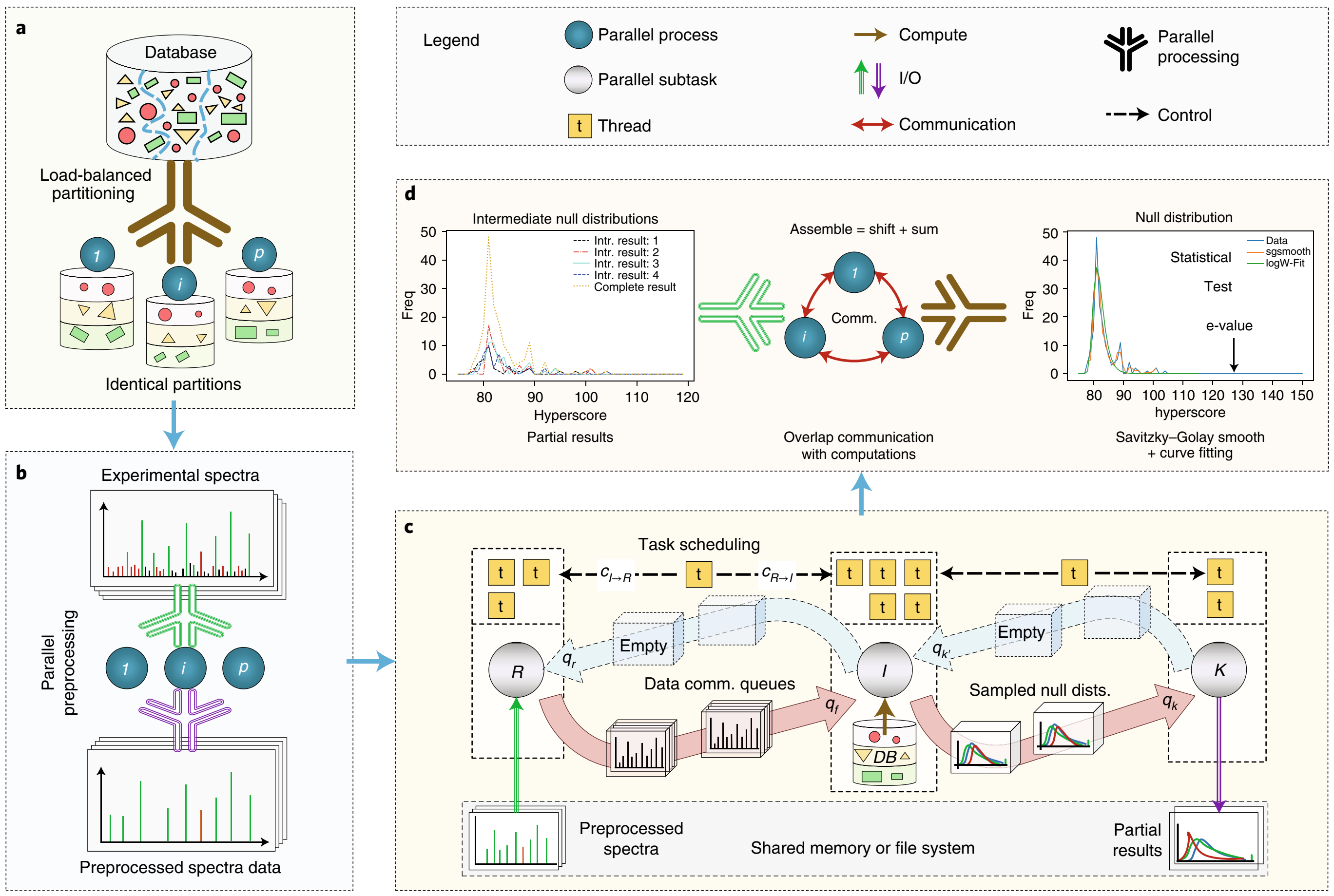

Fig. 1 | Methods overview. **a**, Superstep 1: the massive theoretical spectra database (spectra are shown as shapes) is partitioned among parallel processes and locally indexed. Partitioning is performed in a load-balanced fashion (similar shapes are clustered and scattered across processes). **b**, Superstep 2: the experimental MS/MS spectra data are indexed, tagged, preprocessed and written back to a shared memory in data parallel. **c**, Superstep 3: an asynchronous parallel database peptide search is executed by all processes. In each process, three parallel subtasks *R*, *I* and *K* work in a pipeline to load the preprocessed data, execute a local search and write the produced (sampled) local results to the shared memory, respectively. The task scheduler manages the parallel threads between the pipeline tasks. **d**, Superstep 4: local or intermediate results are assembled followed by curve fitting and expected value computation in data-parallel fashion. Results with expected values < 0.01 are communicated to their origin processes.

which we report between 70 to 80% strong-scale efficiency and less than 25% overall performance overheads (load imbalance, I/O, interprocess communication, pipeline halt); collectively depicting a near-optimal parallel performance.

Results

Methods overview. HiCOPS constructs the parallel database peptide search workflow (task-graph) through four Single Program Multiple Data (SPMD) Bulk Synchronous Parallel (BSP)[37] supersteps. In the BSP model, a superstep[38] refers to a set of distinct algorithmic and data communications blocks asynchronously executed by all parallel processes ($p_i \epsilon P$). Synchronization between the processes is performed at the end of each superstep, as needed. In the first superstep (Fig. 1a), the (massive) theoretical database is partitioned across parallel processes in a load-balanced fashion and locally indexed. In the second superstep (Fig. 1b), the experimental data are divided into batches and preprocessed, if required. In the third superstep (Fig. 1c), the parallel processes execute a local database peptide search, producing intermediate results. In the final superstep (Fig. 1d), the intermediate results are deserialized and assembled into complete (global) results. Supplementary Fig. 4 provides an overview of the overall task-graph as well as the workload profile for each superstep (Methods). The current HiCOPS design allows in-core processing so that the minimum number of nodes ($P_{\min}$) required must be $\geq D/M$, where D is the theoretical database index size and M is the available main memory per node.

The total HiCOPS wall time (T_H) is equal to the sum of individual 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 ($p_i \epsilon P$) 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:

