# BMF: Bitmapped Mass Fingerprinting for Fast Protein Identification

Weikuan Yu* K. John Wu† Wei-Shinn Ku* Cong Xu* Juan Gao*

Auburn University* Lawrence Berkeley Lab†
{wkyu,weishinn,czx0003,jgao}@auburn.edu {kwu}@lbl.gov

***Abstract*—Protein identification is an important objective for proteomic and medical sciences as well as for pharmaceutical industry. With recent large-scale automation of genome sequencing and the explosion of protein databases, it is important to exploit latest data processing technologies and design highly scalable algorithms to speed up protein identification. In this study, we design, implement, and evaluate a new software tool, *Bitmapped Mass Fingerprinting (BMF)*, that can efficiently construct a bitmap index for short peptides, and quickly identify candidate proteins from leading protein databases. BMF is developed by integrating the FastBit indexing technology and the popular Message Passing Interface (MPI) for parallelization. By exploiting FastBit for peptide mass fingerprinting across protein boundaries, we are able to accomplish parallel computation and I/O for a scalable implementation of protein identification. Our experimental results show that BMF brings dramatic performance improvement for protein identification from various protein databases. In particular, we demonstrate that BMF can effectively scale up to 8,192 cores on the Jaguar Supercomputer at Oak Ridge National Laboratory, achieving superb performance in identifying proteins from the National Center for Biotechnology Information (NCBI) non-redundant (NR) protein database.**

***Keywords*- Protein Identification; Peptide Mass Fingerprinting; Protein Databases; FastBit; Cray XT5.**

## I. Introduction

Fast protein identification is an important technique for scientific research in biology, medical sciences, and chemistry. It is also one of the premier objectives in pharmaceutics and proteomics. It dates back to the early 1980s when the molecular weights of peptides and proteins were first determined by atom bombardment [1]. The most popular experimental technique for protein identification is tandem mass spectrometry (MS/MS) [2]. It breaks proteins at different peptide bonds, and produces ionized fragments with different molecular weights (*mass*) and different number of charged ions (*charge*). These fragments are then visualized in a graphic tool called tandem mass spectrum or MS/MS spectrum, based on their mass/charge ratios.

Various genome projects and proteomic experiments are quickly generating a vast amount of genomic and proteomic data. Experimental techniques such as MS/MS are generally too labor-intensive and time-consuming to meet the growing needs of scientists for accessing, analyzing, and identifying their targeted proteins from these extremely large protein databases. High-throughput, automated protein identification systems need to be designed with algorithms that can access the databases in a pipeline fashion with efficiency and accuracy.

Protein identification through database searching is one of the most popular computational approaches [3], [4]. Matching peptides are identified from protein databases using a scoring function, based on the probability that a candidate peptide possesses the observed spectrum. While protein databases are quickly expanding, protein identification algorithms and tools face the challenge of meeting the demands of these genomic and protein databases. It would be beneficial if powerful supercomputers can be leveraged to expedite the process of protein identification.

Peptide Mass Fingerprinting (PMF) is a widely adopted technique for protein identification and validation. It identifies an unknown protein by cleaving the candidate at certain peptide bonds into many small peptides, and measuring the masses and charges of these peptides through mass spectrometers. These known masses are then used to search for candidate peptide matches. Eventually the identity of the protein can be determined by the presence of unique peptides.

In this paper, we report a newly developed software, *Bitmapped Mass Fingerprinting (BMF)*, for fast protein identification from protein databases, ranging from individual organisms to the largest non-redundant (NR) protein database of the National Center for Biotechnology Information (NCBI) [5]. We first describe a basic algorithm for computational peptide mass fingerprinting. Then we develop a new technique that employs the highly efficient FastBit indexing technology to build a searchable database from precomputed molecular weights of all candidate peptides, thereby enabling very efficient PMF. By exploiting parallel computation and I/O, we design BMF as a software architecture for fast protein identification. Furthermore, we perform a comprehensive evaluation of BMF using both a small-size local cluster and the Jaguar supercomputer at Oak Ridge National Laboratory.

We conduct extensive experiments for testing BMF. Our experimental results show that BMF brings dramatic performance improvement in searching for proteins from various databases. While BMF does have a costly initial phase to pre-compute the molecular weights of peptides and build fast searchable indices, we discover that the cost of index generation can be brought down to a very low level through parallel computation and I/O. In particular, we demonstrate that BMF is able to scale up to 8,192 cores on Jaguar for excellent

performance in index creation and protein identification from the NR protein database [5].

In summary, our contributions in this research are twofold:

- We design and develop a parallel tool, BMF, for peptide mass fingerprinting using the highly efficient FastBit indexing technology and the popular Message Passing Interface.
- We leverage parallelized computation and I/O for fast protein identification. By bridging the power of FastBit indexing and that of parallel processing, BMF enables efficient PMF portably on any parallel computing environment.
- We document the use of a powerful supercomputer for efficient peptide mass fingerprinting and and evaluate the performance of BMF in identifying proteins from leading protein databases. The benefits of BMF have been demonstrated across 8,192 cores on Cray XT5.

The rest of the paper is organized as follows. In the next section, an overview is provided on FastBit Indexing. Then we describe the design of BMF in Section III. Experimental results are provided in Sections IV. Related work on protein identification and mass spectrometry is presented in Section V. We conclude the paper in Section VI.

