SCALABLE PARALLEL ALGORITHMS AND SOFTWARE FOR LARGE SCALE PROTEOMICS

By

GAURAV RAMESH KULKARNI

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and
Computer Science

December 2009

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of GAURAV RAMESH KULKARNI find it satisfactory and recommend that it be accepted.

Ananth Kalyanaraman, Ph.D., Chair

William R. Cannon, Ph.D.

Partha Pratim Pande, Ph.D.

ACKNOWLEDGEMENTS

I would like to thank all those people, whose guidance and support made this research project possible. I would like to thank my supervisor, Dr. Ananth Kalyanaraman for trusting my skills. His suggestions and continuous support was invaluable throughout the research. I would also like to express my gratitude to the members of the committee, Dr. William R. Cannon and Dr. Partha Pratim Pande for their expert comments. I would also like to thank Douglas Baxter, for his helpful suggestions in understanding the *MSPolygraph* code and implementing the algorithms proposed. I would also like to thank my family, colleagues Andy Wu, Pawan Agarwal, Souradip Sarkar and Md. Muksitul Haque, and friends for their support and help throughout the journey.

The research was supported by National Science Foundation (grant #0916463).

SCALABLE PARALLEL ALGORITHMS AND SOFTWARE FOR LARGE SCALE PROTEOMICS

Abstract

by Gaurav Ramesh Kulkarni, M.S.
Washington State University
December 2009

Chair: Ananth Kalyanaraman

The problem of identifying the sequence that constitutes a peptide or a protein molecule is referred to as "peptide identification" and is of fundamental importance in proteomics research. The most popular approach to identify peptides is through database search, in which an experimental spectrum generated from fragments of an unknown peptide using mass spectrometry is compared with a database of known protein sequences. The goal is to identify the unknown peptide by comparing to database sequences. The exponential growth rates and overwhelming sizes of sequence databases make this an ideal application for parallel computing. However, the present generation of software tools is *not* expected to scale to the magnitudes of data that will be generated in the next few years, as they are either serial algorithms or parallel strategies that are inherently serial. In this thesis, we present an efficient parallel approach for peptide identification through database search. Three key factors distinguish our approach from that of existing solutions: i) (*space*) Given þ processors and a database with N residues, we provide the first space-optimal parallel algorithm ($O(N/þ)$); ii) (*time*) By masking communication with computation and by using MPI one-sided communication primitives, our algorithm minimizes parallel overhead and ensures efficient scaling of run-time; and iii) (*quality*) The performance gains achieved using parallel processing has allowed us to incorporate highly accurate, albeit compute-intensive statistical models. We present the design and evaluation of two different algorithms that implement our approach. Experimental results using 2.65 million microbial proteins show linear scaling up to 128 processors, with parallel efficiency maintained at ~50%. As a concrete demonstration of a real world application, we applied our approach for *de novo* identification of species from metagenomics data. To the best of our knowledge, this is the first time mass spectral data have been used for metagenomics species identification. Collectively, we expect that the approaches presented in this thesis will be critical to meet the data-intensive and qualitative demands stemming from this important application domain.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

Proteomics is the study of proteins – by their sequence, structure, function and interaction. It is a central topic in systems biology research, where the goal is to understand the composition and functioning of a biological system by its components. A fundamental problem in proteomics is the identification of proteins, or more generally peptides, that are expressed in a specific organism or a community of organisms. As proteins constitute the primary molecular basis for cellular functions, peptide identification is important for understanding the cellular dynamics of organisms under various environmental conditions. It also holds the key to advance our understanding of the molecular basis of diseases and thereby, for their treatment. Recently, peptide identification has also become critically important in the emerging area of *metagenomics*, where the goal is to understand the composition and functions of environmental microbial communities.

A *peptide* is a fragment derived from a protein sequence. For computational purposes, a peptide can be represented as a sequence (or string) of characters over the alphabet of 20 amino acids residues present in the nature. The problem of identifying an unknown peptide is therefore equivalent to the problem of finding the sequence of amino acid residues that constitutes it.

There are two primary approaches followed for peptide identification: i) *De novo* peptide identification [Chen *et al.*, 2005; Dancick *et al.*, 1999; Liu C. *et al.*, 2006; Pevzner *et al.*, 2001; Taylor *et al.*, 1997] and ii) peptide identification through database search [Craig *et al.*, 2003; Craig *et al.*, 2004; Duncan *et al.*, 2005; Bjornson, 2008; Cannon *et al.*, 2005]. Both approaches rely on the use of experimental spectral data obtained using mass spectrometry (aka mass spectroscopy).

1.1 Mass Spectrometry

Mass spectrometry ("MS") is a powerful and now a standard technique used to generate experimental spectrum for an unknown peptide. In this technique, multiple copies of an unknown (target) peptide are experimentally fragmented and the number of occurrences of all the fragments generated along with their mass-to-charge ratio (m/z) is recorded. The abundance or frequency of these fragments is then plotted over a range of mass-to-charge ratio values. The resulting plot of peak intensities (y-axis) to m/z values (x-axis) is called an *experimental spectrum* for the target peptide. Along with the experimental spectrum, this technique also outputs total mass of the parent target peptide. From a sequence standpoint, MS tries to capture as fragments corresponding to all prefixes and suffixes of the parent peptide. A typical graph (spectrum) produced by the spectrometer is as shown in Figure 1.1.

The subsequent computational task is to deduce the peptide sequence from its experimental spectrum. For example, Figure 1.1 shows an experimental spectrum with several peaks. Here, each peak corresponds to a unique fragment and the location of the peak is determined by the sum total of mass and charge of its constituent amino acid residues. Therefore, by inspecting the differences of the m/z values between peaks, the sequence of

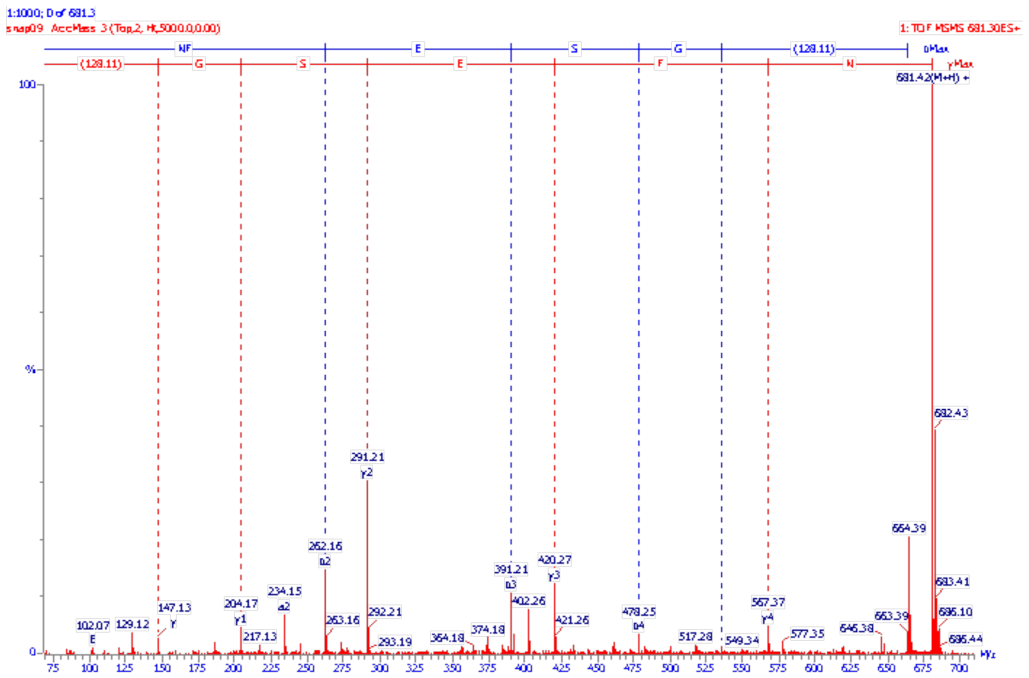

Figure 1.1 – A typical experimental spectrum generated from Mass Spectrometry. Figure Source: *http://www.astbury.leeds.ac.uk/facil/MStut/mstutorial.htm* Last Date Accessed: Nov11, 2009.

the parent peptide (and its individual fragments) could be reconstructed, as shown in the example of Figure 1.1. Such computational approaches are called *de novo* peptide identification approaches [Chen *et al.*, 2005].

There are a couple of limitations with *de novo* approaches. Experimental noise could manifest itself in the form of false peaks which could potentially render the computational identification infeasible. Also, the identification process is compounded by the need to take into account for a number of variations such as post-translational modifications or PTMs.

A complementary and a more popular approach is to use *database search* [Craig *et al.*, 2003; Craig *et al.*, 2004; Duncan *et al.*, 2005; Bjornson, 2008; Cannon *et al.*, 2005], wherein an experimental spectrum is compared against model spectra generated from a database of already known protein sequences. The goal of this matching operation is to identify those protein sequences in the database that are most likely to have generated an experimental spectrum similar to that of the test spectrum of the unknown target peptide. However, this operation could become computationally cumbersome if each test spectrum is to be compared against the entire database. To reduce the burden on computation, a subset of database sequences is first shortlisted as "*candidates*" and the extensive evaluation by spectrum to sequence matching is performed only over the candidates.

Candidates are typically identified by a simple, computationally inexpensive, mass-based filtering scheme. The spectrum to sequence matching is performed by first generating a model spectrum for each of the candidate sequence and then comparing the two spectra using a scoring function that assigns a likelihood score. Subsequently, the results of matching are rank ordered and output by their likelihood scores.