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

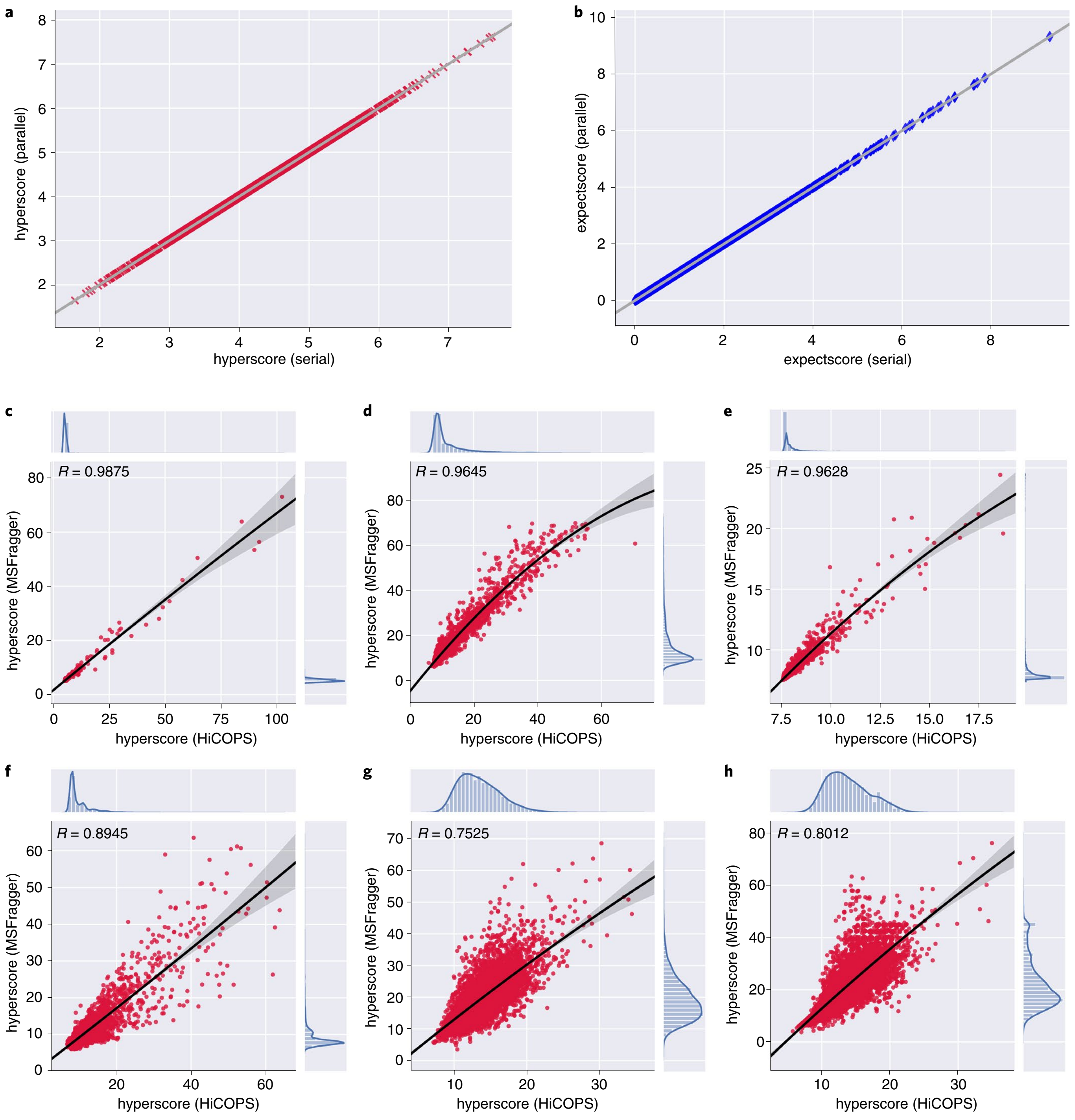

Fig. 2 | Correctness analysis. a,**b**, Comparison of 10,000 out of 251,000 data samples of hyperscores (**a**) and expected values (expectscores) (**b**) computed by HiCOPS in serial (*x*-axis) and parallel (*y*-axis) runs is shown. Note that all 251,000 samples depict the same consistency across parallel runs[51], but they were not feasible to plot. **c**–**h**, Correlations between hyperscores computed by HiCOPS (*x*-axis) and MSFragger (*y*-axis) for the three restricted-search experiments (**c**–**e**) and their corresponding open-search versions (**f**–**h**) (described in section: Correctness analysis) are shown along with Pearson correlation coefficients (*R*).

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 and 015391 combined) and S_5 (all of the above listed datasets combined). The datasets were searched against several theoretical databases constructed by adding combinations of PTMs to databases D_1 (UniProt *Homo sapiens*; UP000005640) and D_2 (UniProt SwissProt; reviewed). See the Detailed experimental setup for detailed discussion on the settings for database digestion, PTMs, theoretical spectra generation and so on. 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 one-million spectra, D is theoretical database size in 100 million