## II. FastBit Indexing Overview

FastBit is an efficient searching tool with a proven record of uses in scientific applications [6]. Its core technology utilizes a set of compressed bitmap indexes. FastBit incorporates several techniques to achieve a very fast searching capability. It employs a bitmap compression method called Word-Aligned Hybrid (WAH) code that performs bitwise logical operations significantly faster than other similar approaches [7]; it implements a set of multi-level bitmap indexes that not only reduce the number of bitmaps needed to answer a query but also reduce the number of bytes it accesses [8]; and it contains a set of advanced binning techniques that reduces the index sizes but at the same time keeps the query response time low [9], [10]. In tests against commercial database management systems, FastBit is shown to answer queries more than 10-50 times faster on a set of realistic scientific data collections [11]. The effectiveness of FastBit is not only demonstrated through timing measurements, but also through a series of rigorous theoretical analyses [7], [8]. These analyses show that FastBit indexes possess the theoretical optimality of providing search time proportional to the number of elements qualified by the query.

A FastBit index can be used to resolve conjunctive or disjunctive range conditions such as "temperature $> 800$" and "density between 0.3 and 0.4." It produces a compressed bitmap where each row satisfying the specified conditions is marked 1, and otherwise 0. This allows the answers from different index searches to be combined efficiently by using logical operations over the compressed bitmaps [11]. This feature is particularly useful to search for records that are independent from each other, such as high-energy collision events or network scan-attack patterns [12], [13]. There are many cases where the records are related, for example, they might be defined on a set of regular mesh points. In such cases, the users might be more interested in finding coherent regions in space, such as flame fronts in combustion simulations [14], [15]. For such applications, FastBit compressed bitmaps are used directly by region-growing methods to efficiently generate the regions of interest. Recently, FastBit is used to compute histograms for multi-dimensional parallel coordinate displays of particles in a laser-based particle accelerator [16]. FastBit is an open source tool, and is used by many applications without any involvement of the FastBit developers. For example, Yahoo employs FastBit to query internal data about advertisements, a German bioinformatics company utilizes it to speedup molecular docking [17], and a Brazilian research group uses it in a geographical data warehousing system [18].

## III. Parallel Mass Fingerprinting through Bitmapped Index

In this section, we describe the design of Bitmapped Mass Fingerprinting (BMF), a parallel program for peptide mass fingerprinting and protein identification. We focus on the computational efficiency of protein identification given a peptide of known mass. First, a basic algorithm for computational mass fingerprinting is described. Then we discuss the use of FastBit for peptide mass fingerprinting. Finally, we examine and exploit various parallelisms in the FastBit-based BMF.

### A. Computational Peptide Mass Fingerprinting

In a simplistic form, proteins are basically represented as strings of single-letter codes. Each code denotes one of the twenty amino acids, and has its canonical molecular weight (mass). Peptides are short sequences of amino acids, i.e., substrings of proteins. A protein database contains many protein sequences, each of which is preceded with their names and other annotation information. We use the FASTA format for protein databases.

The goal of computational PMF is to search peptides that have expected lengths and a molecular weight between a targeted range. We can define it formally as follows.

**Protein Database**: It is a string $D$ of amino acids, separated by protein names at their boundaries. Every letter has an associated mass W, where $W \in \{W_i, \forall i \in [0,19]\}$, the set of molecular weights for twenty amino acids. $\mid D \mid$ represents the total number of amino acids in the database.

**Input Parameters**: Each query has four input parameters: two integers (K, N), and two floating numbers M and $\varepsilon$. K and N specify the range of peptide length from K to N. Approximately, there are $\mid D \mid *(N-K+1)$ of possible substrings with length from K to N. Typically, peptides of interest on a mass spectrum have up to 40-50 amino acids. Each peptide has a "mass" m, which is the combined molecular weight from its constituent amino acids. M is a targeted mass value, and $\varepsilon$ a precision parameter. A peptide p is a good match if it satisfies two conditions: its length $\mid p \mid \in [K,N]$, and its mass $m \in [M-\varepsilon, M+\varepsilon]$.

**Query Output**: The output of a query is a list of two-tuples $(loc, len)$ that specifies the matching peptides. The number *loc* indicates the starting position of a peptide (hence the embedding protein), and *len* its length.

Algorithm 1 shows a basic serial computational algorithm for identifying a list of peptides from a database.

**Algorithm 1** Basic Peptide Mass Fingerprinting

```
1: {Initiate peptides P_i where i ∈ [K,N]}
2: for all i = K to N do
3:    mass(P_i) ← {Sum of mass from D[0] to D[i-2]}
4: end for
5: for all j = 0 to | D−1 | do
6:    for all l = K to N do
7:       if (j+l) <| D | then
8:          mass(P_l) ← mass(P_l) + D[j+l−1]
9:          if mass(P_l) ∈ [M−ε, M+ε] then
10:            output (j,l)
11:         end if
12:         mass(P_l) ← mass(P_l) − D[j]
13:      end if
14:   end for
15: end for
```

### *B. PMF with FastBit*

While Algorithm 1 defines the computational PMF for protein identification, it is important to investigate efficient and scalable approaches. In view of the efficiency of FastBit and its practicality for scientific applications, we consider it a worthy attempt to design a FastBit-based PMF and evaluate its potential benefits.

**Construction of Weight Tables** – The core requirement of FastBit is to organize input data in a two dimensional table. To enable FastBit for peptide mass fingerprinting, we compute the molecular weights of all peptides with matching lengths, and then store them into a 2-D table. Figure 1 shows a 2-D weight table for a protein of length $d$. It contains the masses for peptides of length from $k$ to $n$. Molecular weights for peptides ending at position $i$ are stored as the $i$-th row; and those for peptides of length $j$ are stored in the $j$-th column. Thus, a value $W(i, j)$ in the table represents the mass of a peptide that ends at position $i$, and has a length of $j$.

