Parallel Algorithms For Real-Time Peptide-Spectrum Matching

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Jian Zhang

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Abstract

Tandem mass spectrometry is a powerful experimental tool used in molecular biology to determine the composition of protein mixtures. It has become a standard technique for protein identification. Due to the rapid development of mass spectrometry technology, the instrument can now produce a large number of mass spectra which are used for peptide identification. The increasing data size demands efficient software tools to perform peptide identification.

In a tandem mass experiment, peptide ion selection algorithms generally select only the most abundant peptide ions for further fragmentation. Because of this, the low-abundance proteins in a sample rarely get identified. To address this problem, in [16], the authors develop the notion of a 'dynamic exclusion list', which maintains a list of newly selected peptide ions, and it ensures these peptide ions do not get selected again for a certain time. In this way, other peptide ions will get more opportunity to be selected and identified, allowing for identification of peptides of lower abundance. However, a better method is to also include the identification results into the 'dynamic exclusion list' approach. In order to do this, a real-time peptide identification algorithm is required.

In this thesis, we introduce methods to improve the speed of peptide identification so that the 'dynamic exclusion list' approach can use the peptide identification results without affecting the throughput of the instrument. Our work is based on RT-PSM, a real-time program for peptide-spectrum matching with statistical significance [40]. We profile the speed of RT-PSM and find out that the peptide-spectrum scoring module is the most time consuming portion.

Given by the profiling results, we introduce methods to parallelize the peptide-spectrum scoring algorithm. In this thesis, we propose two parallel algorithms using different technologies. We introduce parallel peptide-spectrum matching using SIMD instructions. We implemented and tested the parallel algorithm on Intel SSE architecture. The test results show that a 18-fold speedup on the entire process is obtained. The second parallel algorithm is developed using NVIDIA CUDA technology. We describe two CUDA kernels based on different algorithms and compare the performance of the two kernels. The more efficient algorithm is integrated into RT-PSM. The time measurement results show that a 190-fold speedup on the scoring module is achieved and 26-fold speedup on the entire process is obtained. We perform profiling on the CUDA version again to show that the scoring module has been optimized sufficiently to the point where it is no longer the most time-consuming module in the CUDA version of RT-PSM.

In addition, we evaluate the feasibility of creating a metric index to reduce the number of candidate peptides. We describe evaluation methods, and show that general indexing methods are not likely feasible for RT-PSM.

Acknowledgements

I would like to express my sincere appreciation to my supervisors Dr. Ian McQuillan and Dr. Fangxiang Wu for supporting me over my study and research, and for their encouragement and guidance throughout my studies.

I would like to give special thanks to Dr. Theodore Kim for providing me with the NVIDIA CUDA development environment, for agreeing to be on my thesis committee, and for his valuable comments and suggestions. Without his generous help, this work would not be done.

I would like to thank Dr. Tony Kusalik and Dr. Daniel Teng for agreeing to be on my thesis committee and for their guidance and support.

I would like to express my deepest gratitude to my family for all of their support and love.

Contents

Permission to Use i

Abstract ii

Acknowledgements iii

Contents iv

List of Tables vi

List of Figures vii

List of Abbreviations viii

1 Introduction 1
- 1.1 Background . . . 2
  - 1.1.1 Tandem Mass Spectrometry . . . 2
  - 1.1.2 Peptide Sequencing and Peptide Identification . . . 3
  - 1.1.3 *De novo* Sequencing and Database Searching . . . 5
- 1.2 Motivation and Objectives . . . 7
  - 1.2.1 Motivation . . . 7
  - 1.2.2 Objectives . . . 8
- 1.3 Thesis Overview . . . 9

2 RT-PSM: A Real-Time Program For Peptide-Spectrum Matching 10
- 2.1 Introduction . . . 10
- 2.2 Function Modules of RT-PSM . . . 11
  - 2.2.1 Experimental Spectrum Preprocessing . . . 11
  - 2.2.2 Candidate Peptides Selection . . . 12
  - 2.2.3 Peptide-Spectrum Match Score . . . 12
  - 2.2.4 Statistical Significance Score . . . 14
- 2.3 Performance analysis of RT-PSM . . . 16