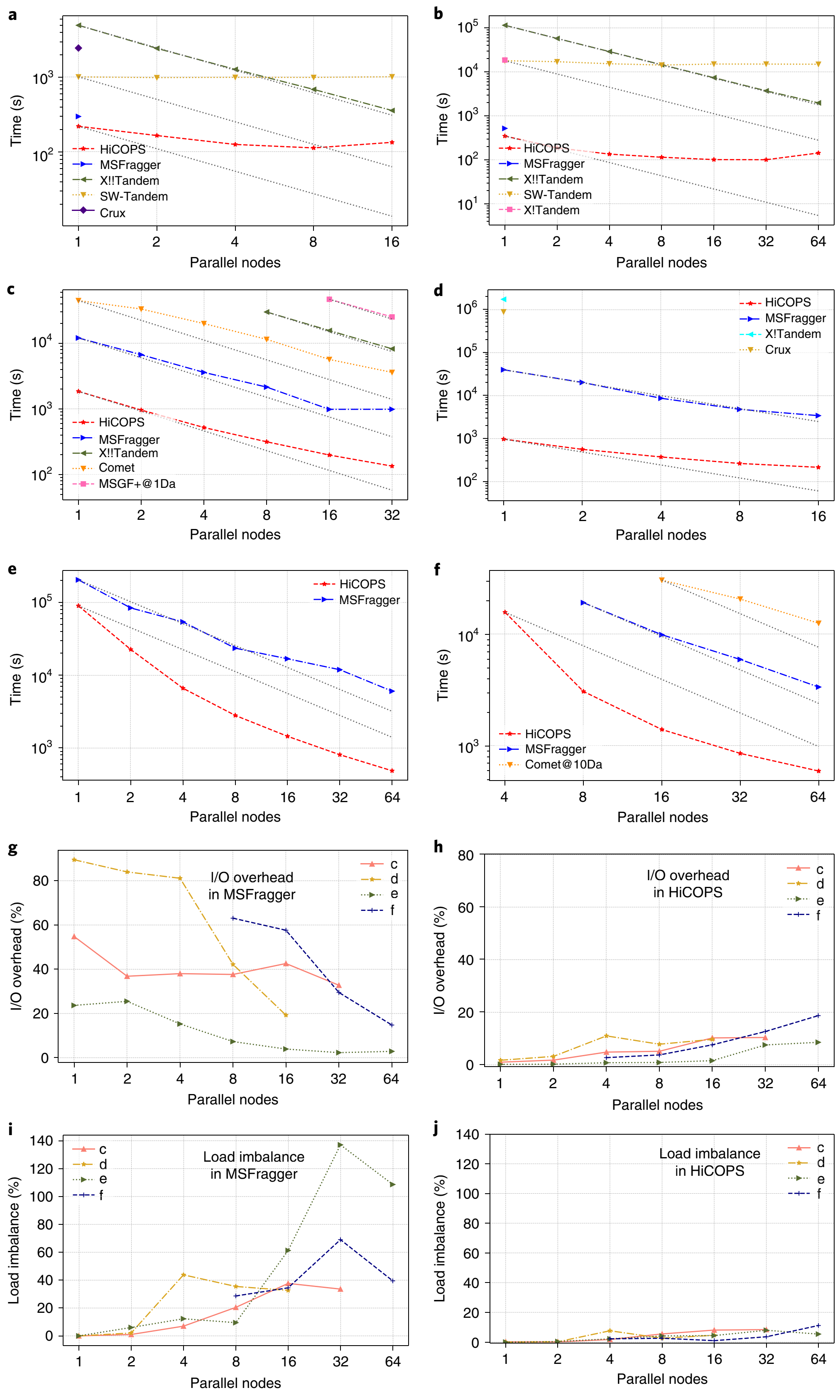

Fig. 3 | Speed comparisons. a–f, Speed comparison between HiCOPS and several other tools with increasing number of parallel nodes is shown for the six experiments described in section: Speed comparison against existing algorithms, respectively. The gray dotted line tracks the ideal speedup times for each tool (log–log scale) in experiments. The *δM* window for MSGF+ and Comet was further tightened in some experiments (indicated by @ labels) due to tool limitations. **g–i**, The percentage I/O and load imbalance overheads exhibited by HiCOPS and MSFragger for experiments in **c**, **d**, **e** and **f** are shown with increasing number of parallel nodes.

Table 1 | A summary of the execution times for three large-scale database search experiments using HiCOPS and MSFragger.

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	1,692	500	0.05	103.5
2	MSFragger	72	15	1,692	500	0.05	1,074.45
2*	MSFragger	1	15	1,692	500	0.05	51,130
3	HiCOPS	64	41	4,000	500	0.01	27.3

Peptide precursor mass tolerance and fragment-ion tolerance (in Daltons) are given as δM and δF respectively. A single-node version of the second experiment using MSFragger (that is 2*) was run on the local (raptor) server. The third experiment was not run using MSFragger due to feasibility issues.

spectra and δM is the peptide precursor mass tolerance setting in ± 100 Da. Note that the tuple does not contain the fragment-ion 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 of the experiments were run on the Extreme Science and Engineering Discovery Environment (XSEDE)[39] Comet cluster at the San Diego Supercomputer Center. The Comet compute nodes are equipped with 2 NUMA nodes × 64 GB of Intel Xeon E5-2680v3 processors (total: 24 cores), 2 NUMA nodes × 64 GB (total: 128 GB) DRAM, 56 Gbps FDR InfiniBand interconnect and a Lustre shared file system. The maximum number of nodes allowed per job is 72 and the maximum allowed job time is 48 hours. Furthermore, the single-node experiments for the Crux and X!Tandem tools requiring more than 48 hours (XSEDE limit) execution time were run on a (comparable) local machine named *raptor*, equipped with Intel Xeon Gold 6152 processor (22 cores), 128 GB DRAM and a 6 TB SSD HDD.

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 three decimal points). A comparison of hyperscores and expectscores between the serial (x-axis) and parallel runs (y-axis)—obtained by searching S_1 against D_1 with no PTMs—is shown in Fig. 2a,b. 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 and Fig. 1d).

In the second step, we verified the quality of the implemented search algorithm by comparing the HiCOPS-computed and MSFragger-computed hyperscores, as both frameworks employ a similar scoring algorithm, that is, shared-peak counting coupled hyperscore. Note that the hyperscores computed by MSFragger and HiCOPS cannot be exactly identical as MSFragger uses several preprocessing 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 ($\delta M = 1$ Da) and three with open-search ($\delta M \geq 100$ Da) settings. The experimental MS/MS data preprocessing and database search settings were kept identical (and as minimal as possible) for both tools for fair comparison.

In the first experiment, a subset of 860,000 spectra from S_4 was searched against D_1 modified with methionine oxidation and NQ-deamidation as PTMs yielding a theoretical database of 18 million spectra at $\delta M = 1$ Da. In the second experiment, S_3 was searched against D_1 modified with methionine oxidation and STY-phosphorylation yielding a theoretical database of 66 million spectra at $\delta M = 1$ Da. In the third experiment, S_3 was searched against D_2 modified with methionine oxidation and serine phosphorylation yielding a database of 80 million spectra at $\delta M = 1$ Da. In the fourth experiment: the entirety of S_3 was searched against D_1 modified with methionine oxidation and NQ-deamidation yielding a theoretical database of 18 million spectra at $\delta M = 200$ Da. In the fifth experiment, S_3 was searched against D_1 modified with methionine oxidation and ST-phosphorylation yielding a theoretical database of 56 million spectra at $\delta M = 100$ Da. In the sixth experiment, S_3 was searched against D_2 modified with methionine oxidation and serine phosphorylation yielding a database of 80 million spectra at $\delta M = 200$ 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 Fig. 2c–h). The PSMs from both tools were then filtered at a q-value (false discovery rate) of 1% and compared (shown in Supplementary Fig. 5). Fig. 2c–e depicts a strong-correlation ($R \geq 0.90$) 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 (Fig. 2f–h, respectively). We suspect that the divergence in hyperscores may have stemmed from open-search specific spectral processing, reconstruction and/or score reranking algorithms implemented in MSFragger. Furthermore, the results in Supplementary Fig. 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, that is, the underlying algorithms can be customized or ported with more sophisticated versions to improve the identification while delivering similar performance.

Speed comparison against existing algorithms. We compared the HiCOPS speed against many existing shared- and distributed-memory database peptide search algorithms including Tide/Crux v.3.2 (ref. [3]), Comet v.2020.01 (ref. [40]), MSFragger v.3.0 (ref. [2]), X!Tandem v.17.2.1 (ref. [41]), X!!Tandem v.10.12.1 (ref. [26]) and SW-Tandem (ref. [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). We also tried to run the UltraQuant HPC tool, which implements a parallel MaxQuant; however, it crashed with unhandled exceptions every time it was run on more than one node.