1.2 Computational Challenges in Peptide Identification

1.2.1 Database Sizes and Growth Rates

Given the reliance of peptide identification on databases, the explosive growths observed in molecular sequence databases is of a primary and immediate concern. The databases used by peptide identification approaches typically include conventional protein sequences (e.g.,UniProt/Swiss-Prot [Boeckmann *et al.*, 2003]) and/or unconventional peptide sequences derived from putative open reading frames (ORFs) of genomic/metagenomic DNA (e.g., CAMERA portal [CAMERA, 2009]). These collections are growing at exponential rates [Boeckmann et al., 2003; McCormack, 1994]. As seen from Figure 1.2, the Swiss-Prot database has grown from 1e+6 to 1e+7 in 5 years. Furthermore, nucleotide databases are also growing at an even faster rate,

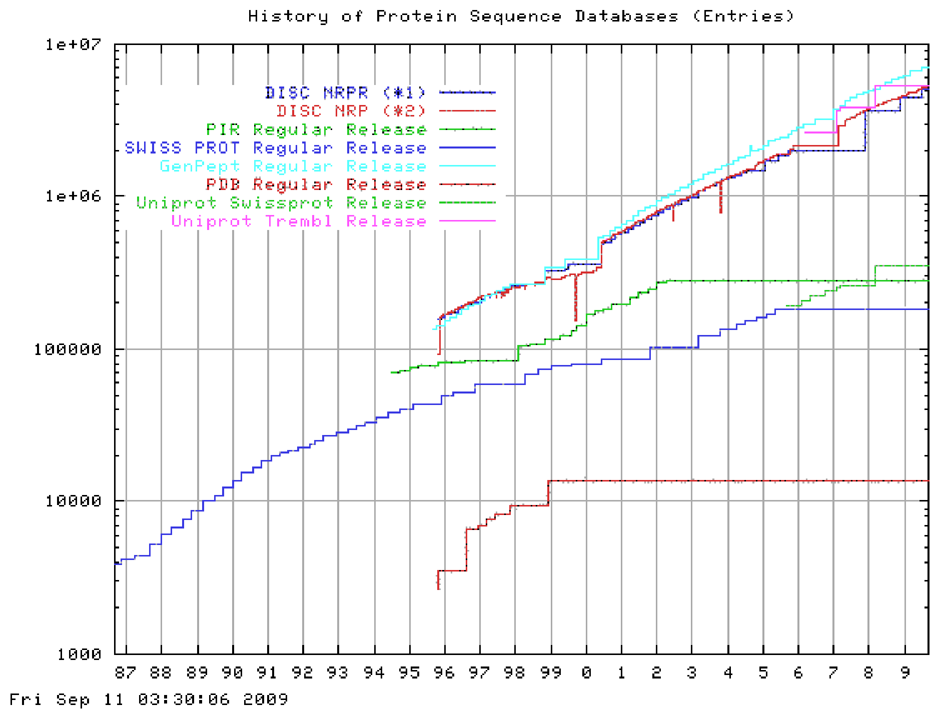

Figure 1.2 – Growth of different databases from the year 1987. Figure source: *http://www.dna.affrc.go.jp/growth/P-history.html.*

as shown in the Figure 1.3. *The rapid growth of these databases has made peptide identification a data-intensive problem.*

Figure 1.4 depicts average number of candidates generated per spectrum for different set of sequences. For protein families this number is 1e+4, while for microbial communities this number is more than 1e+7. If a sample involves any unsequenced genome(s) corresponding to the target peptides, which is typically the case in metagenomic projects, the number of candidates

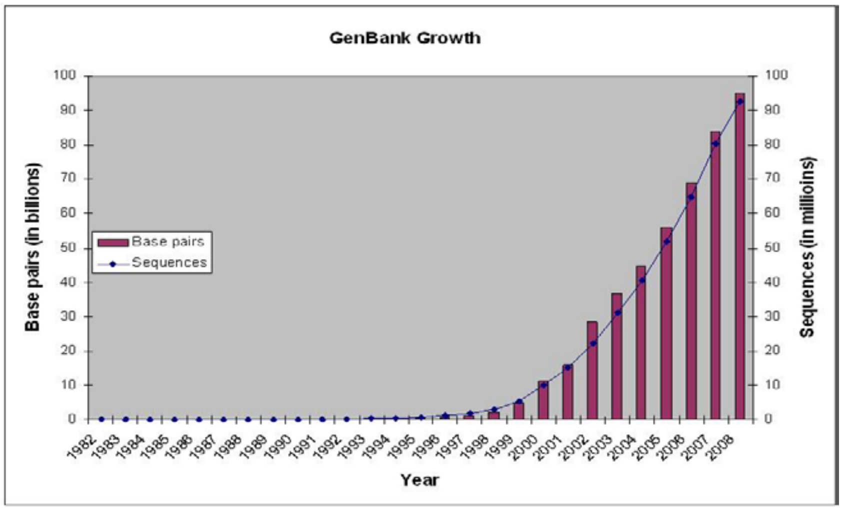

Figure 1.3 –Plot showing data growth in the NCBI GenBank nucleotide database.

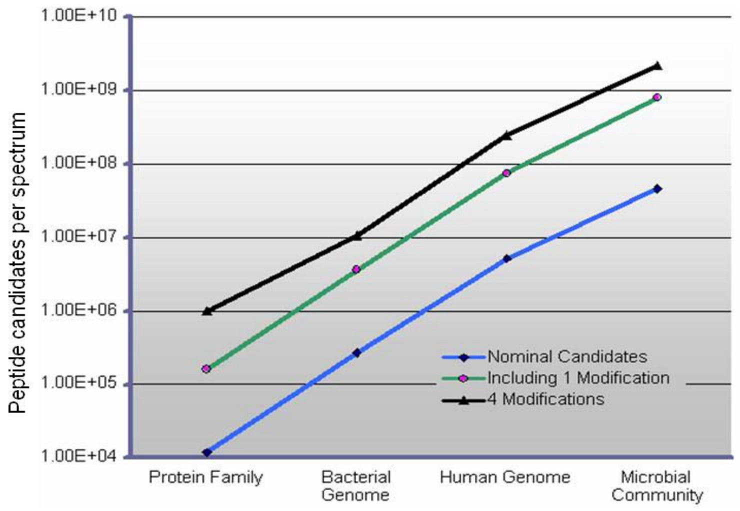

Figure 1.4 – Plot showing the number of peptide candidates to be evaluated per experimental spectrum generated from different data classes.

for evaluation increases by orders of magnitudes. Taking post translational modifications into (PTMs) into account increases it by 10 to100, depending upon the number of PTMs considered. *The need for computing these large volumes of candidate comparisons makes peptide identification a compute-intensive application as well.*

1.2.2 Current State of Computational Tools

While all the above problem characteristics suggest that peptide identification is an ideal candidate to benefit from the application of high performance computing (HPC), there are hardly any HPC approaches in the current suite of software tools. The present generation of software tools [Craig *et al.*, 2003; Craig *et al.*, 2004; Duncan *et al.*, 2005; Bjornson, 2008; Cannon *et al.*, 2005] are all either serial programs or parallel programs that are built upon inherently serial algorithms. Due to this design limitation, both space and time pose a serious scalability bottleneck for their application in large-scale peptide identification. All current software that implement peptide database search store the entire database in the local memory of each compute node. For instance, a processor with 1GB RAM can store up to ~1.27×10^6 sequences, assuming an average sequence length of 300 amino acid residues. Running the code on a machine with more memory is still not a scalable solution here, given the exponential growth rates the sequence databases have been witnessing. One way to overcome this space bottleneck is to resort to hard disk. The database could be stored in hard disk and could be fetched part by part in memory. This would require frequent access to hard disk, which is an expensive operation. This could exacerbate the performance of a system. Multiple processors simultaneously accessing the databases would result in serialization issues as well.

1.3 Motivation for New Development

This gap in computational tools coupled with the potential of the application to benefit from HPC, collectively sets the motivation for this thesis work – i.e., toward the development and application of novel, scalable HPC tools for large-scale peptide identification.

1.4 Thesis Contributions

In this thesis, we investigate the problem of peptide identification from the perspective of large-scale application. We present a new algorithm and software that follows the databases search approach and solves the problem of peptide identification in parallel. Three key factors distinguish our approach from that of existing solutions: i) (*space*) Given p processors and a database with N residues, we provide the first space-optimal algorithm ($O(N/p)$) under distributed memory machine model; ii) (*time*) Our algorithm uses a combination of parallel techniques such as one-sided communication and masking of communication with computation to ensure that the overhead introduced due to parallelism is minimal; and iii) (*quality*) The approach uses same techniques as *MSPolygraph* for peptide identification. These techniques include use of more accurate probabilistic scoring functions which ensure high quality output. Our experimental results using 2.65x10^6 microbial protein sequences show linear scaling up to 128 processors on a Linux commodity cluster, with parallel efficiency at ~50%. Finally, we also describe a real-world large-scale application of species identification in metagenomic data, which makes use of our peptide identification approach.

Collectively, we expect that the work presented in this thesis will be critical to meet the data-intensive and qualitative demands stemming from this important application domain.

CHAPTER TWO

BACKGROUND AND RELATED WORK

2.1 Computational Peptide Identification

There are two approaches primarily followed in peptide identification: i) *De novo* peptide sequencing approach ii) Database search approach.

In the *de novo* sequencing approach [Chen *et al.*, 2000], an experimental spectrum is first converted into a spectrum graph, which is an acyclic directed graph. Each peak from the experimental spectrum is represented as *two* nodes, (called N_i and C_i for a peak *i*) in the graph to represent a possibility that a fragment could either be a prefix molecule or a suffix molecule derived from the target peptide. These nodes (N_i and C_i for each of the peaks *i*) are plotted on an m/z line one after another such that N_i is plotted at a distance equal to the mass of the corresponding fragment/peak (which is obtained from the experimental spectrum) and C_i is plotted at a distance equal to, total mass of the target peptide minus mass of the fragment/peak. Edges are then drawn from a node x to a node y if and only if all the following conditions are satisfied: i) x comes before y on the m/z line. ii) Their mass difference is equal to total mass of some amino acid sequence. iii) They are *not* originated from the same peak [Chen *et al.*, 2000].

Here, each path from start node to end node represents a possible sequence for the peptide. Such a path which goes through *exactly* one node (N_i or C_i) for each pair is then found out. This path corresponds to possible amino acid sequence of the target peptide. This approach however is sensitive to experimental noise and requires accurate data that current technologies are as yet incapable of producing. This approach is spectrometer specific too. Each spectrometer has a different level of accuracy and produces different types of fragments. The de novo approach is therefore limited by its capability or the lack of it to take into account these differences.

An alternative approach to peptide identification is through database search. Several programs exist for peptide identification through database search: *Mascot* [McCormack *et al.*, 1994], *Sequest* [Perkins *et al.*, 1999], *Tandem* [Craig *et al.*, 2004], *X! Tandem* [Craig *et al.*, 2003], *Parallel Tandem* [Duncan *et al.*, 2005], X!!*Tandem* [Bjornson *et al.*, 2008] and *MSPolygraph* [Cannon *et al.*, 2005]. Out of these, *Mascot* [McCormack *et al.*, 1994] and *Sequest* [Perkins *et al.*, 1999] are commercially available and are serial programs. In 2004, Craig et al. [Craig *et al.*, 2004] came up with an implementation of Tandem, which is again is a serial program. X!Tandem [Craig *et al.*, 2003] is one of the most popular software tools available, and is a shared memory implementation of Tandem. X!Tandem executes parallel threads on shared memory multiprocessors (SMPs). However, to achieve significant speed up it requires large SMPs.