3 Parallelizing Peptide-Spectrum Matching Using SIMD Instructions 21
- 3.1 Parallel Computing in MS/MS Spectrum Identification . . . 21
- 3.2 Overview of SIMD technology . . . 22
  - 3.2.1 Introduction . . . 22
  - 3.2.2 Intel MMX and SSE instructions . . . 23
- 3.3 Algorithm and Implementation . . . 25
  - 3.3.1 Comparison Operations in Peptide-Spectrum Matching . . . 25
  - 3.3.2 Parallel Algorithm . . . 26
- 3.4 Evaluation and Results . . . 32

4 Parallelizing RT-PSM with NVIDIA CUDA Technology 35
- 4.1 Introduction . . . 35
  - 4.1.1 Hardware Architecture of CUDA-capable GPUs . . . 35
  - 4.1.2 Programming and Execution Model . . . 36
  - 4.1.3 CUDA Memories . . . 39
- 4.2 Method . . . 39
  - 4.2.1 Data Decomposition . . . 39

4.2.2 CUDA Kernels . . . 40
4.2.3 Kernel Optimization . . . 43
4.2.4 Integration in RT-PSM . . . 45
4.3 Results and Discussion . . . 45
4.3.1 Experimental Setup and Dataset . . . 45
4.3.2 Speedup on the Similarity Score and the Kernel Comparison . . . 46
4.3.3 Speedup on RT-PSM and New Profiling Result . . . 46
4.3.4 Instruction and Memory Bandwidth Analysis . . . 47

5 Indexing the Peptide Database With Metric Data Structures 51
5.1 Similarity Distance function . . . 52
5.2 Evaluation of Triangle Inequality Based Methods . . . 53
5.2.1 Introduction of Pruning Rules and Evaluation Method . . . 53
5.2.2 Evaluation Experiment and Results . . . 54
5.3 Evaluation of SASH . . . 54
5.3.1 Introduction of SASH . . . 54
5.3.2 Evaluation of SASH . . . 56

6 Conclusion and Future Work 58
6.1 Conclusion . . . 58
6.2 Future Work . . . 59

References 60

List of Tables

2.1 These ions are involved in the similarity score in RT-PSM. 12

3.1 Microprocessors with SIMD technology (modified from [31]). 22

4.1 CUDA Memories 39
4.2 The difference of occupancy causes the difference of performance. 43
4.3 Different kernels have different memory allocation schemes. 44
4.4 Both kernels are tested on an GTX 285 graphics card. 46
4.5 The run time comparison illustrates the substantial speedup. 46
4.6 Estimated compute-bounds and bandwidth-bounds. 50

5.1 Results of SASH Evaluation 56

List of Figures

1.1 A typical experiment procedure of tandem mass spectrometry 3
1.2 Fragmentation patterns and ion types 4
1.3 A typical tandem mass spectrum consists of many peaks (modified from [2]). 4

2.1 The flowchart illustrates the process of a MS/MS spectrum identification in RT-PSM. 17
2.2 Function modules of RT-PSM are profiled using this procedure. 18
2.3 Profiling results of RT-PSM 19

3.1 Packed single-precision floating point operation 23
3.2 The Peptide-Spectrum Matching Matrix 26
3.3 Vectors in parallel to theoretical ions. 27
3.4 Vectors in parallel to peaks. 28
3.5 The number of comparison operations can be reduced using column vectors. 30
3.6 Data flow to compute the b_x vector. 31
3.7 Speed of original RT-PSM and parallel RT-PSM 33
3.8 Profiling results of SIMD RT-PSM 34

4.1 The design of GPU and CPU differs. 36
4.2 CUDA architecture 37
4.3 A kernel is executed as a grid of thread blocks (modified from [8]). 38
4.4 Data decomposition scheme. 40
4.5 Speedup factors with respect to CPU version for both kernels are reported. 47
4.6 Parallel RT-PSM using CUDA technology achieves 26-fold speedup. 48
4.7 CUDA Profiling Results 49