**Demarcating Protein Boundaries** – In the FASTA format, the first line of a protein sequence typically starts with a ">", followed by a description of its name, origin, and other attributes, and finally the actual sequence of amino acids. A protein database composed of many proteins will have multiple FASTA sequences concatenated together. Figure 2 shows an example of two FASTA protein sequences taken from the Bacteria database as described in Section IV-A. To properly demarcate protein boundaries, we use $-1.0e6$ as an impossible mass for a peptide, a.k.a *INF*. In the weight table, the value INF will be stored for $W(i, j)$ if there is no peptide that has a length $l \in [K, N]$, and ends at position $i$. For example, when $i$

| | k | k+1 | --- | j | --- | n−1 | n |
|---|---|---|---|---|---|---|---|
| 0 | INF | INF | --- | INF | --- | INF | INF |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| k−1 | W(k−1,k) | INF | --- | INF | --- | INF | INF |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| n−1 | W(n−1,k) | W(n−1,k+1) | --- | W(n−1,j) | --- | W(n−1,n−1) | W(n−1,n) |
| i | W(i,k) | W(i,k+1) | --- | **W(i,j)** | --- | W(i, n−1) | W(i,n) |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| d−1 | W(d−1,k) | W(d−1,k+1) | --- | W(d−1,j) | --- | W(d−1,n−1) | W(d−1,n) |

Fig. 1: A 2-D Table Represents Peptide Masses for One Protein.

points at one of the first $(K-1)$ amino acids, or falls at the protein boundaries. Figure 1 shows that the first row of the 2-D weight table has all values as INF. The weight table for a protein database consists of many small 2-D weight tables that are concatenated together vertically, one for each protein. The size of a column will be proportional to the number of amino acids in a protein database. One column of the weight tables for a protein database is stored as a separate file. FastBit also requires an auxiliary file named "-part.txt" to be associated with a weight table. This file contains a description of data files for all columns.

```
>gi|24345355|gb|AAN53092.1|AE015452_5 hypothetical
protein TIGR00278
MAQTQSPLQWLATTLIRGYQIFISPILGPRCRFNPTCSHYAIEAIKVHGT
AKGCWFALKRILKCHPLHPGGSDPVPPKNDRCNK
>gi|24345356|gb|AAN53093.1|AE015452_6 ribonuclease
P protein
MTSYTFTRELRLLTPAQFKSVFSNPIKASSAEITLLAIPNSEQHPRLGLT
VAKRYVKRANQRNRIKRVIRDSFRLNQHNIPHLDIVVLVRNGVMEMENAE
LNKLIEKLWRKLSRRYNG
```

Fig. 2: Sample Protein Sequences in FASTA Format.

**Building FastBit Indices** – FastBit builds a bitmap index for each column that has been created. A mass query for matching peptides selects a number of bitmaps corresponding to weights between $M-\varepsilon$ and $M+\varepsilon$ and then performs bitwise OR operations on them to produce the answer. These bitmaps are located next to each other in the FastBit index file. When such an index file is used to answer a query, FastBit needs to perform two read operations: one to read the metadata located at the beginning of the index and the other to read the needed bitmaps. FastBit enables a number of different options for reading the indices. It may read the whole index file into memory, perform the sequential read operations using

the generic *read()* system call, or read data after creating a memory map for the file. We used the option of memory mapped files in BMF because it is typically observed as more efficient.

### C. Parallelizing the FastBit-Based Mass Query

Like many bioinformedical programs, FastBit-Based Mass Query can be implemented as an embarrassingly parallel program, particularly if one were to simply divide a large protein database into multiple small ones. However, partitioning the database into many small ones introduces additional processing steps. Instead of a simple division, we use a tiling approach for partitioning the 2-D table of peptide molecular weights.

**Parallel Computation** – To achieve great parallelization, we examine the 2-D weight table for protein databases, and the requirement of FastBit in creating indices from precomputed molecular weights. Instead of directly breaking up a database into smaller ones, it is more effective to apply a tiling approach and break a big weight table into many small 2-D blocks for parallelized mass fingerprinting. For example, one process is assigned with a block of the 2-D table, for which it identifies peptides on a smaller number of rows and columns. This tiling approach also avoids the data partitioning step, resulting in flexible output aggregation for parallelized mass query.

**Parallel I/O** – In addition to parallel computation, we investigate ways to improve the I/O performance in generating the 2-D weight table and the FastBit index files. We designed the system so that all partitioned blocks are organized into a separated directory. We do not distinguish between chunks from a big database and those from small databases. This is to allow flexible partitioning of data and computation. On parallel file systems such as Lustre, we control the striping pattern of newly generated files so that they can aggregate available bandwidth from all storage devices on systems such as Jaguar [19]. Furthermore, we implement I/O buffering in BMF to batch small read and write operations into large ones before accessing the file system. This buffering strategy avoids flurries of small reads and writes that can degrade the I/O performance of large parallel file systems [20], [21].

### D. BMF Software Architecture

With the aforementioned algorithms and techniques, we developed BMF, an integrated parallel program for scalable computational PMF.

Figure 3 shows the software architecture of BMF. To perform peptide mass fingerprinting, the computation for a protein database is divided among a number of MPI processes. Each process has an internal flex [22] based tokenizer, which scans the original database file from an offset based on the division of computation as described in Section III-C. The tokenizer filters out the protein annotations from a FASTA file, but keeps the protein boundaries intact. Based on the scanned input, each process then creates a 2-D weight table, and leverages FastBit to generate searchable bitmap indices. A query for matching peptides is also processed in parallel based on the tiling partition of the 2-D weight table and the FastBit index files. The query results are combined and returned as a sorted output file.

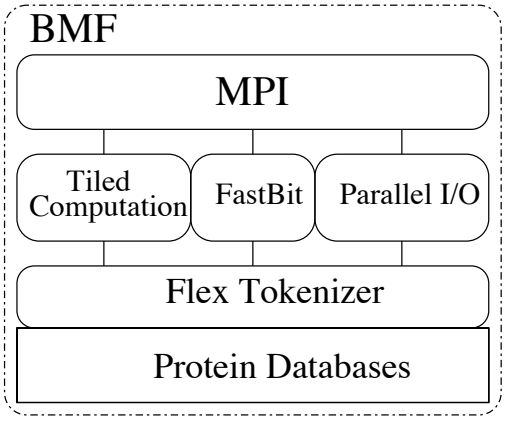


Fig. 3: Software Architecture of BMF.