Since these machines are expensive. Duncan et al. [Duncan *et al.*, 2005] came up with a parallel version of X!Tandem, called Parallel Tandem. Parallel Tandem makes use of either PVM or MPI along with X!Tandem. However, this version is a collection of ad-hoc scripts having following drawbacks: i) Results produced by parallel tandem differ significantly from that of X!Tandem. ii) The final sequential step puts limitations on parallelization. [Bjornson *et al.*, 2008].

Bjornson et al. [Bjornson *et al.*, 2008] created another implementation of Tandem using MPI, in which they addressed the issues discussed above. This version of Tandem, called X!!Tandem, produces identical results as X!Tandem and implements parallelism in a better way by parallelizing both the refined and unrefined steps of X!Tandem. X!!Tandem, therefore, has been observed to provide better speedup and decreased runtime than Parallel Tandem, as shown in Figure 2.1. The strength of this program is its speed. During experimentation, we found that X!!Tandem takes less than 2 minutes to evaluate 1210 experimental spectra against a database having 2.65×10^6 sequences on 8 processors.

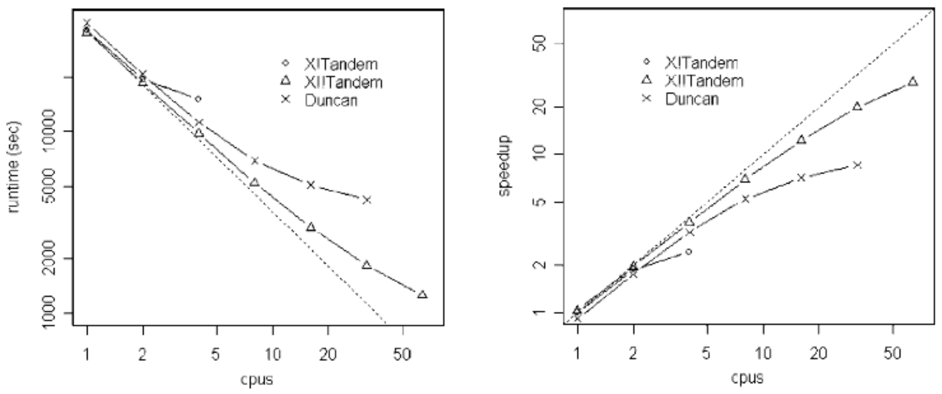

Figure 2.1 – Plots [Bjornson *et al.*, 2008] showing a) Total runtime of the X!Tandem, X!!Tandem, and Duncan methods (Parallel Tandem), plotted log-log and b) Speedup of the X!Tandem, X!!Tandem, and Duncan methods (Parallel Tandem), plotted log-log. The plots have been reproduced here with permission from the *Journal of Proteome Research*.

2.2 Trade offs between Quality and Performance

The superior performance shown by X!!Tandem is mainly because of the fast, fairly simple statistical model used by the algorithm and the aggressive filtering step followed at the beginning. In this step, a rapid survey of protein sequences is carried out to select potential candidates. This step does not take PTMs into account. This could miss out some true predictions, hence affecting quality and accuracy of the output. This is important, especially in case of complex data such as metagenomics data where spectra are to be compared with an overwhelmingly large number of candidates (Refer Figure 1.4). The filtering step could miss out true predictions in this case. Due to this, models with more accuracy are required. However, these models would take significantly more amount of time for evaluation.

Cannon *et al.* [Cannon *et al.*, 2005] came up with a more accurate method (*MSPolygraph*) that makes use of probability and log likelihood ratio for evaluation. *MSPolygraph* is unique in its flexibility to handle model spectra in that it combines the use of highly accurate spectral libraries, when available, with the use of on-the-fly generation of sequence averaged model spectra when spectral libraries are not available. This effort to enrich quality of prediction, however, comes with increased computation cost. For example, we found that *MSPolygraph* takes approximately 42 minutes for 1210 spectra with 2.65×10^6 sequences on 128 processors, while the same run of X!!Tandem on 8 processors takes less than 2 minutes.

2.3 Memory Scalability: A Bottleneck

The algorithmic steps in *MSPolygraph* can be summarized as follows:

S1) Instantiate one master processor and $p-1$ worker processors (p is the number of processors used). The master processor loads query set *Q* into its local memory, while all workers load the entire protein database *D* in its local memory [Gropp *et al.*, 2009].

S2) The master processor starts by distributing small, fixed size batches of experimental spectra (or queries) to individual worker processors.

S3) Each worker processor works on the assigned batch of queries, processing one query at a time and reporting at most τ top matched sequences to an output file. After processing all the queries in the assigned batch, worker processor sends notification to master processor. Master processor then assigns a new batch of queries to that processor.

S4) The process is continued iteratively until all the queries at the master processor have been processed.

The above parallelization scheme has two main advantages: i) the processing of every query is strictly localized within each worker and generates practically no communication *during* processing, thereby saving on the cost of communication; and ii) the master processor plays the role of a load distributor and since the queries are allocated to worker processors in small batches based on demand, the workload is expected to be balanced. However, the underlying algorithm of *MSPolygraph* is such that *each* processor stores entire database in memory, resulting in an $O(N)$ space complexity. Since each processor stores the entire database in memory it could result in a space bottleneck, especially considering the exponential growth rates of the public sequence databases. We observed that with 1 GB memory the current implementation could take 1.27×10^6 protein sequences, beyond which the code resorts to swap space or crashes out of memory. Space is at least as important a consideration as run-time efficiency, given the need for analyzing metagenomics data sets containing peptides from many organisms is becoming increasingly common. This necessitates an efficient and memory scalable implementation such that the resulting tool would scale with processor size with respect to memory and time, and would produce high quality results.

To address these issues, we developed a distributed memory algorithmic model for *MSPolygraph*. The idea here is to distribute database over available processors and then fetch the database by communicating with other processors. Each processor, in this case, will store a part of the database and not the entire database. In the thesis, we present and compare two different strategies that follow this model.

CHAPTER THREE

SCALABLE PARALLEL ALGORITHMS AND SOFTWARE FOR LARGE-SCALE PEPTIDE IDENTIFICATION

3.1 Definitions and Problem Statement

Let *q* denote an unknown peptide sequence, which is fragmented using MS. A fragment of a peptide represents either a prefix or a suffix of *q*. MS generates an experimental spectrum of *q*, which is a plot of the relative abundance of its fragments binned by their m/z ratio values. The spectroscope also reports the m/z of the whole parent peptide *q*, denoted by *m(q)*.

Given *q*, a suffix or prefix of another (known) peptide sequence is said to be a *candidate* for *q* if the suffix's/prefix's m/z is $m(q) \pm \delta$ (where δ is a tolerance constant.)

An experimental spectrum for *q* is said to match with a candidate peptide if it can be shown that the candidate is most likely to generate a model spectrum similar to that of the experimental spectrum. This is achieved in *MSPolygraph* by generating two different spectra [Cannon *et al.,* 2005] — one a model spectrum for the candidate and the other being a spectrum generated for a random peptide — and then comparing both against the experimental spectrum. The result is a likelihood ratio score, and if the score is above a user-specified cutoff then the corresponding matching peptide is reported as a "*hit*".

Let $Q = \{q_1, q_2 \ . \ . \ . \ q_m\}$ denote a set of *m* input experimental spectra generated from *m* unknown peptides, and let $D = \{d_1, d_2 \ . \ . \ . \ d_n\}$ denote a database of *n* already known peptide sequences. Let $|d_i|$ denote the number of residues, i.e. sequence length in sequence d_i, and let $N = \sum_{i=1}^{n} |d_i|$. We use þ to denote the number of processors. The processors are labeled $P_0, P_1 \ . \ . \ . \ P_{p-1}$. For ease exposition, the terms "experimental spectrum" and "query" are used interchangeably.

3.1.1 The Peptide Identification Problem

Given sets Q and D, the peptide identification problem is to identify a list of at most τ top database hits for every input spectrum $q \in Q$, where τ is a user defined constant.

A value within [10...1,000] is used for τ in practice, although the software needs to treat is as a user-specified parameter.

3.2 Parallel Algorithms

As reviewed in chapter 2, all current algorithms and software including *MSPolygraph* assume that the entire database D can fit in the local memory of every processor. This assumption may no longer be practical as sequence databases continue to grow at exponential rates. To circumvent this scalability bottleneck, we designed and investigated two space optimal parallel algorithms. Our underlying approach in both of these algorithms is to partition the database evenly among the p processors, such that space-optimal storage is achieved at every processor. Distributing the database, however, introduces new challenges related to communication overhead and data locality. If a database sequence can generate a candidate for a given query, then it has to be made available to the processor handling that query. In the worst case, a query may need the entire database and such a worst-case is not far from practical expectations (as corroborated in our experiments, presented in chapter 4). If a query on processor P_i requires a database sequence that is resident in a remote processor P_j's memory, then there are two algorithmic design options:

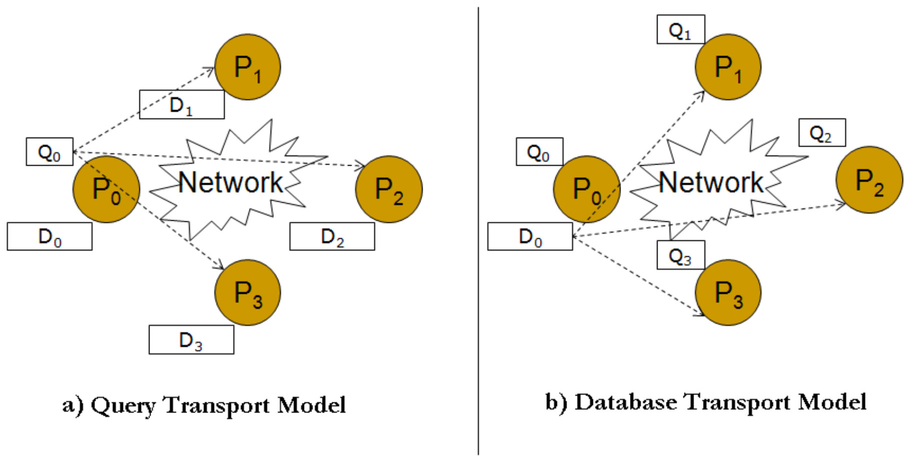

Figure 3.1 – a) Query Transport Model, where query set Q_0 is transported from P_0 to other processors. b) Database Transport Model where database D_0 is transported from P_0 to other processors.

i) (**Database transport model**) Communicate the database sequence from P_j to P_i over the interconnect network so that the query can be locally processed at P_i and

ii) (**Query transport model**) Communicate the query from P_i to P_j for remote query processing at P_j.