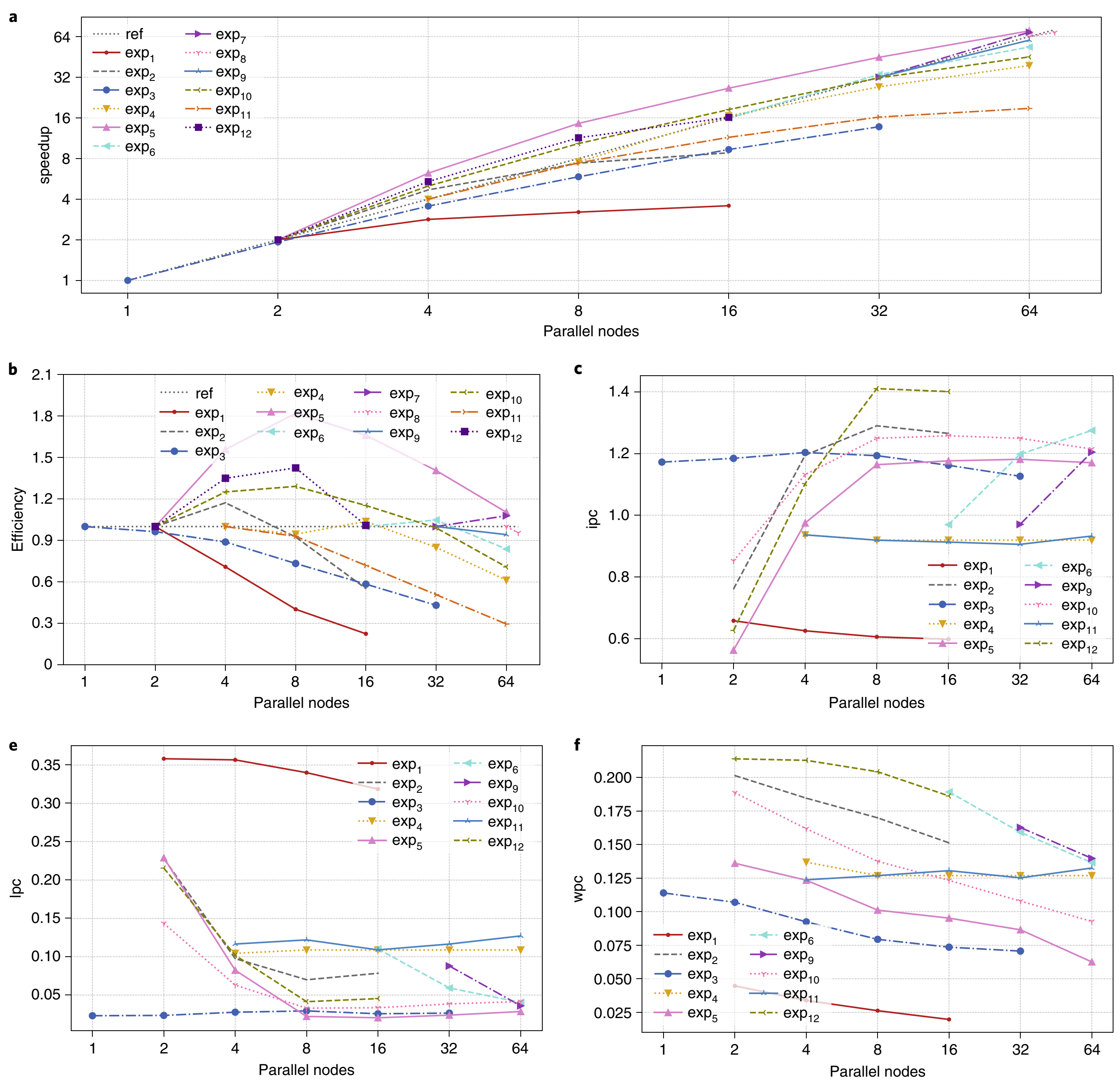

Fig. 4 | Performance Metrics. a–e, Performance metrics including parallel speedup (**a**), strong-scale efficiency (**b**), instructions per cycle (ipc) (**c**), last-level cache misses per total cache-level misses (lpc) (**d**) and writes per total stalled cycles (wps) (**e**) are shown with increasing 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.

We designed six experiments, which are listed as a–f in increasing order of their experimental workload sizes (that is, database and dataset sizes, and experimental settings). In the first two experiments (a, b), a subset of 8,000 spectra from S_3 (file: 7Sep18_Olson_WT24) was searched against D_2 modified with variable methionine oxidation and tyrosine biotin-tyramide yielding a theoretical database of 93.5 million spectra at $\delta M = 10$ Da and $\delta M = 500$ Da, respectively. In the third experiment (c), S_3 was searched against D_1 modified with variable methionine oxidation and tyrosine biotin-tyramide yielding a theoretical database of 7.1 million spectra at $\delta M = 500$ Da. In the fourth (d) and fifth (e) experiments, the entire S_3 was searched against the theoretical database of experiments a and b (the database with 93.5 million spectra) at $\delta M = 10$ Da and $\delta M = 500$ Da, respectively. In the sixth (f) experiment, S_4 was searched against D_1 modified with variable methionine oxidation, STY-phosphorylation and NQ-deamidation yielding a theoretical database of 213 million spectra at $\delta M = 100$ Da. The slower tools such as Comet, MSGF+, Crux and X!Tandem variants were only run for smaller experiments due to XSEDE maximum job time limits.

The obtained wall time results (Fig. 3a–f) show that the HiCOPS outperforms all other tools by more than 10× on average in speed, especially for experiments with larger workloads (Fig. 3d–f). It can also be observed that the HiCOPS exhibits better strong-scale parallel efficiency than other tools as the experimental workload size