## IV. Performance Evaluation

For small scale tests, we conducted our experiments on a cluster of 27 nodes with 2.1 GHz 64-bit dual-socket, quad-core Intel Harpertown processors. Each node is equipped with 8x PCI-Express Gen 2.0 bus. These nodes run Linux 2.6.18 kernels. They are equipped with a Gigabit Ethernet network. One node is configured as an NFS server with local attached disks. Six other nodes are configured as a small experimental Lustre file system with one Lustre metadata server (MDS) and five Lustre Object Storage Servers (OSS). This Lustre file system has 50GB storage space, aggregated from 10 GB disk partitions from the five storage devices.

For large scale experiments, we used the Jaguar supercomputer from Oak Ridge National Laboratory. Because Jaguar is a shared computing facility, we observed noticeable variations in the experimental results depending on the load of the system. Therefore, we include the error bars in the figures. For the results from the dedicated 27-node cluster, we omitted the error bars because variations were negligible.

### A. Protein Databases

To examine the performance of BMF, we used a set of protein databases, including those from individual organisms such as bacteria, yeast and drosophila, as well as large collections of protein databases such as SwissProt, PDB, and NR. All databases were downloaded from NCBI[1], unless specified otherwise.

**Bacteria**: This is a protein database, AE014299.faa, for the bacteria strain, *Shewanella oneidensis*. It contains 4,630 proteins with 1.3 million amino acids. The file size is 1.8 MB.

**Yeast**: This is a database that contains protein translations for the genome of yeast, an eukaryotic microorganism. There are 6,298 proteins with 2.9 million amino acids. This database

[1] ftp://ftp.ncbi.nih.gov/

is originally downloaded as a compressed file, yeast.aa.gz. The file size is 3.3 MB.

**Drosophila**: This is a database with protein translations for another eukaryotic organism, drosophila, commonly known as fruit fly. It consists of 14,331 proteins with 7.8 million amino acids. This database is originally downloaded as a compressed file, drosoph.aa.gz. The file size is 8.2 MB.

**PDB**: PDB is the Protein Data Bank [23], maintained by Brookhaven National Laboratory (Long Island, New York, USA). It contains all publicly available solved protein structures. There are 161,299 proteins with 38.1 million amino acids. This database is originally downloaded as a file pdb_seqres.txt from a Worldwide Protein Data Bank server[2]. The file size is 46 MB.

**SwissProt**: SwissProt [24] is a protein database maintained at the University of Geneva. It is a highly-curated, highly-cross referenced, non-redundant database. It contains 297,480 proteins with 115.6 million amino acids. The database is originally downloaded as a compressed file, swissprot.aa.gz. The file size is 139 MB.

**NR**: The NR protein database is maintained by NCBI [5] as a target for their BLAST [25] search services. It is a composite of multiple protein databases, including SwissProt and PDB, among others. Protein entries with absolutely identical sequences have been merged. There are 10,434,019,480 amino acids in 3.6 million proteins. This database is originally downloaded as a compressed file, nr.gz. The file size is 5.4 GB.

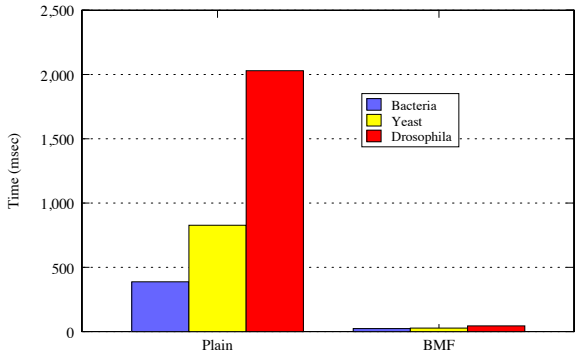



Fig. 4: Fast Mass Query using BMF.

### B. Benefits of FastBit

Using the databases for the three organisms, we measured the performance of BMF on a single compute node. The performance is also compared to a serial program of mass fingerprinting without FastBit. We developed a sequential program based on Algorithm 1. It directly calculates the molecular weight of peptides of all matching lengths, and compares them to the range of targeted masses. This "plain" implementation searches peptides from the first amino acid all the way to the last one.

Figure 4 shows the execution time of BMF. The candidates are peptides of 3 to 43 amino acids. The range of targeted molecular weights affects the number of candidate proteins. We used two different ranges: $[1900.0, 2000.0]$ and $[2900.0, 3000.0]$. In our experience, the actual range has little impact to the computational complexity because all entries in the weight table are compared with the range in any case. Compared to the plain implementation, BMF significantly reduces the query time to identify all candidate peptides. BMF achieved 16, 30, and 45 times speedup in execution, respectively for Bacteria, Yeast, and Drosophila.

[2]ftp://ftp.wwpdb.org/pub/pdb/derived_data/

### C. Tuning of Performance Parameters

As discussed in Section III, BMF pre-computes molecular weights of all peptides of suitable lengths. To support fast query of candidate peptides, it partitions a large 2-D weight table into many small blocks at protein boundaries. A varying number of columns were generated for FastBit. The actual number of columns determines the query range of peptide lengths in BMF.

We measured the time to create FastBit indexes in BMF, denoted as the "index" time. This time includes the duration to pre-compute and store the molecular weights as floating-point numbers, and the time to index these weights.