The query transport model can help especially when m is expected to be much smaller than n. However, the challenge with such a scheme is that a query can get processed at multiple processor locations, thereby necessitating the need for communicating the results to one root processor for merging. This merge step could create bottleneck for scaling, for large number of queries. On the other hand, the database transport model does not generate such kind of serialization issues as each processor takes full responsibility of a query, and does the processing locally. However, this model could potentially trigger large volumes of communications for large database sizes and could therefore introduce new challenges to maintain parallel scalability.

We chose the database transport model to develop our approach. The trick here is to efficiently mask the communication costs introduced by the transport of the database. The database and queries are distributed statically at the beginning. Hence, the model does not require master processor. In what follows, we propose two different algorithms that follow the database transport model. We call the algorithms simply as Algorithm A and Algorithm B.

3.3 Algorithm A

The pseudocode for this algorithm is shown in Figure 3.2. The first phase is the initialization phase which involves loading of the database and the queries. Since the database is distributed over all the processors involved, each processor P_i reads i^{th} (N/þ) part of the database. This is done by calculating the size of the database file and then every processor P_i loading the database, starting from offset ((N/þ)*i) till ((N/þ)*(i+1)) in the database file. Special care is taken to fully read the sequences at the processor boundaries. This way, the database is evenly distributed, guaranteeing its space complexity to be *O*(*N*/þ). Similarly, each processor then loads its *O*(*m*/þ) part of the queries into memory.

The next phase is query processing. In this phase, each processor processes its share of queries in parallel. Processing a query involves i) generation of candidates from the database sequences, ii) Generation of model spectrum for those candidates ii) matching queries against the spectra generated for candidates. For each query, list of up to τ top hits is maintained throughout the process, where τ is specified by the user. List of up to τ top hits per query

```
At Proc $P_i$:
A1) (Loading)
      $D_i \leftarrow P_i$'s share of database (size O( $N/p$ ))
      $Q_i \leftarrow P_i$'s share of $\sim m/p$ queries

A2) (Query Processing)
      $D_{recv} \leftarrow D_i$
      FOR $s \leftarrow 0$ to $p$ -1 DO
            $j = (i + s) \bmod p$
            IF $s > 0$, wait until $D_j$ is received in $D_{recv}$
            $D_{comp} \leftarrow$ copy $D_{recv}$
            $j_{next} = (j + 1) \bmod p$
            $D_{recv} \leftarrow$ Place non-blocking request for $D_{jnext}$
            Compute: Match $\forall q \in Q_i$ against $D_{comp}$
                              For each local query, keep a running list of the top
                              hits $\tau$ hits seen so far, and update in every iteration

A3) (Output reporting)
      Output the final top $\tau$ hits for every local query.
```

Figure 3.2 – Pseudocode for Algorithm A

is then output finally. This processing phase is implemented using þ iterations. During each iteration *s*, processor P_i fetches database from processor $P_{[(i+s)mod þ]}$, evaluates *all* the assigned queries against that part and generates a temporary list of at the most τ top hits, for each query. This list is then compared with the list from the previous iteration and top τ hits from both the lists are retained for next iteration. For the very first iteration this step is not required, as each processor starts by processing their queries against their locally stored database fragments. A processor then outputs final list of top τ hits after þ iterations.

As sequences have to be fetched from other processors, it could lead to two possible problems: i) Increase in wait time, and ii) Processors being busy serving other processors as the number of processors and hence requests grow more. To overcome these problems we use two techniques: i) *Use of one sided communication primitive MPI_Get(), which is a non-blocking call and does not involve the processor holding the data in communication. ii) Masking of communication with computation.* At any iteration *s*, processor P_i evaluates all its queries against D_j, where j = (i+*s*) mod þ. *Before* the queries are processed, a non-blocking request to receive the database portion for the next iteration is issued. At the first iteration, each processor P_i issues a pending request from processor $P_{(i+1)mod\ þ}$ and processes queries against its local part of the database. This overlapping of communication and computation is expected to mask the communication overhead incurred.

3.3.1 Maintaining the Top τ Candidates at Every Iteration

At any given processor, each iteration generates an arbitrary number of candidates along with their scores for every query from its local set. Since only top τ candidates are to be retained per query, a running list of top τ candidates is maintained and carried forward to every iteration. This running list is maintained as follows: the new batch of candidates generated at iteration s is merged with the old batch of τ hits carried over from the iteration $s\text{-}1$, and the top τ hits from that union is output as the new running list. Following are the steps taken at the end of each of the iteration s, to maintain the list.

M1) At iteration s, the running list of top τ candidates per query generated at the end of the previous iteration is concatenated with the new batch of candidates generated for that query at iteration s.

M2) A MinHeap data structure is then built of all these candidates, using the *BuildHeap()* procedure, based on the scores of the candidates.

M3) The function *DeleteMin()* is then called τ times. This gives us the new running list of top τ candidates for iteration s.

The above procedure is then carried out for each query.

3.4 Algorithmic Analysis of Algorithm A

3.4.1 Space Complexity:

Lemma: The space complexity of Algorithm A is $O(\frac{N+m}{p})$ at every processor.

Proof: The space complexity to store local queries is $O(\frac{m}{p})$. As for the database storage, each P_i keeps three $O(N/þ)$ buffers: i) D_i stores the local portion of the database; ii) D_{recv} is the communication buffer. It is written into and reused by the non-blocking MPI_Get() primitive at every iteration; and iii) D_{comp} is the buffer against which local queries are compared at any given iteration. This buffer is also reused over all the iterations. Therefore, the space complexity of this algorithm is $O(\frac{N+m}{p})$.

3.4.2 Runtime Complexity:

3.4.2.1 Computation Complexity

Steps A1 and A3 in Figure 3.2 collectively take $O(\frac{N+m}{p}+\frac{m}{p}+\tau)$ for input loading and output reporting. For step A2, let *r* be the total number of candidates evaluated against each query, on average. Let ϱ be the constant time it takes to compare each query against each candidate. Then, the total computation complexity for query comparison at every processor is $O(\frac{m}{p} \times r \times \varrho)$. Before that, each of t he n database sequences is checked to see whether it can generate candidates based on its m/z value. In addition, the cost of maintaining a running list of the top τ hits for every query is $O(\frac{r}{p}+\tau+\tau\log(\frac{r}{p}+\tau))$, where $O(\frac{r}{p}+\tau)$ is the time required for build_heap function and $O(\tau\log(\frac{r}{p}+\tau))$ is the time required for running the DeleteMin() function τ times. For *m*/þ queries over þ iterations and assuming $\frac{r}{p}\geq\tau$, the total time for merging would be $O(m(\frac{r}{p}+\tau\log\frac{r}{p}))$. Therefore, the overall computation complexity is: $O(\frac{N+m}{p}+\frac{m}{p}\times r\times\rho+\frac{m}{p}\times n+m\tau\log\frac{r}{p})$.

3.4.2.2 Communication Complexity

Let λ be the network latency and μ be the time to transfer one byte over the network. Then the total communication complexity is $O(\lambda\times þ+\mu\times N)$. However, due to masking, the net observed communication cost is expected to be significantly less, as demonstrated by our experiments (see chapter 4).

3.5 Algorithm B

Algorithm A reads in the entire database, although over p separated iterations. However, it may so happen that the local set of queries need only a subset of database sequences as candidates. Therefore, in such scenarios it may be wasteful to read in all *n* sequences at each processor. We present here an alternate algorithm based on this simple observation: Given a query *q*, its candidates can originate only from database peptides *d* such that $m(d) \geq m(q)$. This condition for a database sequence to be a candidate could be used to reduce the volume of database sequences for communication. The main idea of Algorithm B is to sort database sequences based on mass and then store the sorted array of sequences in a distributed manner. This allows us to fetch data only from a subset of processors. The algorithm, as shown in Figure 3.3, is as follows: Prior to query processing, the database is sorted in parallel based on the sequences' m/z values (step B2). The output is a non-decreasingly sorted array, generated such that each processor receives

$O(N/þ)$ characters from the sorted database. During subsequent query processing (B3), the sorted order could help pre-determine only that subset of processors which have sequences with candidates to offer the local batch of queries, implying that it is sufficient to restrict the communication to the identified subset. More specifically: Given a query *q*, its candidates can originate only from any database peptide *d* such that $m(d) \geq m(q)$. Therefore,

```
At Processor $P_i$

B1) (Loading)
      $D_i \leftarrow P_i$'s share of database (size $O(\frac{N}{þ})$)
      $Q_i \leftarrow P_i$'s share of $\sim\frac{m}{þ}$ queries
B2) (Sorting)
      Parallel sort $D_i$ based m/z, s.t., after sorting
      $D_i^s$ has its $O(\frac{N}{þ})$ fraction of the sorted database
B3) (Query Processing)
      Let $i' \leftarrow$ lowest ranked proc. w/ candidates for $Q_i$
      $D_{recv} \leftarrow$ Place non-blocking request for $D^s_{i'}$
      FOR {i', i'+1 … $p$-1} DO
            Wait until $D_j^s$ is received in $D_{recv}$
            $D_{comp} \leftarrow$ copy $D_{recv}$
            $j_{next} = (j + 1) \bmod p$
            Compute: Process $\forall q \in Q_i$ against $D_{comp}$
                  For each local query, keep a running list of the top hits $\tau$ hits seen so far, and
                  update in every iteration
B3) (Output reporting)
      Output the final top $\tau$ hits for every local query.
```

Figure 3.3 – Pseudocode for Algorithm B

if processor P_i computes $m(q)_{min} = \min_{j=1}^{|Qi|} |m(q_j)|$, then candidates for any query in Q_i can originate from only database sequences with m/z values $m(d) \geq m(q)_{min}$. Let P_i' be the lowest ranked processor which has database sequences in the sorted order satisfying this property. Then the sender group for P_i is the subset $\{P_i' \ldots P_{þ-1}\}$.

3.5.1 Algorithm for Parallel Sorting

We implemented the sorting step using parallel counting sort as the m/z values are bounded in practice ([1 . . . 300,000]). Our parallel sorting algorithm is as follows:

S1) Each processor computes the parent m/z value of each sequence in D_i. The processors then compute the global maximum of the m/z values (m/z_{max}) using the MPI Allreduce primitive.