5.1 Pruning Example 53
5.2 Frequency plots of distances between pairs of spectra 55

List of Abbreviations

RT-PSM	Real-Time Peptide-Spectrum Matching
MS	Mass Spectra
MS/MS	Tandem Mass Spectra
SIMD	Single-Instruction Multiple-Data
SIMT	Single-Instruction Multiple-Thread
PSM	Peptide-Spectrum Matching
RAM	Random-Access Memory
VP	Vantage Point
k-NN	k-Nearest Neighbours
SASH	Spatial Approximation Sample Hierarchy
PVM	Parallel Virtual Machine
MPI	Message Passing Interface
MMX	MultiMedia eXtension
SSE	Streaming SIMD Extension
IA	Intel Architecture
FPU	Floating Point Unit

Chapter 1

Introduction

Proteins are complex and important molecules that carry out the tasks of life. They play crucial roles in almost every biological process. They catalyze reactions in living organisms, transport energy, oxygen and nutrients, and provide the mechanisms for, communication within and between cells, movement, structural support, storage and defense against outside invaders [20].

Every protein is initially formed as a chain of amino acids. There are twenty standard amino acids that are used to make up proteins. Amino acids are linked end-to-end during protein synthesis by peptide bonds. A *peptide bond* is formed between two amino acids when the carboxyl group of one amino acid reacts with the amine group of the other one, thereby losing a molecule of water (H_2O). This process is repeated as the chain elongates. As a result, the amino group of the first amino acid of a peptide chain and the carboxyl group of the last amino acid remain free. The end with the amino group is called *amino terminus* (N-Terminus) and the other end with the carboxyl group is called *carboxyl terminus* (C-Terminus) [5]. Then, protein sequencing involves trying to calculate the sequence of the twenty amino acids from one terminus to the other.

The ability to determine the amino acid sequence of a protein is very important. The amino acid sequence of a protein can be compared to other proteins to find similarity in sequence and infer similarity in function. Moreover, we can computationally predict the three-dimensional structure of a protein from its amino acid sequence to try to infer its function. In addition, the identication, quantification and characterization of proteins in a given sample can offer a great deal of valuable information for modern pharmaceutical research [25]. For example, the identification of proteins can allow researchers to determine whether an organism has a genetic disease. A genetic disease can be caused by a mutant protein, which has a slightly different composition from the normal protein.

Determining the sequence of amino acids in proteins is quite difficult. It was not until the 1950s that the first method for sequencing proteins was developed by Pehr Edman. The method is called Edman Degradation and it is one of the two dominant methods to sequence proteins. The other method is mass spectrometry, which has been developing in recent years as new techniques and increasing computing power have facilitated it. In this thesis, we will focus exclusively on mass spectrometry for protein sequencing.

1.1 Background

1.1.1 Tandem Mass Spectrometry

Nowadays, *tandem mass spectrometry* (MS/MS) has become a widespread experimental tool for protein sequence analysis due to its speed and high sensitivity. Information from the protein components in complex biological mixtures can be obtained efficiently by tandem mass spectrometry.

We will only briefly describe the experimental method, as it is quite complex. Please see [37] for a more thorough investigation of mass spectrometers. Although the procedures of proteomic mass spectrometric experiments may be slightly different from each other, a typical procedure can be divided into four main steps, as shown in Figure 1.1. In the first step, protein mixtures are digested into peptides of suitable size for mass spectrometric analysis using site-specific proteases (usually Trypsin, which cleaves proteins mainly at the carboxyl side of the amino acids lysine or arginine). Peptides are short polymers of amino acids linked by peptide bonds. Generally, they are much shorter than proteins. In the second step, the generated peptides in the mixture are separated using reversed-phase high-performance liquid chromatography before entering the mass spectrometer. In the third step, these peptides are ionized via electrospray ionization. The spectrometer scans the peptide ions and the mass spectra of the peptides (MS) are captured. In the last step, the spectrometer selects some of the detected peptides, fragments them further and their so-called tandem mass spectra (MS/MS) are collected. Two kinds of mass spectra are measured in the experiment. One is the spectrum of peptides and the other is the spectrum of peptide fragments.

From a computational perspective, the final result of the experimental MS/MS pipeline, which is the MS/MS spectra, is used as input. They are used in an attempt to identify a peptide sequence and these peptides are used to attempt to identify the proteins.