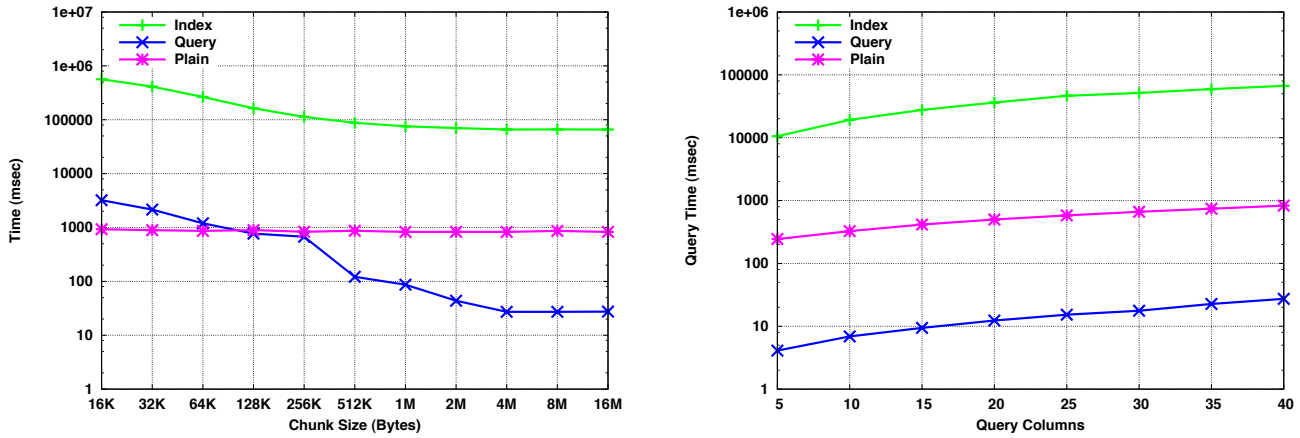
Figure 5 shows the timings for different phases of BMF on a single compute node. They are also compared to that of the plain implementation, which does not have any extra phase. We vary the query range from 5 to 40 amino acids and the chunk size from 16 KB to 16 MB. As shown in the figure, BMF spends a significant amount of time to create FastBit indices, the majority of which was spent on indexing the precomputed molecular weights. Note that the time for weight computation and index creation are one-time costs for a mass fingerprinting server. This costly step is rather paid off with the time savings on numerous queries.

Figure 5 (a) illustrates the performance of BMF with a varying chunk size. The chunk size of 4 MB and above led to optimal BMF performance. However, the actual chunk size for a particular run of BMF is determined based on other tradeoffs as well. For example, smaller chunk sizes can be helpful to facilitate parallelized protein identification from a small-size database.

Figure 5 (b) demonstrates the performance of BMF with varying query ranges. A wider query range increases the execution time of BMF. However, the growth of execution time is slow in comparison to the query range. Thus it makes sense for BMF to create all columns that can support a wide query range. In the rest of this performance evaluation, FastBit indices are always generated for forty columns (an adjustable query range) in a weight table of a protein database.

### D. Benefits of Parallel Computation

Based on results from the above performance evaluation, we carried out an experiment to examine the benefits of parallel computation to BMF. To minimize the impact of I/O access, we replicated the protein databases for bacteria, yeast, and drosophila across all nodes, on their local disk file system.




(a) Chunk Size

(b) Query Range

Fig. 5: Performance of BMF with Varying Parameters.

In this experiment, each process creates FastBit indices for different chunks (or different columns of the same chunk) of an identical database. Given the small size of three databases, we used a chunk size of 1 MB to achieve good parallelization.

Figure 6 shows the performance of BMF with parallel computation. With an increasing number of processes, the execution times for index creation and protein query are dramatically reduced. The improvement from parallel computation is in close proportion to the number of processes, until the execution time is minimized due to the availability of parallelization, e.g. when using 160 processes to handle a small bacteria protein database.

### *E. Benefits of Parallel I/O*

BMF generates large amounts of data when when it computes peptide masses and creates a database of searchable masses from a protein database. In Section IV-D, we demonstrate that BMF can achieve good parallelized computation when the protein database is replicated on the local disk of all nodes. However, contemporary systems are typically configured with a shared file system. Parallel file systems are also commonly used to enable parallel I/O for data intensive applications. Thus, it is very important to examine the impact of I/O on the performance of BMF in environments with shared file systems.

We conducted an experiment to evaluate the benefits of parallel I/O to BMF. In our tests, protein databases for bacteria, yeast, and drosophila are located on a shared file system, NFS or Lustre. The files generated by BMF for molecular weights and their indices are also stored on the shared file system.

We measured the time to index the computed molecular weights and the time for searching peptides. Figure 7 demonstrates the performance comparisons of BMF on NFS and Lustre. Lustre performs significantly better than NFS for indexing the molecular weights. It reduces the index time by nearly one order of magnitude for all databases. Thus, Lustre is able to exhibit its performance advantage when creating the index files from these databases. NFS, on the other hand, performs better in answering peptide queries, though the absolute savings are only several 10s of milliseconds.

### *F. Large-scale Parallelization of BMF*

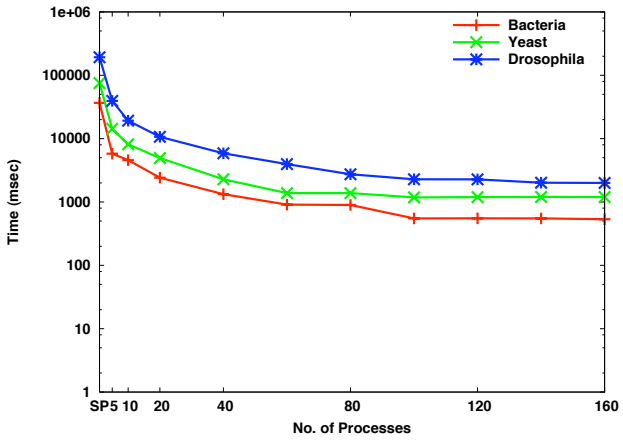
With genomic sequences and their translations into increasingly large protein databases, it is critical for computational mass fingerprinting programs such as BMF to handle these gigantic databases.