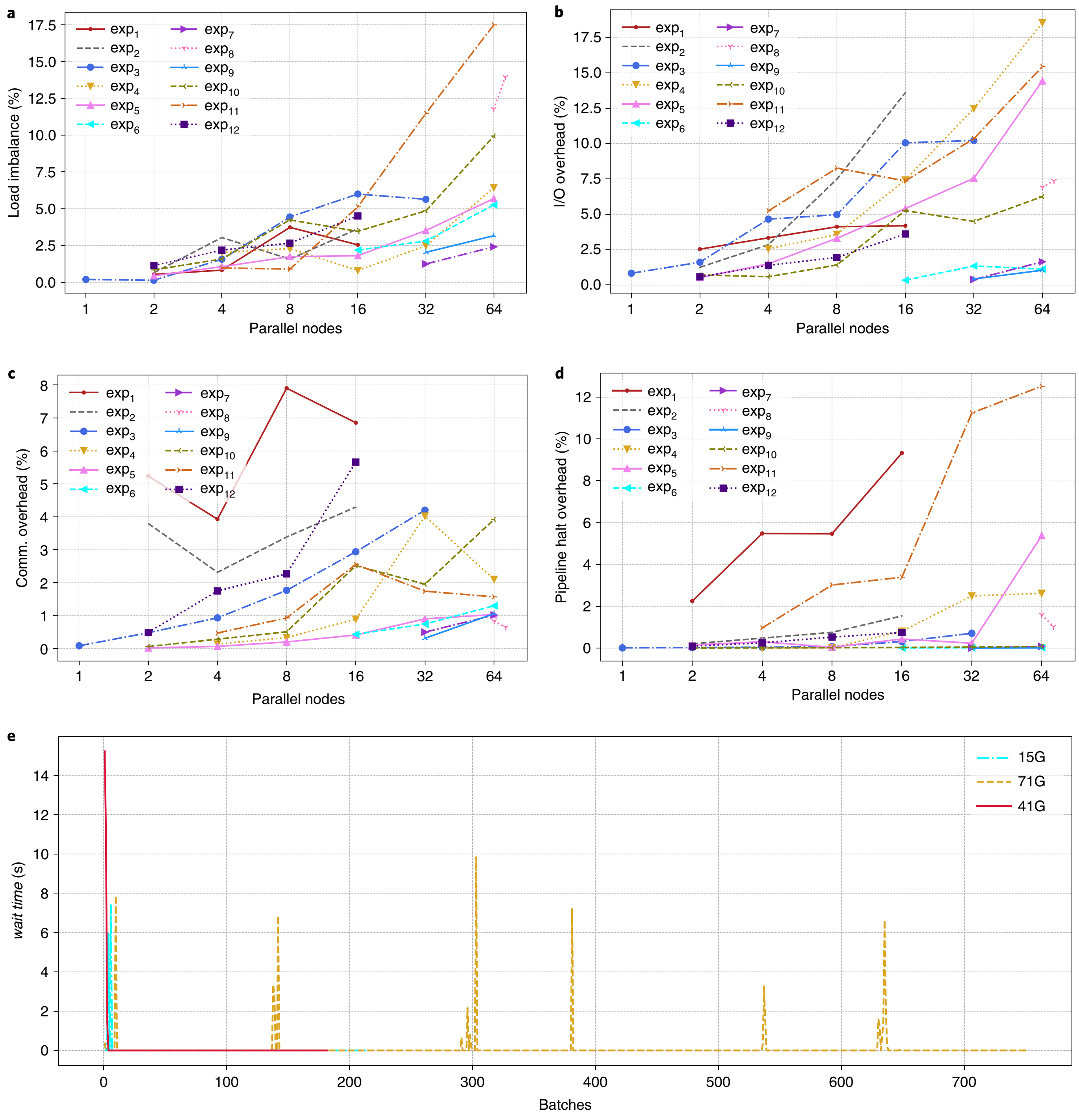

Fig. 5 | Overhead analysis. a–d, Overhead costs including load imbalance (**a**), I/O (**b**), communication (**c**) and pipeline halt time (**d**) are shown with increasing parallel nodes for all performance evaluation experiments (labeled as tuples: exp$_n$ in section: Performance evaluation). (**e**), The time-series shows the per-batch subtask pipeline halt time (scheduling performance) in superstep 3 when searching datasets of sizes 15 GB, 41 GB and 71 GB in open-search using 64 nodes. The *wait time* shows the time the pipeline subtasks in superstep 3 waited for corresponding data batches.

increases (a → f). For smaller workloads (Fig. 3a–c), the parallel efficiency is limited by the Amdahl's law. The scalability is shown as the deviation (positive is sublinear whereas negative is hyperlinear) from the ideal speedup track (dotted gray) lines in each experiment in Fig. 3a–f. The parallel efficiency results for MSFragger were particularly peculiar as it appears to be scaling superlinearly up to a certain number of parallel nodes and then dropping to sublinear. To explain this, the runtime components of MSFragger were further analyzed in detail. The results (Fig. 3g–i) show that a large percentage of MSFragger's runtime is composed of I/O and load imbalance, which results in a low overhead/compute ratio (effective resource utilization). Comparatively, HiCOPS exhibits substantially improved memory performance (Fig. 3h,j), resulting in lower runtime even though the effective search times (useful compute time) for MSFragger and HiCOPS are comparable. The results (Fig. 3a–c) show that the existing HPC tools—including X!!Tandem, SW-Tandem, parallel Comet and parallel MSGF+ (MS-PyCloud)—are >100× slower even for small-scale experiments. Finally, we observed zero parallel efficiency for SW-Tandem in all experiments, meaning, no speedups whatsoever (Supplementary Section 3).

Application in tera-scale experimentation. Application of HiCOPS in tera-scale experiments was demonstrated using three further experiments. In the first experiment, S_3 was searched against a theoretical database of 766 million spectra (780 GB) at $\delta M = \pm 500$ Da and $\delta F = \pm 0.01$ Da. In the second experiment, S_4 was searched against a theoretical database of 1.59 billion spectra (1.7 TB) at $\delta M = \pm 500$ Da and $\delta F = \pm 0.05$ Da. In the third experiment, S_2 was searched against a theoretical database of 3.89 billion spectra (4 TB) at $\delta M = \pm 500$ Da and $\delta F = \pm 0.01$ Da. HiCOPS completed these three experiments in 14.55 min (64 nodes), 103.5 min (72 nodes) and 27.3 min (64 nodes). By contrast, MSFragger completed the execution of first experiment in 158.8 min (64 nodes; ten-times slower). The second experiment was completed by MSFragger in 18 h (72 nodes; 10.3× slower) and 35.5 days when using one 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.

Performance evaluation. Twelve experiments of varying workload sizes were designed using combinations of aforementioned D_i and S_i, PTMs 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 $\delta F = \pm 0.01$ Da in all of these experiments.

Parallel scalability. Strong-scale efficiency for all twelve experiments was measured and the results (Fig. 4a,b) 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 Fig. 6 further confirm that the superstep 3 constitutes the largest fraction of the overall runtime, indicating its importance in optimizations. Note that the minimum number of nodes ($P_{\min}$) required by HiCOPS for each experiment must be: $P_{\min} \geq D/M$. The speedup and efficiency calculations were therefore performed using the runtime for the experiment with minimum nodes as the base case. The serial runtime (T_s) was first computed using the base case experiment runtime ($T_{P_{\min}}$) as $T_s = P_{\min} \times T_{P_{\min}}$. The speedups and efficiency were then computed relative to $T_{P_{\min}}$ for experiments with nodes $\geq P_{\min}$ using the computed T_s. Essentially, the speedups are relative to the base case runtime, which may not be the one-node time depending on the $P_{\min}$ (limitation of HiCOPS). Furthermore, superlinear 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, last-level cache misses per all cache-level misses, and the cycles stalled due to writes per total stalled cycles. The results (Fig. 4c–e) 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 as 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 (superlinear) in performance as wf/P reduces to the normal (optimal) range. For instance, the experiment set exp_5 depicts superlinear speedups (Fig. 4a) that can be correlated to the hardware performance surge in Fig. 4c.

Performance overhead. Several metrics including load imbalance, I/O, communication, and pipeline halt time costs were also measured to identify and quantify the performance overheads. The obtained results (Fig. 5a–c) depict that the load imbalance, I/O and intertask communication costs remain ≤10%, ≤10% and ≤5%, respectively, 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 the Methods) when searching three datasets of increasing size. The *wait time* is the time when any of the pipeline subtasks wait for a batch of data from its predecessor. The results (Fig. 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 subtasks keeping the total cost to ≤5% in most experiments (Fig. 5d).

Discussion

Recent trends in HPC have shifted towards heterogeneous architectures[42] as several top-500 supercomputers combine CPUs with GPUs and field-programmable gate arrays (FPGAs) to deliver petascale (and in the 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. Furthermore, 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; meaning, 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[9,44,45] 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 due to the limitations of the infrastructure that is at our disposal. We are therefore confident that our current and future efforts will provide useful advances in enabling scientific investigations in this application domain.

Methods

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

Runtime cost model. As 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 $p_i \epsilon P$ to execute a superstep j can be modeled by only its local input size (database and dataset sizes) and available resources (the number of cores, memory bandwidth). The parallel processes may execute the algorithmic work in a data-parallel, task-parallel, or hybrid fashion. As an example, the execution runtime (cost) for 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 a data-parallel fashion (using all c_{p_i} cores) will be given as follows:

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

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 subtask parallel fashion with $c_{p_i}/2$ cores each, the execution time will be given as

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

For analysis purposes, if the time complexity of the algorithms used for step k_{jz} is known (for example, $O(.)$), we will convert it into a linear function k' 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. If, for example, it is known that the sorting algorithms used for k_{j2} have time complexity: $O(N\log N)$, then 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.