A mass spectrometer does not measure the mass of a molecule directly. It measures the mass-to-charge ratio (m/z) of the fragmented peptide ions. In order to infer the sequence of the peptide given by a peptide MS/MS spectrum, we need to consider the manner in which peptides can be sheared by a mass spectrometer. This shearing is nondeterministic, and can produce different fragment patterns and also different ions. Figure 1.2 illustrates how a peptide with four amino acids can fragment into different fragment ions. For example, if the fragmentation occurs at the peptide bond (between the C and N atoms), the left component makes up what is called a b-ion and the right component forms what is referred to as a y-ion. The corresponding subscript i of the b_i- or y_i-ion indicates the number of amino acid residues in the fragment. In Figure 1.2, the letter 'R' represents the side chain which varies on different amino acids.

Generally, a peptide can fragment into two parts at any of the labelled sites to generate various

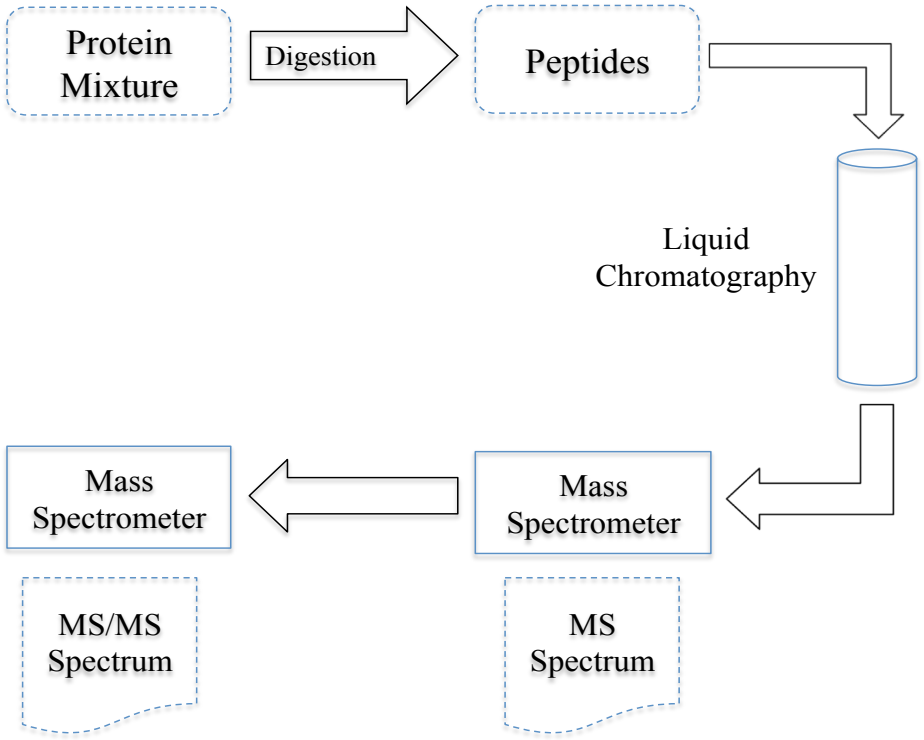

Figure 1.1: A typical liquid chromatography tandem mass spectrometry experiment procedure can be divided into four steps.

types of fragments ions. The common fragment ions are grouped into two classes: N-terminal ions and C-terminal ions. The N-terminal fragments include a, b and c ions and x, y and z ions belong to the C-terminal fragments. In Figure 1.2, they are $a_i,\ b_i,\ c_i,\ x_i,\ y_i,\ z_i$ (for $i = 1, 2, 3$). A typical MS/MS spectrum of a peptide can be represented as a histogram that consists of many peaks (see Figure 1.3). Each peak represents one type of fragment ion of one of the peptides in the experimental sample. The position of each peak on the horizontal axis indicates the mass to charge ratio (m/z value). The intensity of the fragment ion is represented by the height of the peak.

1.1.2 Peptide Sequencing and Peptide Identification

We denote the set of the 20 amino acids by AA. A peptide P consists of a sequence of n amino acids, $P = a_1...a_n$. Let the molecular mass of an amino acid residue (i.e. the amino acid losing a water molecule in the process of forming a peptide bond) be $m(a), a \in AA$. The twenty specific amino acid residues and their masses can be found in [37]. Then the mass of the whole peptide P is $m(P) = \sum m(a_i) + 18$ Daltons where 18 Daltons is the mass for an extra H_2O in the peptide.