S2) Each processor creates a local "count" array of size m/z_{max} in which it records the frequency occurrence of each m/z value in D_i. Subsequently, using the MPI_Allreduce primitive on the local count arrays, the processors compute a global count array, which they use as a reference to redistribute i) the sequences in D_i. Sequences are redistributed such that those with the same m/z are sent to the same processor, and the sum of the lengths of the sequences resulting in each processor is $O(N/þ)$. This data exchange is implemented using the MPI_Alltoallv primitive. Let D^s_i denote the output of sorting in processor P_i. Based on the partitioning pivots used in sorting, all processors store an array of þ tuples of the form ($begin_i$, end_i) to keep track of m/z boundary values at every other processor P_i. This tuple information can be used to determine the value of i'. The query processing step is almost identical to that of Algorithm A, with a minor addition: To accelerate identifying the range of queries that require any database sequence, we maintain the local query set Q_i also sorted by their parent m/z values and then use binary search.

3.6 Algorithmic Analysis of Algorithm B

3.6.1 Space Complexity

Lemma: The space complexity of Algorithm A is $O(\frac{N+m}{p}+p)$ at every processor.

Proof: Along with the buffer to store Q_i, this algorithm requires at most three of the following four database buffers, each of size $O(N/þ)$, at any given execution point: D_i, D^s_i, D_{comp}, and D_{recv}. This, along with the þ tuples stored at every processor for indexing yields a net space complexity of $O(\frac{N+m}{p}+p)$.

3.6.2 Runtime Complexity

3.6.2.1 Computation Complexity

Relative to Algorithm A, the only addition to computation in this algorithm is the computation performed during sorting. As we use integer sorting, this additive factor is only $O(n/p)$ which is dominated by $O(N/p)$. In addition, as only those database sequences with at least one candidate to offer for a given query are compared against that query, the term $O(\frac{m}{p} \times n)$ should be removed. Therefore, the overall computation complexity is: $O(\frac{N+m}{p} + \frac{m}{p} \times r \times \rho + m\tau \log \frac{r}{p})$.

3.6.2.2 Communication Complexity

The communication complexity of parallel counting sort is dominated by the cost of the MPI_Alltoallv primitive. As each processor sends and receives approximately N/p characters, this cost is $O(\lambda \times p + \mu \frac{N}{p})$. During query processing, each processor performs *at most* $p-1$ MPI Get function calls, and each such call fetches $O(N/p)$ characters. Therefore, the overall communication complexity of Algorithm B is $O(\lambda \times p' + \mu \times N')$, where $p' \leq p$ and $N' \leq N$.

3.7 Implementation

The code is written in C/MPI, and can be obtained by contacting one of the authors. In our implementation, a linear search as opposed to a binary search for queries was used in Algorithm B. Also, for the maintaining candidates list, instead of using a heap, two lists (current and previous iteration) corresponding to each query were combined and then sorted. Top τ candidates were then taken out for each query.

CHAPTER FOUR

EXPERIMENTAL RESULTS

All experiments were conducted on a 24-node, 192-processor Linux commodity cluster. Each node contains 8 2.33 GHz Xeon CPUs and a shared 8 GB RAM. The network interconnect is a gigabit Ethernet, and all nodes share an NFS mounted file system. To mimic a modest setting under most commodity clusters, we set the RAM usage limit to 1 GB RAM per MPI process.

4.1 Input Data

We tested our implementations on two collections of data: i) (Human) 88,333 human protein sequences; and ii) (Microbial) ~$2.65x10^6$ microbial protein sequences, both downloaded from NCBI GenBank. The human database was used for validating our results against *MSPolygraph*'s results as published in [Cannon *et al.*, 2005]. The microbial database was used for large-scale performance studies. To conduct scalability tests, we extracted arbitrary subsets of sizes 1K, 2K, 4K and so on from both databases. A set of 1210 spectra from peptides that were used as standards to weekly quality assessment purposes at PNNL was used as queries in all experiments. These peptides were obtained from human, cow and bacterial peptides. Table 4.1 shows details of input data.

4.2 Validation

The output produced by our approach is expected to be the same as *MSPolygraph*'s, as we internally use the same scoring functions and statistical modeling as *MSPolygraph*. Upon validation, we found that both algorithms A and B successfully reproduce *MSPolygraph*'s output on the human protein collection.

Table 4.1. Input dataset statistics

	Human	**Microbial**
#Protein Sequences	88,333	2,655,064
Total seq. length (in residues)	26,647,093	834,866,454
Avg. seq. length (in residues)	301.66	314.44

4.3 Performance Analysis of Algorithm A

The performance analysis was done using database of ~2.65x10^6 microbial protein sequences along with a collection of 1,210 human experimental spectra, used as queries. The runs were carried out on up to 128 processors.

4.3.1 Runtime Analysis

Table 4.2 shows the parallel run-time of Algorithm A over the entire range of input sizes and processor sizes tested. Although the asymptotic run-time is data-dependent (depends on the number of candidates evaluated for all queries), the practical expectation is that the run-time scales linearly with the database size. This expectation is consistent with our observations within each column of Table 4.2. The run-time growth as a function of processor size can be explained as follows: Our parallel runtime can be broken down into two parts: computation time and "*residual communication*" time. Residual communication time is defined as the time spent by the code waiting for the next batch of data, and is equal to the total communication time minus its portion masked by computation. In practice, it is only the residual communication that matters for the total time. Figure 4.1 shows the effect of masking communication with computation for input 100K. We observed that the mean$\pm$std. deviation of the ratio of residual communication to computation time to be 0.36 $\pm$0.11, for all processor sizes greater than 2 on all

Table 4.2. The Runtime for Algorithm A for Various Database and Processor Sizes. An entry ' – ' means the corresponding run was not performed.

Database size (*n*)	**Number of processors (*p*)**							
	1	**2**	**4**	**8**	**16**	**32**	**64**	**128**
1K	36.14	20.08	17.37	9.54	6.55	5.04	4.58	14.95
2K	66.85	34.87	31.58	16.09	9.37	6.14	5.18	8.69
4K	132.25	67.9	61.04	30.93	14.95	9.54	6.94	9.26
8K	255.02	131.19	116.15	55.62	28.70	16.64	9.46	10.86
16K	590.18	327.38	234.77	121.40	59.36	33.89	17.92	14.64
32K	1246.52	679.18	488.38	244.16	125.39	74.44	36.78	26.16
64K	2102.74	1240.47	1034.38	463.45	239.71	137.65	69.37	39.98
100K	3318.38	1963.77	1414.24	754.22	369.19	224.42	110.41	68.23
200K	7413.12	3837.21	3152.60	1530.59	804.84	438.72	221.86	119.29
400K	-	-	-	2894.21	1459.62	840.85	436.36	236.99
800K	-	-	-	5823.06	2953.05	1580.39	840.82	478.66
1M	-	-	-	7089.82	3564.05	1948.99	1014.79	583.38
2M	-	-	-	14322.90	7308.14	4167.30	2056.01	1100.21
2.6M	-	-	-	17431.66	9495.18	4535.70	2661.97	1382.60

input sizes. In other words, the overhead due to communication is approximately 25% of the total time even for larger processor sizes.

4.3.2 Speedup and Efficiency

The parallel speedup and efficiency are shown in Figure 4.2 and 4.3 respectively, for all input sizes. Because a run of Algorithm A at p = 1 is equivalent to the uni-worker run of *MSPolygraph*, the speedup values in Figure 4.2 represent *real* speedup. As expected, the speedup reduces after 64 processors for small inputs (1K to 8K), as the input becomes too small for the processor size. For input sizes 16K or more, we observed that the speedup is linear – i.e., it approximately doubles with doubling of processor size (with the only exception being when p is increased from 2 to 4).

Figure 4.3 shows the parallel efficiency. As shown, the efficiency is largely above 90% at p = 2, but reduces to ~50% at p = 4. Thereafter, it is maintained at roughly 50% until p = 128. Maintaining parallel efficiency as high as 50% for a wide range of input and processor sizes as shown is a significant result. The only anomaly observed is when processor size increases from 2 to 4: At p = 2, each processor has to communicate with just one other processor and masking of communication by computation in the first iteration ensures that the residual communication is practically negligible. However, at p = 4 each processor is required to communicate with three other processors, which is a

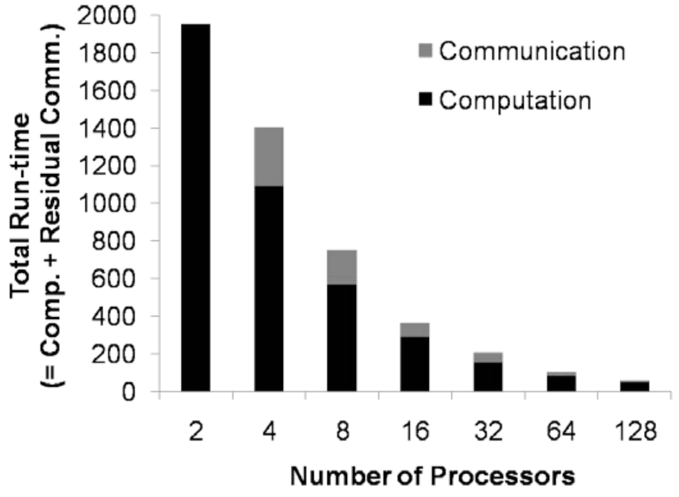

Figure 4.1 – A graph showing communication and computation time for 100K input

3X-fold increase in communication as compared to communication for þ = 2 case. Due to the high latency costs involved, the contribution of the residual communication to the total runtime increases from almost 0% to nearly 35%, affecting the efficiency and speedup. Except for this anomaly, the efficiency for all processor sizes between 4 and 128 is maintained at ~50% — implying strong scaling. To assess the positive effect of masking, we implemented a second version of the algorithm that does not mask communication with computation. Results showed that the masking technique reduces the total run-time by a factor of 72.75% ± 0.02%. For example, if the parallel run-time without masking for a given input is 100s, then the same analysis with masking will take only ~27.25s.