As described in Section III, BMF is designed to partition its 2-D table of molecular weights into small blocks, and then distribute the computation to all available processes. Using Jaguar, we examined the benefits of massive parallelization to BMF. In these tests, we used three large protein databases, PDB, SwissProt, and NR. Given the large size of these databases, we used a chunk size of 8 MB for this experiment.

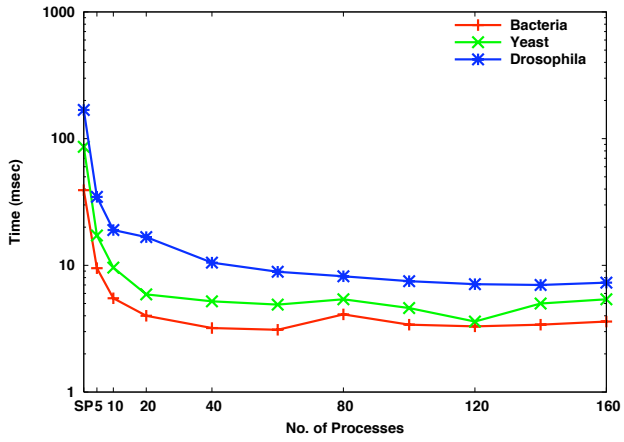
Figure 8 shows the performance of BMF in processing PDB and SwissProt databases on Jaguar. For both PDB and SwissProt, BMF is able to significantly reduce the index time. Once the indices for a database of molecular weights are generated, the query time is very small, and the benefit from massive parallelization is small.

The NR protein database is a combination of several protein databases. We evaluated the performance of BMF for NR on Jaguar, using up to 8,192 cores. Because NR contains 3.9 billions amino acids, BMF is able to make good use of the massive power of these systems. Figure 9 demonstrates the performance of BMF in processing the NR database. BMF is able to reduce the execution time for index creation, Based on the results from both systems, the query time gradually scales down until it is about 20 microseconds.

In summary, these results demonstrate that BMF is able to leverage the massive power of Jaguar supercomputer, particularly in creating FastBit indices from large-scale protein databases. In addition, these results suggest that an effective way to utilize the massive power of supercomputers is to create the indices on the supercomputers, and then enable peptide

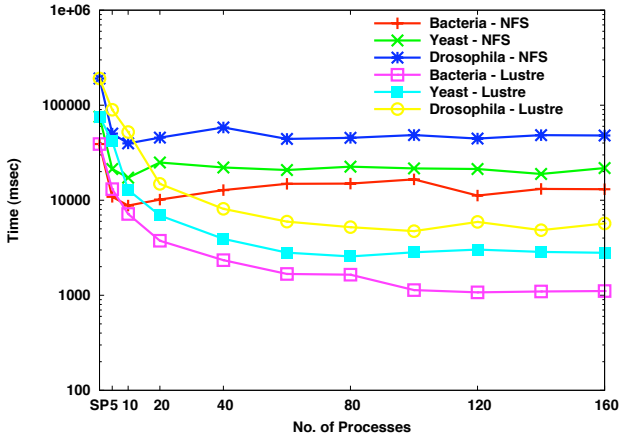



(a) Index

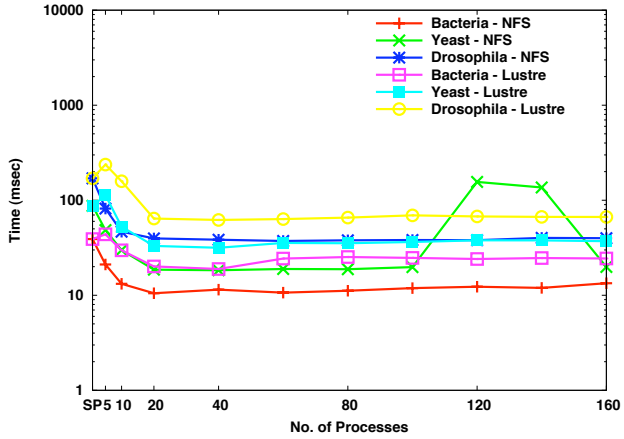



(b) Query

Fig. 6: Performance of Parallelized BMF.




(a) Index




(b) Query

Fig. 7: Performance of BMF on NFS and Lustre.

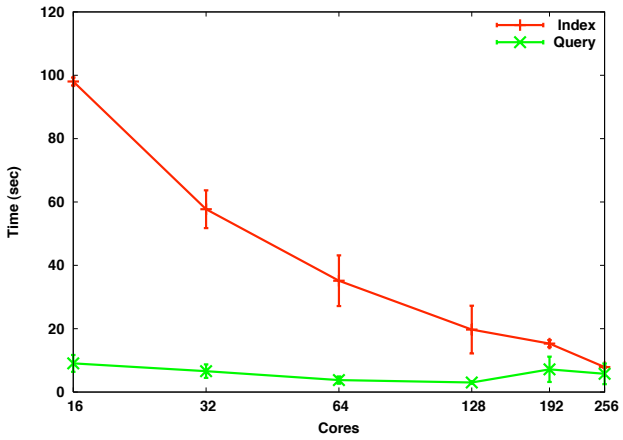
search queries through less powerful computers such as small or medium size clusters.