Let S_q be an experimental MS/MS spectrum. Then $S_q = \{(q_1, h_1), ..., (q_m, h_m)\}$ consists of a set of m/z values of fragmented ions and their corresponding intensities. The problem of *peptide*

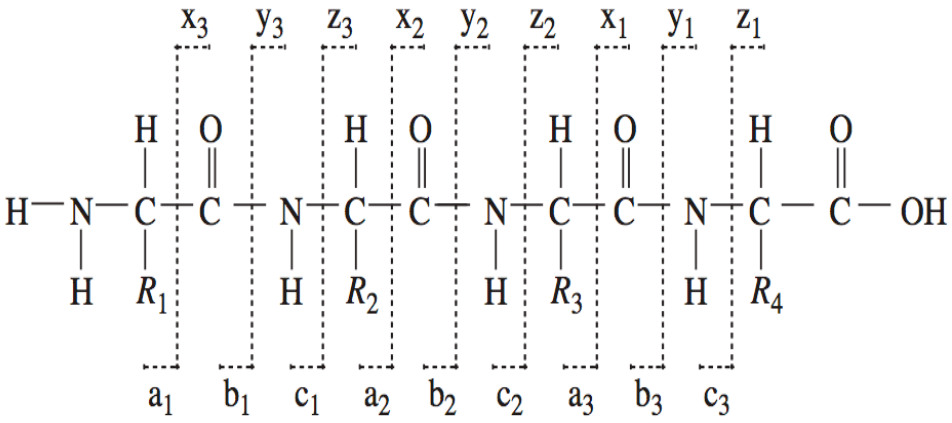

Figure 1.2: The fragmentation of a four-residue peptide in tandem mass spectrometry can occur between any of the labelled bonds (modified from [26]).

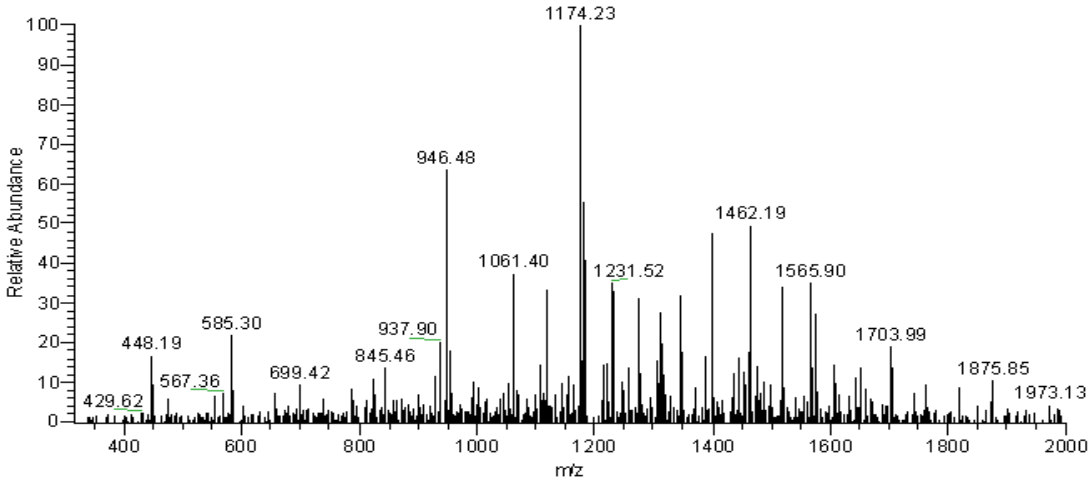

Figure 1.3: A typical tandem mass spectrum consists of many peaks (modified from [2]).

sequencing is to attempt to derive the peptide sequence P from S_q and $m(P)$.

Ions can be charged in various ways in an MS/MS experiment and generate a few different peaks. Fortunately, there are standard methods to convert all the peaks for multiply charged ions to their singly charged equivalents [37]. Then, these ions can be treated as singly charged ions. Therefore, in this thesis we focus on the ions that have a charge of 1. In this case, we consider the m/z value of an ion to be the same as the mass of the ion.