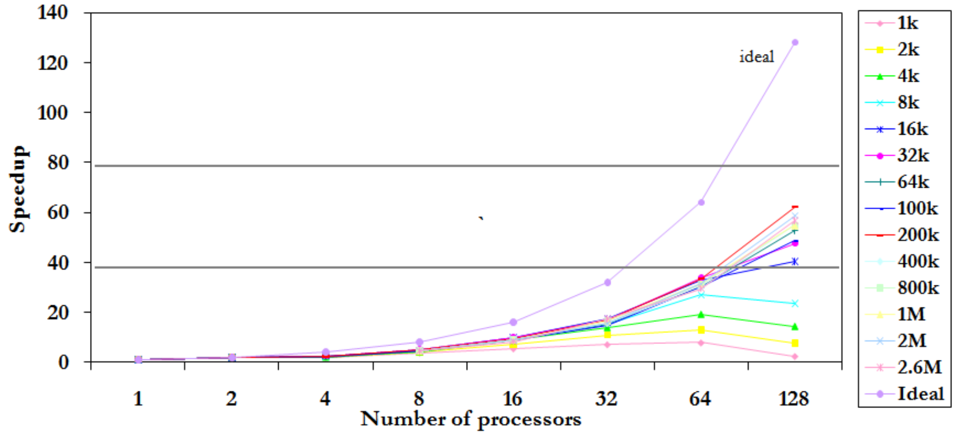

Figure 4.2 – Real Speedup of Algorithm A

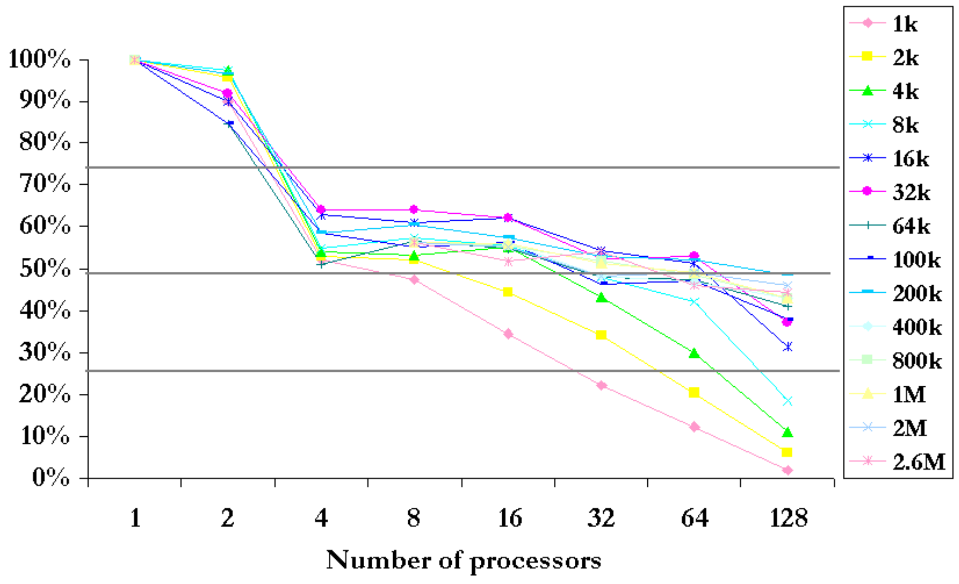

Figure 4.3 – Parallel Efficiency of Algorithm A

4.3.3 Candidates Evaluation Rate

Finally, we also measured the candidate evaluation rate of our algorithm. This result is shown in Table 4.3. From an application point of view, this is likely to be the most interesting performance measure as it directly conveys the effect of parallel processing on peptide identification. As shown, our implementation achieves linear scaling of the number of candidates processed every second with processor size.

4.4 Performance Analysis of Algorithm B

We analyzed Algorithm B's performance and observed that it was consistently outperformed by Algorithm A in speedup and efficiency. Table 4.4 shows a concrete example of this behavior, for 20K input. As shown in the Table, Algorithm B's speedup drops down to 10.44 for 64 processors after starting at 1, whereas the speedup for Algorithm A is equal to 31.02 for 64 processors.

Table 4.3. Number of candidates evaluated per second as a function of processor size, for 2.6x10^6 Microbial dataset

þ	**8**	**16**	**32**	**64**	**128**
#Candidates per sec.	41,429	76,057	159,220	271,294	522,331

Algorithm A outperforms Algorithm B in terms of runtime as well. For example, Algorithm A's runtime for 64 processors is 33.62s, while Algorithm B's runtime is 97.51s for the same run. Upon investigation, we found that this decline in speedup and runtime for Algorithm B was because the overhead due

Table 4.4. Comparative analysis of Algorithm A & Algorithm B for a database with 20K sequences

	Algorithm A		**Algorithm B**		
Þ	**Run-time(s)**	**Speedup**	**Run-time(s)**	**Speedup**	**Sorting Time(s)**
1	1043.14	1.00	1018.74	1.00	1.03
2	596.95	1.75	833.12	1.22	0.68
4	514.66	2.02	366.8	2.77	1.29
8	251.05	4.16	238.0	4.28	1.27
16	118.97	8.76	124.96	8.10	4.33
32	62.94	16.57	89.28	11.40	27.82
64	33.62	31.02	97.51	10.44	65.44

to its sorting step. The sorting step was found to be taking a major part of the total runtime, with increase in processor size (up to ~67% of total time). For example, as shown in the table, for 64 processors the sorting time was found to be 65.44s out of the total runtime of 97.51s. In addition to this, the input queries were such that each processor had to communicate and fetch database segments from a majority of the other $p - 1$ processors, thereby defeating the purpose of sorting. However, note that the set of 1,210 experimental spectra used in our experiments are from a human spectral database. Therefore, each spectrum is expected to result in the evaluation of an order of magnitude larger number of candidates than for a spectrum from, say a bacterial genome (see Figure 1.4). Because of this property, we expect that the sorting-based approach will better serve its purpose of reducing the computation workload when applied on spectra generated from less complex data classes (e.g., when spectra are from a known protein family or a bacterial genome).

CHAPTER FIVE

A NOVEL APPLICATION OF PEPTIDE IDENTIFICATION: SPECIES IDENTIFICATION FROM METAGENOMICS SAMPLES USING SPECTRAL DATA

Species identification is one of the fundamental problems in metagenomics. In this chapter, we present a novel application of peptide identification to identify species in metagenomics samples using MS/MS spectral data. To the best of our knowledge, our approach represents the first effort to use MS/MS spectral data for metagenomics species identification, as all other existing approaches use DNA-based information.

5.1 The Species Identification Problem

The goal of the species identification problem is to identify all the species that inhabit a given environmental microbial community.

5.2 Current Methods

All current methods for metagenomic species identification use random samples of DNA sequenced from the target community as the primary source of information. These methods can be broadly classified into three different categories [McHardy *et al.*, 2007]:

i) ***Phylogenetic marker methods:*** Here, the main idea is to use highly conserved DNA segments as phylogenetic markers to delineate a species present in a community. A widely used method of this class is the 16s rRNA based classification [Woese *et al.*, 1977], in which the differences observed among 16s rRNA gene derived from multiple species is used as a means to discriminate among species of the community.

ii) ***Composition based methods:*** This class of methods exploits the observation that different species have different DNA composition-based signatures such as short, fixed-length DNA oligomers (or substrings) [Abe *et al.,* 2003; Abe *et al.*, 2005; Deschavanne *et al.*, 1999; Nakashima *et al.* 1998; Sandberg et al., 2001; McHardy *et al.*,2007].

iii) ***Homology-based methods:*** In this approach [Huson *et al.*, 2007], homology-detection tools such as BLAST [Altschul *et al.*, 1990] are used to compare the community DNA against already sequenced genomes.

Even though the above approaches are frequently used, they have their respective limitations. For the 16s rRNA based method, it has been observed that the fragments having these marker genes tend to be less in number

[McHardy *et al.*, 2007]. For example, rRNA fragments are found to be present in only 0.071% of agricultural soil samples [Mavromatis *et al.*, 2007]. Similarly, only 0.06% of the fragments carry rRNA genes in the Sargasso Sea sample [Venter *et al.*, 2004].

For the homology detection and the compositional signature method, false contigs, which are artifacts of the assembly process, could lead to misleading results [McHardy *et al.*, 2007]. The false positive rate increases as the underlying community structure becomes increasingly complex and/or as the contig length becomes shorter. The latter is increasingly a concern with next-generation high throughput sequencing technologies, because the read lengths are much shorter than generated by traditional sequencing technologies. Furthermore, compositional signature methods have been observed to lack the ability to differentiate among sub-species [DeSantis *et al.*, 2006].

5.3 Our Approach to Species Identification using Mass Spectral Data

Despite the large amounts of proteomics data that are currently available, they are completely ignored by the current set of approaches toward metagenomics species identification. In this work, we propose a novel way to use proteomic data for metagenomics species identification. Our approach is based on the following simple observation: During peptide identification, the

MSPolygraph algorithm outputs a set of protein sequence hits from the database for each experimental spectrum. However, most of the protein sequences in public databases are already mapped and annotated to specific species from which they were originally sequenced. Therefore, the species source corresponding to an experimental peptide could be inferred from their respective database hits. This is the main idea of our approach. Henceforth, we refer to our approach as *SpecPolygraph*.

Using proteomics data for metagenomics species identification has two advantages:

a) It presents an alternative to currently known DNA-based methods for species identification, and could serve a complementary purpose to DNA-based methods; and

b) It also gives additional information about proteins that were expressed in the community, and therefore could directly lead to the understanding of community functions and interactions. This information is typically unavailable through DNA-based strategies.

Even though the idea of our approach to use proteomics data is simple, it has to be supplemented with a thorough statistical method in order to ensure high prediction accuracy. For this reason, we developed an iterative method whereby a null hypothesis is calculated using simulations and is evaluated against an alternate hypothesis corresponding to the output predictions made by *SpecPolygraph*. The comparison yields the basis for a p-value that is calculated as a measure of significance for each species identified.

5.3.1 Determining the Statistical Significance of *SpecPolygraph*:

Let Q denote a set of *m* experimental spectra collected from a given metagenomic community. Let *D* denote a database with *n* protein sequences. Let $q \in Q$. Let $S = \{s_1, s_2 \ldots s_k\}$ denote the set of all species represented in D. Let *p* denote the protein sequence which is the topmost hit for *q* as output by *MSPolygraph*; and let s_i denote the species corresponding to *p*.

(Alternate Hypotheis) The species s_i is present in the target metagenomic community.

(Null Hypothesis) The prediction that species s_i is present in the metagenomic community is random.

In what follows, we will describe a simulation strategy to generate the null hypothesis. The goal of each simulation is to randomly assign database matches, and hence species, to each individual experimental spectrum in *Q*. We achieved this by modifying the implementation of the original

MSPolygraph code as described below. For convenience, we refer to this modified implementation as *SimPolygraph*.