## V. Related Work

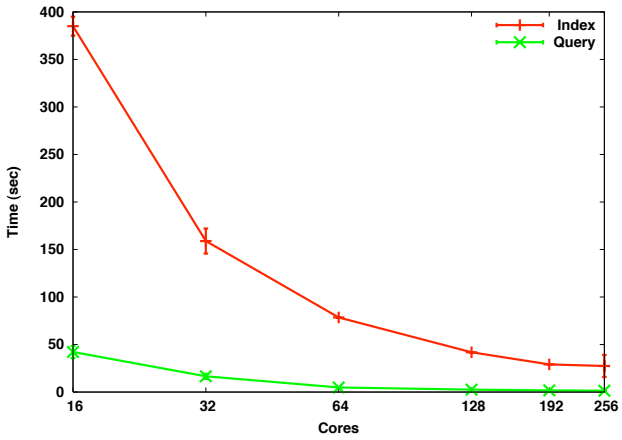
Protein identification is an active topic of research in proteomics and biology. Here we review several studies that make use of computing tools and protein databases. Perkins *et al.* [26] investigated the use of a scoring function to enable probability-based protein identification. Brunetti *et al.* [27] introduced a parallel algorithm to find peptide sequence tags for protein identification. Han *et al.* [28] developed an Internet-based server that enables protein identification by using amino acid sequence tags computed through de novo sequencing. Mann *et al.* [29] demonstrated the use of combined sequence tags for highly-sensitive peptide identification, and validated with practical examples from electrospray mass spectrometry. Shadforth *et al.* [30] provided a comprehensive review of protein identification, and compared recent studies on the basic algorithms, database searching tools, and de novo sequencing. All these efforts have not tried to use supercomputers for protein identification. In this paper, we show how to combine the strength of FastBit index technology and the power of supercomputers to create searchable databases of peptide masses, thereby enabling efficient and scalable protein identification from gigantic protein databases such as NR [5].

## VI. Conclusion

We have successfully demonstrated a massively parallelized software, *Bitmapped Mass Fingerprinting*, that can leverage the power of supercomputers for fast protein identification. This research is accomplished by defining computational peptide mass fingerprinting algorithmically and designing BMF from scratch to maximize the parallelism in computation and I/O. To realize a scalable tool for mass fingerprinting, we design BMF as a software architecture that integrates a flex-based amino acid tokenizer, the FastBit index technology,




(a) PDB




(b) SwissProt

Fig. 8: Performance of BMF in Processing PDB and SwissProt Databases.

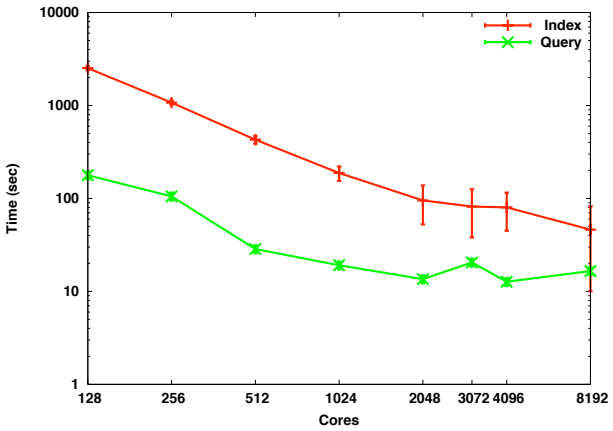



Fig. 9: BMF-based Protein Identification from the NR Database.

and the efficient and portable MPI programming library. Our experimental evaluation indicates that BMF is able to deliver superb performance for protein identification from various small and large protein databases. In addition, because BMF exploits the power of parallel I/O and parallel computation, it is able to efficiently create the index files required by FastBit. In one case, we demonstrate that up to 8,192 cores from the Jaguar supercomputers can be effectively utilized by BMF for protein identification from the world's largest non-redundant (NR) protein database [5].

In the future, we plan to build a BMF-based mass fingerprinting server using a hybrid of a supercomputer and a small cluster, for index creation and query services respectively. We also intend to study the applicability of BMF to other loosely coupled computing environments, such as distributed and cloud computing. We believe that the flexibility of BMF in exploiting parallelism can also provide a similar advantage when using contemporary cloud computing software and hardware tools such as Hadoop [31] and Microsoft Azure services [32]. Furthermore, we plan to investigate the applicability of FastBit to other computational biology problems such as sequence alignment and motif identification.

## ACKNOWLEDGMENTS

We are very thankful to Dr. Arie Shoshani from Lawrence Berkeley National Laboratory and Dr. Xiao Qin from Auburn University for their comments on earlier drafts of this paper, and Xinyu Que from Auburn University for his help on running some experiments for this work.

This research is sponsored in part by NSF awards CNS-1059376 and CNS-0917137, and in part by the Office of Advanced Scientific Computing Research, U.S. Department of Energy. This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

## REFERENCES

[1] M. Barber, R. S. Bordoli, R. D. Sedgwick, and A. N. Tyler, "Fast atom bombardment of solids (f.a.b.): a new ion source for mass spectrometry," *J. Chem. Soc., Chem. Commun.*, vol. 7, pp. 325–327, 1981.

[2] A. R. Dongr, J. K. Eng, and J. R. Yates III, "Emerging tandem-mass-spectrometry techniques for the rapid identification of proteins," *Trends in Biotechnology*, vol. 15, no. 10, pp. 418 – 425, 1997.

[3] D. C. Reiber, T. A. Grover, and R. S. Brown, "Identifying proteins using matrix-assisted laser desorption/ionization in-source fragmentation data combined with database searching," *Analytical Chemistry*, vol. 70, no. 4, pp. 673 – 683, 1998.

[4] A. Frank, S. Tanner, V. Bafna, and P. Pevzner, "Peptide sequence tags for fast database search in mass-spectrometry," *J. Proteome*, vol. 4, no. 4, pp. 1287–95, 2005.

[5] NCBI, "National Center for Biotechnology Information," http://www.ncbi.nlm.nih.gov/.