We use a set of numbers $\triangle = \{\gamma_i : \gamma \in \{a, b, c, x, y, z\}, i \in \mathbb{N}\}$ to denote fragmented ions. In tandem mass spectrometry, the *theoretical spectrum* of peptide P can be calculated by extracting all possible fragmented ions from the peptide sequence and computing the mass of each ion. Due to the sequential structure of the sequence of amino acids and the fragment patterns of ions, there are different formulas to calculate the mass of certain ions. For b-ions, the mass of a b_i-ion can be calculated with $b_i = 1 + \sum_{1 \leq j \leq i} m(a_j)$. Similarly, the mass of the y-ion of P with i amino acids, denoted by y_i, can be computed with $y_i = 19 + \sum_{n-i+1 \leq j \leq n} m(a_j)$ [26].

A *match* between a spectrum S_q and a peptide P occurs if there is a certain number of m/z values that experimental and theoretical spectra have approximately in common (the difference between values is less than the error tolerance).

Given by a spectrum S_q, the mass of a peptide $m(P)$, set of ion types $\triangle$ and a peptide database, the *peptide identification* problem is to find the peptide P in the peptide database that has the maximal match to S_q.

1.1.3 *De novo* Sequencing and Database Searching

There are two major approaches to determine peptide sequence based on a MS/MS mass spectrum. The approach without the help of a protein database is called the *de novo* method. A *de novo* sequencing algorithm takes an MS/MS spectrum as input and it then outputs peptide sequences whose fragment ions can interpret the spectrum best. A number of algorithms and software packages have been developed using this approach. An algorithm that enumerates all possible peptide sequences and compares each of them with the spectrum was proposed by Sakurai [34]. This method is computationally expensive and only feasible on very short peptides and simple spectra. Prefix pruning was introduced to improve this algorithm [17]. Prefix pruning has the drawback that it only works well on the spectrum whose prefix can be represented clearly. Another *de novo* algorithm constructs a spectrum graph based on a MS/MS spectrum [3] and then finds the longest path on this graph. Many software packages, such as Lutefisk [38], use this model and this model has since been widely investigated. A different model [26] taking the intensity of peaks into account was reported to achieve better accuracy. Though the *de novo* method is important in the study of new proteins, it has its own limitations. It requires higher quality spectrometry data and tends to fail on spectra when some peaks are missing.

Another approach is to determine the peptide sequence with the assistance of a protein database [13]. In this approach, we are usually given a protein database that hopefully contains all the target proteins in the experimental protein mixture. Therefore, if the target protein is not included in the database the identification will fail using this method. The first computational task is to simulate the MS/MS experimental procedure on the protein database. Every protein sequence in the database is cut into short peptides *in silico* to simulate the digestion process. As a result, a peptide database is generated to help the interpretation of MS/MS spectra. The next step is to find out peptides that are likely to generate the corresponding experimental MS/MS spectrum. In order to compare the experimental spectrum with the peptide sequences, hypothetical spectra are generated based on the peptide sequences. These hypothetical spectra consist of the theoretical m/z values of the fragment ions of the peptides. The comparison between a hypothetical spectrum and an experimental spectrum is done by matching the peaks of the experimental spectrum with the m/z values in the hypothetical spectrum. Different software packages and algorithms may consider different types of fragment ions [13]. Some algorithms may even generate different intensities for each peak [13]. In most packages, after the matched peaks are found, a *scoring function* is used to compute a score, which represents how well the experimental spectrum and the hypothetical spectrum matches. Any score above a parameterized confidence threshold is regarded as a so-called *hit*.

A good scoring function is an important factor for peptide identification. There are various factors that may be included in a scoring function, such as the ion types, the intensities of peaks and the shared peak counts. Further computation of the final score may be performed based on this 'raw score' to give a ranked list of the hits. Various methods are used in calculating this final score. Sequest [13] uses cross correlation to do further similarity comparisons to generate the final output. Other software packages like X!Tandem [10] and Mascot [29] use a probability based scoring scheme. They employ a similar idea from the well-known BLAST [27] database sequence searching tool by assuming the distribution of lower scores is an exponential distribution. After taking the logarithm of this distribution, a linear curve fitting can be performed to find the expected number of random matches of the best score. This number is called the E-value which represents the likelihood that the best match occurred by random chance. X!Tandem calculates this E-value empirically while Mascot tries to get the same kind of number using theoretical calculations.