Superstep 1: database partitioning. In this superstep, the HiCOPS parallel processes construct a local database partition through the following three algorithmic data-parallel steps (Fig. 1a): (1) generate and extract a (balanced) local partition of the (peptides + PTM variants) database; (2) generate the theoretical spectra data; and (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 Fig. 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 the Mod Distance (Δm), which allows better separation of database spectral-pairs that 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. 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), Δm is given as follows:

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

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 refs. [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 $k'(D_{p_i}\log D_{p_i})$. Collective runtime for this superstep is given by equation (5):

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

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

Superstep 2: experimental MS/MS data preprocessing. In this superstep, the HiCOPS parallel processes preprocess a partition of experimental MS/MS spectra data through the following three algorithmic data-parallel steps (Fig. 1b): (1) read the dataset files, create a batch index and initialize internal structures; (2) preprocess (normalize, clear noise, reconstruct and so on) a partition of experimental MS/MS data; and (3) write back the preprocessed 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 10,000 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 preprocessed experimental data in the superstep 3.

Cost analysis. The first step reads the entire dataset (size = $q\beta$) and creates a batch index in runtime: $k_{21}(q\beta)$. The second step may preprocess a partition of the dataset (of roughly $q\beta/P = Q_{p_i}$ in size) using a data preprocessing algorithm such as in refs. [5,44,47] in runtime $k_{22}(Q_{p_i})$. The third step may write the preprocessed 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 preprocessed. Collective runtime for this superstep is given by equation (6):

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

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

Superstep 3: database peptide search. This is the most important superstep in the HiCOPS workflow and is responsible for 80–90% of the total algorithmic workload. In this superstep, the HiCOPS parallel processes search the preprocessed experimental spectra against their local database partitions through the following three hybrid (task and data parallel) steps (Fig. 1c and Supplementary Fig. 4): (1) load the preprocessed experimental MS/MS data batches into memory; (2) search the loaded spectra batches against the (local) database partition and produce intermediate results; and (3) serialize and write the intermediate results to the shared file system assigning them unique tags.

Three parallel subtasks are created (R, I and K) that work in a producer–consumer pipeline to execute the algorithmic work (Fig. 1c). The data flow between the subtasks is handled through queues to create a buffer between the producers and consumers. The first subtask (R) loads batches of the preprocessed experimental spectra data and puts them in queue (q_f) as depicted in Supplementary Algorithm 2. Subtask R may also perform minimal computations on the experimental spectra before putting them in queue. For example, peak selection and/or intensity normalization. The parallel cores assigned to R are given by: $|r|$. The second subtask (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 I are given by: $|i|$; I also recycles the memory buffers back to R using the queue (q_r). The third subtask (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. Subtask R reads the experimental data batches in runtime $k_{30}(q\beta)$; Subtask I iteratively filters the database and computes spectral comparisons against the database (scoring step). 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[9,10]. In current implementation, we use the first two filtration methods, which execute in runtimes $k_{31}(qD_{p_i})$ and $k_{32}(q\beta\alpha_{p_i})$, respectively. Here, 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, 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, subtask 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. We therefore capture them using an extra runtime cost $V_{p_i}(q, D_{p_i}, P)$. Several optimizations including buffering, task scheduling, load balancing and data sampling (discussed in section: Optimizations) were implemented to alleviate these overhead costs. Collective runtime for this superstep is given by equation (10).

The runtime of R, $t_{p_i}(r, |r|)$, is given as

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

The runtime of I, $t_{p_i}(i, |i|)$, 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

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

The runtime of K, $t_{p_i}(k, |k|)$, is given as

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

Combining equations (7–9), we have

$$\begin{aligned} T_3 = \max_{p_i} \big(&\max(t_{p_i}(r, |r|), t_{p_i}(i, |i|), t_{p_i}(k, |k|))+ \\ &V_{p_i}(q, D_{p_i}, P) + \gamma_{p_i}\big) \end{aligned} \tag{10}$$

Remarks. Equations (7–10) depict that the parallel runtime portion of this superstep grows quadratically superseding the serial portion if the experimental load is sufficient.

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 algorithmic steps (Fig. 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; and (3) write the complete results to the file system.

Two parallel subtasks are created to execute the algorithmic steps in this superstep. The first subtask reads sets of intermediate results from the shared file system (or shared memory) (satisfying: $\text{tag} \bmod P = p_i; p_i \epsilon$ MPI ranks), deserializes 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 $i = 0, 1, \ldots, P-1$. The assembly process is done through signal addition and shift operations (Fig. 1d). The expected values 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 the log-Weibull (Gumbel) Fit method (Fig. 1d). The computed expected values, 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 subtask. Supplementary Algorithm 4 depicts the algorithmic work performed by this subtask.

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

Cost analysis. The first subtask 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 subtask 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).

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

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

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

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

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

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.

Performance analysis. To quantify the parallel performance, we decompose the HiCOPS's time (T_{H}) (equation (1)) into three runtime components: parallel runtime (T_{p}), serial runtime (T_{s}) and overheads runtime (T_{o}) given as:

$$T_{\text{H}} = \sum_{j=1}^{4} \max_{p_i}(T_{j,p_i}) = T_{\text{o}} + T_{\text{s}} + T_{\text{p}} \tag{13}$$

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

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

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

and

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

T_{s} is the minimum serial time required for HiCOPS execution and cannot be further reduced. We will therefore focus on optimizing the remaining runtime (T_F) given as: $T_F = T_{\text{p}} + T_{\text{o}}$. Using equations (14) and (16), we have

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

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

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

Recall that the superstep 4 runtime is optimized for maximum parallelism (and least interprocess communication) and that the superstep 3 performs the largest fraction of overall algorithmic workload. We can thus also remove the superstep 4 terms from T_F to simplify analysis:

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

Furthermore, as the superstep 3 is executed in a producer–consumer pipeline (Fig. 1c) where 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

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

In the above equation we can rewrite the max(.) term as the time to complete subtask I: ($t_{p_i}(i, |i|)$) plus the extra time to complete subtask K (the last consumer): $t_x(k)$. Using equation (9) we have:

$$\begin{aligned} T_F = &k'_{31}(q\log(D_{p_i}), |i|) + k'_{32}(q\beta\log(\alpha_{p_i}), |i|)+ \\ &k_{33}(q\beta\sigma_{p_i}, |i|) + k_{34}(q\mu_{p_i}, |i|) + t_x(k) + T_{\text{o}} \end{aligned} \tag{19}$$

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

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

Optimizations. The following sections discuss the optimization techniques employed to alleviate the overhead costs in equation (20).

Buffering. Four queues—the forward queue (q_f), recycle queue (q_r) and result queues (q_k and q'_k)—are initialized and routed between the producer–consumer subtasks in superstep 3 (Fig. 1c) as $R \rightarrow I$, $R \leftarrow I$, $I \rightarrow K$ and $I \leftarrow K$, respectively; q_r is initialized with (default: 20) empty buffers for subtask R to fill the preprocessed experimental data batches and push in q_f. Subtask 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 subtask K and pushed back to q'_k for re-use. Three regions are defined for q_f on the basis of the number of data buffers it contains at any time. i.e. w_1: $(\text{len}(q_f) < 5)$; w_2: $(5 \leq \text{len}(q_f) < 15)$; and w_3: $(\text{len}(q_f) \geq 15)$. These regions ($w_l$) are used by the task-scheduling algorithm discussed in the following section.