For each (peptide, sequence) match, the original *MSPolygraph* code computes a legitimate score. We discard this score and instead assign a random score to the match. Following this manipulation, the original *MSPolygraph* code takes over and sorts all the matches by their scores, and reports only the top τ number of hits for each experimental spectrum. Subsequently, the *SimPolygraph* implementation interprets the i^{th} significant hit as the prediction result for the i^{th} simulation. This way, predictions for τ independent simulations are assigned using just a single run of the original *MSPolygraph* code. In our experiments, we assigned tau to 1,000.

Using all the simulation results, we construct a matrix C of size $k \times \tau$, such that $C[i,j]$ stores the number of times the species s_i was observed during the j^{th} simulation. Alternatively, let c_i denote the number of times the species s_i was observed by *SpecPolygraph* (alternate hypothesis). Let $N_{sim}(i)$ denote the number of simulations in which species s_i was observed c_i times or less – i.e., the corresponding $C[i,j] \leq c_i$.

Then the p-value for species s_i is given by: $1 - (N_{sim}(i) / \tau)$.

The above expression could be understood as follows: Intuitively, if a given species is observed numerous times in a majority of simulations, it implies that the observation made by *SpecPolygraph* of that species is not significant (i.e., p-value approaches 1). On the other hand, if the species is rarely observed to be abundant during simulations, then the significance of *SpecPolygraph*'s prediction for that species could be labeled as statistically significant (i.e., p-value approaches 0).

5.3.2 The *SpecPolygraph* Algorithm

Using the above method to calculate the statistical significance for a given species prediction, we developed an iterative algorithm as shown in Figure 5.1. Within each iteration, all the species identified by the *SpecPolygraph* code are assigned individual p-values. Subsequently, only those species which have a p-value less than or equal to a user-defined cutoff (0.1 in our experiments) are retained for further evaluation in subsequent iterations. Furthermore, the sequences that correspond to species that have been eliminated in a given iteration have to be removed from the database, so that only those sequences from the retained species are allowed to compete in future iterations. The method iterates until there is one or no species left satisfying the p-value cutoff – i.e., there is no more competition among species.

```
SpecPolygraph ( Spectral Queries Q, Sequence Database D, $\tau$ , PvalueCutoff ) {

    1. $D_{curr} = D$;
    2. Let $S$ = set of all species in D and let $|S| = k$;
    3. REPEAT {
        a. $S_{sel} = \phi$;
        b. (Alternate Hypothesis)
            1. Run: MSPolygraph(Q, $D_{curr}$, $\tau$);
            2. Let $S_{pred}$ denote the set of species predicted from top hits of the above
               MSPolygraph run;
        c. (Null Hypothesis)
            1. Run: $C[1..k][1..\tau]$ = SimPolygraph(Q, Dcurr, $\tau$);
        d. FOREACH species $s_i \in S_{pred}$ DO
            1. Let $c_i$ denote the frequency of observing $s_i$ under Alternate Hypothesis;
            2. Compute P-value for $s_i$ as a function of $c_i$ and matrix $C$;
            3. IF p-value for $s_i \leq$ PvalueCutoff THEN
                a. $S_{sel} = S_{sel} \cup \{s_i\}$;
        e. FOREACH sequence $i \in D_{curr}$ DO
            1. Let $sp_i$ denote the species corresponding to $i$;
            2. IF $sp_i \notin S_{sel}$ THEN $D_{curr} = D_{curr} \setminus \{i\}$;
    } UNTIL ($|S_{sel}| \leq 1$)
}
```

Figure 5.1 – Algorithm for *SpecPolygprah*.

5.4 Experimental Results

We validated our new approach for species identification, *SpecPolygraph*, by comparing its results against three different benchmark data sets. In each of these benchmark data sets, there already exists information about their respective species composition, although based on DNA evidence (e.g., 16s rRNA) methods. Hence, these data sets provide a reasonable benchmark to test the outcome of species prediction through our novel proteomics-based approach. The spectral data sets, in the increasing order of species complexity, are as follows:

i) (*Shewanella DB*) By selecting 30,000 experimental mass spectra originally obtained from the species *Schewanella oneidensis MR-1*, we constructed an artificial community with just one species.

ii) (*ERSP-01*) This is a collection of 28,306 spectra corresponding to a metagenomic community containing the following species: *Shewanella oneidensis*, *Deinococcus radiodurans*, and *QC proteins*. The species are represented in varying proportions, the ratio of which is 4:1:0.5 respectively.

iii) (*ERSP-02*) This is a collection of 27,146 spectra corresponding to another metagenomic community containing the same set of species as in ERSP-01, although with a different ratio of distribution: 1:4:0.5 respectively.

In what follows, we present the results of running and evaluating the results of *SpecPolygraph* against each of the three benchmark data sets. For each run of *SpecPolygraph*, we used the entire collection of microbial proteins as the database and a p-value cutoff of 0.1 for filtering within each iteration. The microbial protein database was downloaded from the NCBI GenBank repository and as of May 2009, it contained ~2.65×10^6 protein sequences. The total number of species represented in this microbial DB (*k*) is 728.

5.4.1 Shewanella DB

Table 5.1 shows the number of species that were selected and advanced, after each iteration of the *SpecPolygraph* run. The method, after its first iteration (I0), detected a total of 229 species with a "significant" p-value ($\leq$ 0.1), of which 109 species had a p-value of 0. These 229 species advanced to the next iteration, and the process was repeated until termination condition. Table 5.1 shows the results for the first four iterations. There were four species identified after iteration I3 with significant p-value. Those four species and their respective p-values are as follows: i) *Shewanella sp. W3-18-1*: 0.0; ii) *Shewanella oneidensis MR-1*: 0.0; iii) *Kineococcus radiotolerans SRS30216*: 0.002; and iv) *Frankia alni ACN14a*: 0.085.

Since *Shewanella sp. W3-18*-1 and *Shewanella oneidensis MR-1* are closely related species, we continued with two more experiments, one by forcefully eliminating *W3-18*-1 and another by forcefully eliminating *oneidensis MR-1*. After subsequent iterations of these experiments, the *Shewanella oneidensis MR-1* outperformed all three species, and hence emerged the winner. This test confirmed the 100% specificity and sensitivity of *SpecPolygraph* for this benchmark data set.

Table 5.1. Results of *SpecPolygraph* on the Shewanella data set. The original number of sequences in the complete microbial database is 728.

	#Species with p-value <= 0.1	**#Species with p-value = 0**
I0	229	109
I1	57	27
I2	15	5
I3	4	2

5.4.2 ERSP-01

For this data set, the results of *SpecPolygraph* are shown in Table 5.2. In this experiment, the results were mixed. The species *Shewanella oneidensis* survived the first few iterations, but was eliminated out of contention at iteration I2. Instead, Shewanella sp. W3-18-1 progressed until iteration I3. Furthermore, the *QC proteins* was eliminated prematurely at iteration I0. On the positive side, *Deinococcus radiodurans R1* emerged the final winner over all iterations, with a p-value of 0. These results reveal the challenge faced by *SpecPolygraph* in maintaining sensitivity under circumstances where closely related species are competing against one another. It is also noteworthy that *SpecPolygraph* allowed the progression of a few other species which were not originally expected to be part of the ERSP-01 data set. This additional set could *potentially* represent species (or its close relatives) that were originally missed out by the DNA-based evidence, but has been discovered for the first time in this community using proteomics data.

5.4.3 ERSP-02

For the ERSP-02 data set, we followed a slight variant of the *SpecPolygraph* algorithm, in which the species that were allowed to progress to the next iteration were selected from the top 50 percentile, ranked by the p-values.

Table 5.2. Results of *SpecPolygraph* on ERSP-01 data set.

	#species with p-val ≤ 0.1	**#species with p-val = 0**
I0	228	62
I1	25	7
I2	2	2
I3	1	1

This variant is a more conservative way to retain as much competition as possible without drastically increasing the number of iterations, and thereby the computation cost.

The results of this modified experiment are shown in Table 5.3. It took 10 iterations (I0 through I9) for the program to complete. After iteration I8, the species selected were: i) *Frankia alni ACN14a*: 0.909; ii) *Deinococcus radiodurans* R*1*: 0.0; iii) *Shewanella sp. W3-18-1*: 0.0. This meant that *Deinococcus radiodurans R1* and *Shewanella sp. W3-18-1* advanced to the next iteration. At the end of the last iteration, *Deinococcus radiodurans R1* was identified as the winner. In this experiment too, the results of *SpecPolygraph* do not directly match the DNA-based benchmark expectations. However, the results are overwhelming encouraging from the point of view of the high potential of a novel, complementary technique for metagenomics species identification. The results demonstrate the high value that can be expected to be added by incorporating proteomics data into the process of metagenomics species identification.

Table 5.3. Results of *SpecPolygraph* on ERSP-02 data set.

	#Species with p-val = 0	**#Species Used in the Iteration**
I0	50	762
I1	14	381
I2	31	191
I3	10	96
I4	6	48
I5	4	24
I6	3	12
I7	3	6
I8	2	3
I9	1	2

CHAPTER SIX

CONCLUSION AND FUTURE WORK

As public molecular databanks continue to be flooded with experimentally acquired data, significant scalability challenges are imminent in peptide identification. The current suite of software tools, however, are not designed to meet the constantly increasing computation demands owing to the increased data size and/or growing qualitative requirements. The state of analysis is further exacerbated by the growing interest among the research community for increasingly complex projects. For example, in 2007, a single project that studied ocean metagenomics [Yooseph *et al.*, 2007] added over $17x10^6$ ORFs/peptides to the public databases.

The primary strength of the approaches developed as part of this thesis is their combined effectiveness in addressing all three factors of critical importance to the peptide identification problem: space, time, and quality. Another strength is its design simplicity, thereby laying the foundation for further improvements and easy extensions.

6.1 The Significance of the Space-Optimality Result

The space-optimality result is significant in that it will allow application scientists to analyze very large input sizes that were beyond the reach of previously used software. Our new algorithm overcomes this major scalability limitation by exploiting the vast, aggregate memory easily available from large-scale distributed memory supercomputers. For example, we were able to store and analyze 2.65×10^6 sequences using as little as 8 processors, with 1 GB RAM available to each processor. It is to be noted, however, the application of our approach is expected to be better suited where the data size does not fit into a single or few processor's aggregate memory. For small inputs that fit within a processor's memory, the older version of *MSPolygraph* is more appropriate because it will output the same result with no added communication delays. For medium and larger range inputs, however, it could be worth exploring an extension of our approach in which processors can divide themselves into smaller sub-groups, where the database is partitioned within each sub-group and the query set is partitioned across sub-groups.