The database searching approach is particularly good at identifying proteins that have already been found and documented. Therefore, it plays an important role in the early detection of genomic-related diseases, and the discovery of associated biomarkers. However, it also has its own disadvantages. Database searching approaches cannot be applied to sequence new proteins (those not in the database). The computational complexity of the database approach largely depends on the scale of the database. The rapid expansion of protein databases and the improvements of the throughput

of modern mass spectrometers demand efficient database searching algorithms.

1.2 Motivation and Objectives

1.2.1 Motivation

Protein databases have been growing rapidly, and due to the recent advances in technology of mass spectrometry, there has been a dramatic increase in the number of mass spectra being generated over the past few years. However, it is difficult for database searching algorithms to catch up with this data explosion. Therefore, it becomes more and more important to improve the efficiency of database searching algorithms. Furthermore, mass spectrometer manufacturers expect to implement real-time peptide identification and MS/MS data acquisition [40]. This real-time control of MS/MS data acquisition will greatly help to increase the number of peptides identified for low-abundance proteins in protein mixtures.

Low-abundance proteins are often identified based on only a few peptides. Therefore, the final score of these identifications is always very low. Moreover, some of the low-concentration proteins fail to be detected because they are masked by high-concentration proteins. In a typical MS/MS protein experimental analysis, the peptide mixtures are so complex that the mass spectrometer is not able to fragment all peptides eluting from the chromatographic column. Due to this complexity, the MS data acquisition algorithms generally select the most abundant precursor ions for further fragmentation to generate MS/MS spectra. In this way, the process of selecting ions for an MS/MS experiment is always biased towards abundant peptides. Indeed, most of the fragmented peptides belong to proteins of high abundance in the sample while only few fragmented peptides are from proteins of low abundance. As a result, the abundant proteins usually end up with far more peptides identified than it is necessary for confident protein identification while the low abundant proteins hardly get identified with confidence. Thus an efficient algorithm that can make a more equal portion of the fragmentations among all analyzed proteins would improve the identification of low-abundance proteins.

In recent years, an approach to address this problem has used a 'dynamic exclusion list' [16]. Once a peptide ion is fragmented, it is placed in an exclusion list. The peptide ions in the exclusion list will not be selected for fragmentation for a certain period. In addition, the exclusion list is dynamically changed according to the time span that ions stay on the list. In this way, the data acquisition time is controlled, consequently increasing the number of peptides identified in an experiment. However, this approach just places peptides that got fragmented in the exclusion list without considering the identification results. If we can include the identification results into the 'dynamic exclusion list' approach, a better data acquisition control can be achieved. That is, we can focus more on identifying low abundance peptides. But this cannot be achieved unless identification

is done in real-time. This can be thought of as an "intermingling" of the computational prediction within the Tandem MS experimental process.

With the original workflow for protein identification, MS/MS spectra are collected first and are analyzed only after a MS/MS experiment is done. There is no feedback from the identification result to the peptide selection algorithm to help the control of data acquisition. If sufficient peptides of an abundant protein could be identified fast enough, the feedback can effectively inform the selection algorithm to put the remaining peptides of this identified protein on an exclusion list. In this way, the selection algorithm could ignore those abundant proteins that have already been identified and give the low-abundant proteins more opportunity to get selected.

As mass spectrometers are high throughput instruments, the feedback of peptide identification results should not slow down the performance of the instruments and a 'real-time' feedback will be crucial for this application. Real-Time means that the time spent on peptide identification is short and strictly limited. For typical ion trap mass spectrometers such as LCQ DECA XP, it takes around 1 second to produce a mass spectrum [23]. This indicates that the peptide identification should be done in less than 1 second. Thereby, a very efficient identification algorithm is required to provide real-time feedback. To our knowledge, however, no software tools have shown the capability of being fast enough for real-time peptide identification. Most existing software packages are developed for off-line identification and for better accuracy. For these software packages, accuracy is much more important than efficiency. In our application however, efficiency is of greater relative importance.