Task scheduling. The task-scheduling algorithm is used to maintain a synergy between the producer–consumer (subtask) 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, two threads are assigned to the subtasks R and K, whereas the remaining $c_{p_i} - 2$ threads are assigned to subtask 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} (that is, t_{fct}) using double exponential smoothing[49]; t_{wait} is also accumulated into t_{cum}. Two thresholds are defined: the minimum wait ($t_{\min}$) and maximum cumulative wait ($t_{\max}$). Using all this information, a thread is removed from I and added to R if the following conditions are satisfied:

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

t_{cum} is set to 0 every time a thread is added to R. Similarly, a thread is removed from 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.

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

K uses its two oversubscribed threads to perform the overlapped I/O operations concurrently (Fig. 1c).

Load balancing. The algorithmic workload in equation (20) is given by $k_{33}(\text{q}\beta\sigma_{p_i}, |i|) + k_{34}(\text{q}\mu_{p_i}, |i|)$. Here, the terms $q\beta$ and q are constants (experimental data size), whereas the terms σ_{p_i} and μ_{p_i} are variables representing 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, Fig. 1a and Supplementary Fig. 6). The correctness of the LBE algorithm for load balancing is proven in Supplementary Section 6. We plan to devise and develop dynamic load balancing techniques for better results in the future.

Sampling. Sampling is used to reduce the interprocess 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 of the number of local hits, the hyperscore for the top hits and so on (12 bytes), and the local null distribution histogram of hyperscores (2,048 bytes). Communicating this, the size of each data packet (1 per batch) will be ~20 MB, which can result in serious overheads. It has been shown that the null distribution of 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 the most intense non-zero data points around it. If the total number of non-zero samples exceed s (default: 120), 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 1.5 MB 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 Fig. 7 illustrates an example of sampling.

Detailed experimental setup. The two databases (D_1 and D_2) were digested in silico using trypsin as enzyme (fully tryptic) with two allowed missed cleavages, peptide lengths between 6 and 46, and peptide masses between 500 Da and 5,000 Da. The theoretical spectra were simulated by generating b- and y-ions of 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- ($\delta M \approx \pm 500$ Da) and closed-search ($\delta M \leq \pm 10$ Da) scenarios. The closed-search criterion was set to a few Daltons (≤ 1 Da in correctness analysis and ≤ 10 Da in performance evaluation) instead of 10–20 ppm to cover the differences in calculated peptide precursor masses due to monoisotopic or average masses, and isotopic masses across search tools. All experimental MS/MS datasets were converted to MS2 format before use. The experimental MS/MS spectra preprocessing 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 5,000 Da; precursor charges, +1 to +4; minimum number of matched peaks for PSM candidacy, 4; minimum number of database hits for statistical scoring, 4; denoising, only top-100 peaks picked (by intensity); peak transformations, none; mass calibration, no; precursor peak removal, no; partial spectrum reconstruction, no; clip n-term M, no.

Data availability

All of the datasets used in this study are publicly available from PXD 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 S_1 PXD009072, use https://www.ebi.ac.uk/pride/archive/projects/PXD009072). The *Homo sapiens* protein sequence database can be downloaded from UniProtKB via https://www.uniprot.org/proteomes/UP000005640. The UniProt SwissProt (reviewed) database can be downloaded via https://www.uniprot.org/uniprot/?query=reviewed:yes. Source data are provided with this paper.

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 and so on will be updated on the same web page. The source code is available open-source at https://doi.org/10.5281/zenodo.5094072 (ref. [50]) and https://github.com/hicops/hicops. Please refer to https://hicops.github.io for detailed documentation, licensing and future software updates.

Received: 2 December 2020; Accepted: 16 July 2021;
Published online: 20 August 2021

References

1. Nesvizhskii, A. I. A survey of computational methods and error rate estimation procedures for peptide and protein identification in shotgun proteomics. *J. Proteomics* **73**, 2092–2123 (2010).
2. Kong, A. T., Leprevost, F. V., Avtonomov, D. M., Mellacheruvu, D. & Nesvizhskii, A. I. MSfragger: ultrafast and comprehensive peptide identification in mass spectrometry-based proteomics. *Nat. Methods* **14**, 513 (2017).
3. McIlwain, S. et al. Crux: rapid open source protein tandem mass spectrometry analysis. *J. Proteome Res.* **13**, 4488–4491 (2014).
4. Yuan, Z.-Fe et al. pParse: a method for accurate determination of monoisotopic peaks in high-resolution mass spectra. *Proteomics* **12**, 226–235 (2012).
5. Deng, Y. et al. pClean: an algorithm to preprocess high-resolution tandem mass spectra for database searching. *J. Proteome Res.* **18**, 3235–3244 (2019).
6. Degroeve, S. & Martens, L. Ms2pip: a tool for ms/ms peak intensity prediction. *Bioinformatics* **29**, 3199–3203 (2013).
7. Zhou, X.-X. et al. pDeep: predicting MS/MS spectra of peptides with deep learning. *Anal. Chem.* **89**, 12690–12697 (2017).
8. Zhang, J. et al. PEAKS DB: de novo sequencing assisted database search for sensitive and accurate peptide identification. *Mol. Cell. Proteomics* **11**, M111–010587 (2012).
9. Devabhaktuni, A. et al. TagGraph reveals vast protein modification landscapes from large tandem mass spectrometry datasets. *Nat. Biotechnol.* **1**, 469–479 (2019).
10. Chi, H. et al. Comprehensive identification of peptides in tandem mass spectra using an efficient open search engine. *Nat. Biotechnol.* **36**, 1059–1061 (2018).
11. Bern, M., Cai, Y. & Goldberg, D. Lookup peaks: a hybrid of de novo sequencing and database search for protein identification by tandem mass spectrometry. *Anal. Chem.* **79**, 1393–1400 (2007).
12. Eng, J. K., McCormack, A. L. & Yates, J. R. An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. *J. Am. Soc. Mass Spec.* **5**, 976–989 (1994).
13. Craig, R. & Beavis, R. C. A method for reducing the time required to match protein sequences with tandem mass spectra. *Rapid Commun. Mass Spec.* **17**, 2310–2316 (2003).
14. Diament, B. J. & Noble, W. S. Faster sequest searching for peptide identification from tandem mass spectra. *J. Proteome Res.* **10**, 3871–3879 (2011).
15. Eng, J. K., Fischer, B., Grossmann, J. & MacCoss, M. J. A fast sequest cross correlation algorithm. *J. Proteome Res.* **7**, 4598–4602 (2008).
16. Park, C. Y., Klammer, A. A., Kall, L., MacCoss, M. J. & Noble, W. S. Rapid and accurate peptide identification from tandem mass spectra. *J. Proteome Res.* **7**, 3022–3027 (2008).
17. Geer, L. Y. et al. Open mass spectrometry search algorithm. *J. Proteome Res.* **3**, 958–964 (2004).
18. Hebert, A. S. et al. The one hour yeast proteome. *Mol. Cell. Proteomics* **13**, 339–347 (2014).
19. Nesvizhskii, A. I. et al. 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. *Mol. Cell. Proteomics* **5**, 652–670 (2006).
20. Eng, J. K., Searle, B. C., Clauser, K. R. & Tabb, D. L. A face in the crowd: recognizing peptides through database search. *Mol. Cell. Proteomics* **10**, R111.009522 (2011).