6.2 Is Sorting Necessarily an Overhead?

A dominant fraction of the query processing time is spent on generating candidates on-the-fly. Each query, in practice, may require generation of hundreds of thousands to even millions of candidates (as shown in Figure 1.4). From this perspective, it may be worth exploring an alternative strategy in which already generated candidates, and not the database sequences, are stored in-memory and are communicated on-demand to compute nodes. This strategy has the potential to drastically reduce the overall computation time, as it cuts down the need to generate candidates on-the-fly. However, the candidates generated are orders of magnitude times the actual database size. Current approaches are not designed to store such large magnitudes of candidates in memory. Our algorithm, because of its space-optimality, makes the investigation of this alternative approach feasible. Furthermore, the sorting version of our approach (Algorithm B) could prove more useful under this setting. The idea here is to store candidates in distributed and parallel sorted fashion. Each processor then fetches candidates from only those processors which contain candidates within the desired range dictated by the masses of the local query set. To further improve performance, the queries also could be stored sorted (by their masses) and this could help avoid redundant fetching of data.

6.3 MapReduce Based Implementation

The approach followed in our current implementation uses static query distribution. In practice, however, queries could take varying amounts of processing time, depending upon the number of candidates generated for matching. In this regard it would be worth exploring a MapReduce [Dean *et al.*, 2004] implementation. The MapReduce paradigm is a new, emerging model of parallel computing that is rapidly becoming the de facto standard for data-intensive cloud computing frameworks. Several open source libraries already exist for this model, and all of them provide a robust support for handling dynamic load balancing and fault tolerance. The model is also well suited for supporting Software-as-a-Service (SaaS) and coupled with virtualization, could be readily deployed inside clouds for scientific computing. Therefore, developing a MapReduce based implementation would allow us to simply inherit, without duplicating, the already available strengths of the MapReduce paradigm for peptide identification. However, developing a MapReduce based implementation entails algorithmic re-engineering.

A typical MapReduce paradigm consists of a master processor and two types of worker processors, called *mapper* and *reducer*. The *mapper* workers process input and emit a <key, value> pair, for each input unit. These pairs are then processed by the *reducer* processors, which gather results corresponding to the same key from across *mapper* processors. A dedicated master processor handles load balancing and task allocation.

There could be different ways to assign roles to *worker* processors. One strategy could be to assign the task of generating and matching candidates to the *mapper* nodes, while the *reducer* nodes are responsible for gathering and outputting the list of top candidates for each query. An alternative strategy is not to have *reducer* processors. Instead, all the worker processors are used as *mapper* nodes which completely evaluate the assigned queries.

6.4 Metagenomics Species Identification

Even though protein sequences and mass spectral libraries are being amassed at overwhelming rates, current methods hardly allow us to exploit and tap into these repositories. To this end, the *SpecPolygraph* proposed as part of this thesis, is a novel and important step which could close this critical data-to-knowledge gap. It could serve as a viable, complementary approach to the other DNA-based methods, and also has the added advantage of shedding new light into the functional space of the metagenomics communities under study. However, more experiments will have to be carried out to fine tune the prediction accuracy of the method. One way to achieve this qualitative improvement goal, is to study both techniques proposed (i.e., using a p-value cutoff against using a top percentile) in tandem. It will also be interesting to investigate the possibility of assigning abundance to species using p-values and hit-counts (frequency).

6.5 Conclusion

In this thesis, we presented the design and development of a new parallel algorithm for conducting large-scale peptide identification using mass spectrometry data. The approach proposed here is better equipped than any other contemporary software tool for meeting the scalability demands of the peptide identification application. The highlights of our new algorithm are its space-optimality, the ability to maintain runtime efficiency, and its incorporation of accurate statistical models for improved accuracy. This approach has enabled us to begin conducting a full-scale application. We also presented here a novel application of peptide identification, namely for the purpose of species identification in unsequenced microbial communities. The species identification problem is one of the most pressing and challenging problems in metagenomics. Our method, called *SpecPolygraph,* for the first time uses MS/MS spectral data for species identification. Our experiments demonstrate the promising qualitative aspects and the high potential in incorporating proteomic data for this novel purpose.

In conclusion, we believe that the contributions made by this thesis toward peptide identification and species identification have laid out a strong foundation for the proteomics community to benefit from parallel processing and advance overall scientific pursuit.

BIBLIOGRAPHY

1. Abe T, Kanaya S, Kinouchi M, Ichiba Y, Kozuki T, Ikemura T: Informatics for unveiling hidden genome signatures. *Genome Research*, 13:693-702, 2003.
2. Abe T, Sugawara H, Kinouchi M, Kanaya S, Ikemura T: Novel phylogenetic studies of genomic sequence fragments derived from uncultured microbe mixtures in environmental and clinical samples. *DNA Research*, 12:281-290, 2005.
3. Altschul S., Gish W., Miller W., Myers E., Lipman D. Basic local alignment search tool. *Journal of Molecular Biology*, 215 (3): 403–410, 1990.
4. Boeckmann B., Bairoch A., et al. The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Research*, 31:365–370, 2003.
5. Benson D., Karsch-Mizrachi I., Lipman D., Ostell J., Rapp B., Wheeler D. GenBank. *Nucleic Acids Research*, 35:D21–D25, 2007.
6. Bjornson R., Carriero N., Colangelo C.,Shifman M., Cheung K., Miller P.,Williams K. X!!Tandem, an improved method for running X!Tandem in parallel on collections of commodity clusters. *Journal of Proteome Research*, 7:293–299, 2008.
7. CAMERA: Community Cyberinfrastructure for Advanced Marine Microbial Ecology Research and Analysis, *http://camera.calit2.net/*, Last date accessed: Nov. 11, 2009
8. Cannon W., Jarman K., Webb-Robertson B., Baxter D., Oehmen C., Jarman K., Heredia-Langner A., Auberry KJ., Anderson G. Comparison of probability and likelihood models for peptide identification from tandem mass spectrometry data. *Journal of Proteome Research*, 4:1687–1698, 2005.
9. Chen T., Kao M., Tepel M., Rush J., Church G. A Dynamic Programming Approach to De Novo Peptide Sequencing via Tandem Mass Spectrometry. *Journal of Computational Biology*, 8(3):325-337, 2001.
10. Craig R. and Beavis R. A method for reducing the time required to match protein sequences with tandem mass spectra. *Rapid communications in Mass Spectrometry*, 17(20):2310–2316, 2003.
11. Craig R. and Beavis R. TANDEM: matching proteins with tandem mass spectra. *Bioinformatics*, 20(9):1466–1467, 2004.

12. Dancík V., Addona T., Clauser K., Vath J., Pevzner P. De novo peptide sequencing via tandem mass spectrometry. *Journal of Computational Biology*, 6(3/4):327–342, 1999.
13. Dean J. and Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters. *Symposium on Operating Systems Design and Implementation* (OSDI '04)
14. DeSantis T., Hugenholtz P., Larsen N., Rojas M., Brodie E., Keller K., Huber T., Dalevi D., Hu P., Andersen G., a chimera-checked 16S rRNA gene database and workbench compatible with ARB. *Applied and Environmental Microbiology*, 72:5069-5072, 2006.
15. Deschavanne PJ, Giron A, Vilain J, Fagot G, Fertil B: Genomic signature: characterization and classification of species assessed by chaos game representation of sequences. *Molecular Biology and Evolution*, 16:1391-1399, 1999.
16. Duncan D., Craig R., and Link A. Parallel tandem: a program for parallel processing of tandem mass spectra using PVM or MPI and X!Tandem. *Journal of Proteome Research*, 4(5):1842–1847, 2005.
17. Gropp W., Lusk E., and Thakur R., Using MPI-2: Advanced Features of the Message-Passing Interface. *MIT Press*, 1999
18. Huson D,Auch A, Qi J., and Schuster S. Schuster SC: MEGAN analysis of metagenomic data. *Genome Ressearch*, 17:377-386, 2007.
19. Liu C., Song Y., Yan B., Xu Y., Cai L. Fast de novo peptide sequencing and spectral alignment via tree decomposition. *In Pacific Symposium on Biocomputing*, volume 11, pages 255–66, 2006.
20. McCormack A., Eng J. and Yates J. An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. *Journal of the American Society for Mass Spectrometry*, 5(11):976–989, 1994.
21. McHardy A. and Rigoutsos I. What's in the mix: phylogenetic classification of metagenome sequence samples. *Current Opinion in Microbiology*, 10:499–503, 2007.
22. McHardy A., Garcia-Martin H., Tsirigos A., Hugenholtz P., Rigoutsos I. Accurate phylogenetic classification of variable length DNA fragments. *Nature Methods*, 4:63-72, 2007
23. Nakashima H, Ota M, Nishikawa K., Ooi T. Genes from nine genomes are separated into their organisms in the dinucleotide composition space. *DNA Research*, 5:251-259, 1998.
24. Perkins D., Pappin D., Creasy D., Cottrell J. Probability based protein identification by searching sequence databases using mass spectrometry data. *Electrophoresis*, 20(18):3551–3567, 1999.

25. Pevzner P., Mulyukov Z., Dancik V., Tang C. Efficiency of database search for identification of mutated and modified proteins via mass spectrometry. *Genome Research*, 11(2):290–299, 2001.
26. Sandberg R, Winberg G, Branden CI, Kaske A, Ernberg I, Coster J: Capturing whole-genome characteristics in short sequences using a naive Bayesian classifier. *Genome Research*, 11:1404-1409, 2001
27. Staden, R. "A strategy of DNA sequencing employing computer programs", *Nucleic Acids Research*. 6(7):2601-2610, 1979.
28. Taylor J. and Johnson R. Sequence database searches via de novo peptide sequencing by tandem mass spectrometry. *Rapid Communications in Mass Spectrometry*, 11:1067–1075, 1997.
29. Mavromatis K., Ivanova N., et al. Use of simulated data sets to evaluate the fidelity of metagenomic processing methods. *Nature Methods*, 4:495-500, 2007.
30. Venter J., Remington K. et al. Environmental genome shotgun sequencing of the Sargasso Sea. *Science*, 304:66-74, 2004.
31. Woese C. and Fox G. Phylogenetic structure of the prokaryotic domain: the primary kingdoms. *Proceedings of the National Academy of Sciences*, 74:5088-5090, 1977.
32. Yooseph S., Sutton G., et al. The Sorcerer II Global Ocean Sampling Expedition: Expanding the Universe of Protein Families. *PLoS Biology*, 5:e16, 2007.