[6] K. Wu, S. Ahern, E. W. Bethel, J. Chen, H. Childs, E. Cormier-Michel, C. Geddes, J. Gu, H. Hagen, B. Hamann, W. Koegler, J. Lauret, J. Meredith, P. Messmer, E. Otoo, V. Perevoztchikov, A. Poskanzer, Prabhat, O. Rubel, A. Shoshani, A. Sim, K. Stockinger, G. Weber, and W.-M. Zhang, "Fastbit: Interactively searching massive data," in *SciDAC 2009*, 2009, lBNL-2164E.

[7] K. Wu, E. Otoo, and A. Shoshani, "Optimizing bitmap indices with efficient compression," *ACM Transactions on Database Systems*, vol. 31, pp. 1–38, 2006.

[8] K. Wu, A. Shoshani, and K. Stockinger, "Analyses of multi-level and multi-component compressed bitmap indexes," *ACM Trans. Database Syst.*, vol. 35, no. 1, pp. 1–52, 2010.

[9] R. Sinha, M. Winslett, K. Wu, K. Stockinger, and A. Shoshani, "Adaptive bitmap indexes for space-constrained systems," in *ICDE 2008*, 2008, pp. 1418–1420.

[10] K. Wu, K. Stockinger, and A. Shosani, "Breaking the curse of cardinality on bitmap indexes," in *SSDBM'08*, 2008, pp. 348–365, preprint appeared as LBNL Tech Report LBNL-173E.

[11] K. Wu, E. Otoo, and A. Shoshani, "A performance comparison of bitmap indexes," in *CIKM*, 2001, pp. 559–561.

[12] E. W. Bethel, S. Campbell, E. Dart, K. Stockinger, and K. Wu, "Accelerating network traffic analysis using query-driven visualization," in *Symposium on Visual Analytics Science and Technology (VAST), Baltimore, Maryland, USA, October 31 - November 2, 2006*. IEEE Computer Society Press, 2006.

[13] K. Wu, W.-M. Zhang, V. Perevoztchikov, J. Lauret, and A. Shoshani, "The grid collector: Using an event catalog to speed up user analysis in distributed environment," in *Proceedings of Computing in High Energy and Nuclear Physics (CHEP) 2004*, 2004.

[14] K. Stockinger, J. Shalf, W. Bethel, and K. Wu, "Query-driven visualization of large data sets," in *IEEE Visualization 2005, Minneapolis, MN, October 23-28, 2005*, 2005, lBNL report LBNL-57511.

[15] K. Wu, W. Koegler, J. Chen, and A. Shoshani, "Using bitmap index for interactive exploration of large datasets," in *Proceedings of SSDBM 2003*, Cambridge, MA, USA, 2003, pp. 65–74.

[16] O. Rübel, Prabhat, K. Wu, H. Childs, J. Meredith, C. G. R. Geddes, E. Cormier-Michel, S. Ahern, G. H. weber, P. Messmer, H. Hagen, B. Hamann, and E. W. Bethel, "High performance multivariate visual data exploration for extemely large data," in *SuperComputing 2008 (SC08)*, Austin, Texas, USA, Nov. 2008, p. 51, lBNL-716E.

[17] J. Schlosser and M. Rarey, "Beyond the virtual screening paradigm: Structure-based searching for new lead compounds," *Journal of Chemical Information and Modeling*, vol. 49, no. 4, pp. 800–809, 2009.

[18] T. L. L. Siqueira, C. D. de Aguiar Ciferri, V. C. Times, A. G. de Oliveira, and R. R. Ciferri, "The impact of spatial data redundancy on SOLAP query performance," *J. Braz. Comp. Soc.*, vol. 15, no. 2, pp. 19–34, 2009.

[19] W. Yu, J. Vetter, R. Canon, and S. Jiang, "Exploiting Lustre File Joining for Effective Collective I/O," in *7th Int'l Conference on Cluster Computing and Grid (CCGrid'07)*, Rio de Janeiro, Brazil, May 2007.

[20] D. Hildebrand, L. Ward, and P. Honeyman, "Large files, small writes, and pnfs," in *Proceedings of the 20th ACM International Conference on Supercomputing (ICS06)*, Cairns, Australia, June 2006.

[21] W. Yu, J. Vetter, and H. Oral, "Performance Characterization and Optimization of Parallel I/O on the Cray XT," in *22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS'08)*, Miami, FL, April 2008.

[22] The Flex Project, "flex: The Fast Lexical Analyzer," http://flex.sourceforge.net/.

[23] WWPDB, "Worldwide Protein Data Bank," 2010, http://www.wwpdb.org/.

[24] Swiss Institute for Bioinformatics, "Swiss-Prot Protein Knowledgebase," http://ca.expasy.org/sprot/.

[25] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–10, October 1990.

[26] D. N. Perkins, D. Pappin, D. M. C. DM, and J. Cottrell, "Probability-based protein identification by searching sequence databases using mass spectrometry data," *Electrophoresis*, vol. 20, no. 18, pp. 3551–67, December 1999.

[27] S. Brunetti, E. Lodi, E. Mori, and M. Stella, "Parpst: a parallel algorithm to find peptide sequence tags," *BMC Bioinformatics*, vol. 9, 2008.

[28] Y. Han, B. Ma, and K. Zhang, "Spider: software for protein identification from sequence tags with de novo sequencing error," *Journal of Bioinform. and Comput. Biol.*, vol. 3, pp. 697–716, 2005.

[29] M. Mann and M. Wilm, "Error-tolerant identification of peptides in sequence databases by peptide sequence tags," *Analytical Chemistry*, vol. 66, no. 24, pp. 4390–4399, 1994.

[30] I. Shadforth, D. Crowther, and C. Bessant, "Protein and peptide identification algorithms using ms for use in high-throughput, automated pipelines," *Proteomics*, vol. 5, no. 16, pp. 4082–95, 2005.

[31] The Apache Group, "Hadoop," http://hadoop.apache.org/.

[32] MicroSoft, "MicroSoft Azure Cloud," http://www.microsoft.com/azure/.