21. Haseeb, M. & Saeed, F. 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)* 275–278 (IEEE, 2019).
22. Williams, S., Waterman, A. & Patterson, D. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* **52**, 65–76 (2009).
23. Chi, H. et al. pFIND–Alioth: a novel unrestricted database search algorithm to improve the interpretation of high-resolution MS/MS data. *J. Proteomics* **125**, 89–97 (2015).
24. Marx, V. The big challenges of big data. *Nature* **498**, 255–260 (2013).
25. Duncan, D. T., Craig, R. & Link, A. J. Parallel tandem: a program for parallel processing of tandem mass spectra using PVM or MPI and X! tandem. *J. Proteome Res.* **4**, 1842–1847 (2005).
26. Bjornson, R. D. et al. X!!Tandem, an improved method for running X!Tandem in parallel on collections of commodity computers. *J. Proteome Res.* **7**, 293–299 (2007).
27. Pratt, B., Howbert, J. J., Tasman, N. I. & Nilsson, E. J. MR-tandem: parallel X! Tandem using Hadoop MapReduce on Amazon Web Services. *Bioinformatics* **28**, 136–137 (2011).
28. Li, C., Li, K., Li, K. & Lin, F. MCtandem: an efficient tool for large-scale peptide identification on many integrated core (MIC) architecture. *BMC Bioinformatics* **20**, 397 (2019).
29. Li, C., Li, K., Chen, T., Zhu, Y. & He, Q. SW-Tandem: a highly efficient tool for large-scale peptide sequencing with parallel spectrum dot product on Sunway TaihuLight. *Bioinformatics* **35**, 3861–3863 (2019).
30. Chen, L. et al. MS-PyCloud: an open-source, cloud computing-based pipeline for LC-MS/MS data analysis. Preprint at https://www.biorxiv.org/content/10.1101/320887v1 (2018).
31. Prakash, A., Ahmad, S., Majumder, S., Jenkins, C. & Orsburn, B. Bolt: a new age peptide search engine for comprehensive MS/MS sequencing through vast protein databases in minutes. *J. Am. Soc. Mass Spec.* **30**, 2408–2418 (2019).
32. Kaiser, P. et al. High-resolution community analysis of deep-sea copepods using maldi-tof protein fingerprinting. *Deep Sea Res. I* **138**, 122–130 (2018).
33. Rossel, S. & Arbizu, P. M. Revealing higher than expected diversity of Harpacticoida (Crustacea: Copepoda) in the North Sea using MALDI-TOF MS and molecular barcoding. *Sci. Rep.* **9**, 1–14 (2019).
34. Yates III, J. R. Proteomics of communities: metaproteomics. *J. Proteome Res.* **18**, 2359 (2019).
35. Saeed, F., Haseeb, M. & Lyengar, S. S. Communication lower-bounds for distributed-memory computations for mass spectrometry based omics data. Preprint at https://arxiv.org/abs/2009.14123v2 (2021).
36. Beyter, D., Lin, M. S., Yu, Y., Pieper, R. & Bafna, V. Proteostorm: an ultrafast metaproteomics database search framework. *Cell Syst.* **7**, 463–467 (2018).
37. Valiant, L. G. A bridging model for parallel computation. *Commun. ACM* **33**, 103–111 (1990).
38. Tiskin, A. *BSP (Bulk Synchronous Parallelism)* 192–199 (Springer, 2011); https://doi.org/10.1007/978-0-387-09766-4_311
39. Towns, J. et al. XSEDE: accelerating scientific discovery. *Comput. Sci. Eng.* **16**, 62–74 (2014).
40. Eng, J. K., Jahan, T. A. & Hoopmann, M. R. Comet: an open-source MS/MS sequence database search tool. *Proteomics* **13**, 22–24 (2013).
41. Craig, R. & Beavis, R. C. Tandem: matching proteins with tandem mass spectra. *Bioinformatics* **20**, 1466–1467 (2004).
42. Madsen, J. R. et al. Timemory: modular performance analysis for HPC. In *International Conference on High Performance Computing* 434–452 (Springer, 2020).
43. Stevens, R., Ramprakash, J., Messina, P., Papka, M. & Riley, K. *Aurora: Argonne's Next-Generation Exascale Supercomputer* Technical Report (Argonne National Laboratory, 2019).
44. Liu, K., Li, S., Wang, L., Ye, Y. & Tang, H. Full-spectrum prediction of peptides tandem mass spectra using deep neural network. *Analytical chemistry* **92**, 4275–4283 (2020).
45. Lin, Y.-M., Chen, C.-T. & Chang, J.-M. MS2CNN: predicting MS/MS spectrum based on protein sequence using deep convolutional neural networks. *BMC Genomics* **20**, 1–10 (2019).
46. Haseeb, M., Afzali, F. & Saeed, F. 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)* 191–198 (IEEE, 2019).
47. Ding, J., Shi, J., Poirier, G. G. & Wu, F.-X. A novel approach to denoising ion trap tandem mass spectra. *Proteome Sci.* **7**, 9 (2009).
48. Fenyö, D. & Beavis, R. C. A method for assessing the statistical significance of mass spectrometry-based protein identifications using general scoring schemes. *Anal. Chem.* **75**, 768–774 (2003).
49. LaViola, J. J. Double exponential smoothing: an alternative to kalman filter-based predictive tracking. In *Proc. Workshop on Virtual Environments 2003* 199–206 (The Eurographics Association, 2003).
50. Haseeb, M. & Saeed, F. *hicops/hicops: HiCOPS v1.0.0—1st Public Release* (Zenodo, 2021); https://doi.org/10.5281/zenodo.5094072
51. Haseeb, M. & Saeed, F. *Source Data: High Performance Computing Framework for Tera-Scale Database Search of Mass Spectrometry Data* (Zenodo, 2021); https://doi.org/10.5281/zenodo.5076575

Acknowledgements
This work used the National Science Foundation (NSF) XSEDE supercomputers through allocations TG-CCR150017 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 NSF under 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 NIH and/or the NSF.

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.

Additional information
Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s43588-021-00113-z.

Correspondence and requests for materials should be addressed to M.H. or F.S.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Reviewer recognition statement *Nature Computational Science* thanks Robert Bjornson, Benjamin Neely, Yasset Perez-Riverol and the other, anonymous, reviewer(s) for their contribution to the peer review of this work.

Editor recognition statement Handling editor: Ananya Rastogi, in collaboration with the *Nature Computational Science* team.