1.2.2 Objectives

The implementation of such real-time processes is not trivial. But due to the rapid development of computer hardware and parallel computing architectures, it is possible to develop more efficient algorithms for peptide identification. The requirement of this real-time algorithm is not about achieving very high sensitivity and specificity, but rather to achieve a good sensitivity and a good specificity in a short time. In doing this, it might be acceptable to sacrifice some amount of accuracy for speed. In fact, a sensitive and specific identification result can be obtained with a post-processing, using a slower but more accurate program such as Sequest, Mascot or X!Tandem.

RT-PSM, a real-time program for peptide-spectrum matching with statistical significance, has been developed and introduced in [40] to achieve these requirements. It describes a design for the real-time peptide identification in peptide identification using a database searching approach with a final score to represent the statistical significance of the search result. Our work in this research is based on the RT-PSM design.

Our objectives for this thesis are to analyze the performance of RT-PSM and to develop methods to improve the speed of RT-PSM for fulfilling the real-time control of data acquisition. We have done

performance evaluation and profiling on the different modules of RT-PSM. Then according to the profiling result, we focus on the most time consuming aspects and try to address the problem using various methods. First, we investigate two common parallel computing methods to optimize speed. One is to apply a data level parallel algorithm in calculating the shared peak counts using Single-Instruction Multiple-Data (SIMD) [28] instructions and thus improving the speed of the scoring function. The other is to utilize NVIDIA CUDA technology [6] to perform parallel computing at the Single-Instruction Multiple-Thread (SIMT) level. We show that both parallel computing methods can improve the speed significantly. Then, we introduce the method of making an index based on metric spaces and provide test results and an explanation as to why the metric space indexing structures do not fit in RT-PSM.

Though we developed and tested our methods on RT-PSM, these works can be applied to most other database searching software directly. Most database searching software packages need to calculate the shared peaks counts as it serves as a basis for scoring functions. On the other hand, the parallel computing model for RT-PSM running on the NVIDIA CUDA architecture can also easily be reused. Hence, our work does not only help in real-time peptide identification but can contribute to a general peptide identification algorithm as well, although improving the speed is more important for real-time identification.

1.3 Thesis Overview

In this thesis, we describe the methods to address the problem of real-time peptide identification. As our implementation is based on RT-PSM, we give an introduction of the workflow and the design of RT-PSM in Chapter 2. Moreover, Chapter 2 gives important performance profiling results of every functional module in RT-PSM. Under the guidance of the profiling results, there are two major approaches to optimize RT-PSM. One approach is parallel computing. Chapter 3 provides a method to parallelize the critical operation of calculating the shared peak counts using SIMD instructions, present on common processors. Chapter 4 then describes the method of parallel computing using NVIDIA CUDA technology, used in recent NVIDIA graphics processors. The other approach is discussed in Chapter 5 where we describe how we evaluate the feasibility of making a metric index on peptide databases, in order to facilitate the selection of candidate peptides and give evaluation results. In the last chapter, we provide conclusions of our work and discuss possible future directions.

Chapter 2

RT-PSM: A Real-Time Program For Peptide-Spectrum Matching

2.1 Introduction

RT-PSM is a program that performs peptide-spectrum matching with statistical significance for achieving real-time data acquisition [40]. We will describe their work in detail in this chapter, as it will form the basis for most of the new work in this thesis.

RT-PSM employs a database searching algorithm, which aims to achieve real-time peptide identification and subsequently give feedback to the peptide selection algorithm to perform better data acquisition. Moreover, RT-PSM estimates the statistical accuracy of peptide identification in an efficient manner. In calculating this statistical significance score, RT-PSM first calculates *Peptide-Spectrum Matching* scores (PSM scores) between an experimental MS/MS spectrum and all the theoretical spectra of candidate peptides selected from a peptide sequence database. Then, RT-PSM calculates the empirical expectation value distribution of random PSM based on all the PSM scores. The expectation values are fit onto a second-degree polynomial function in the PSM score by regression. In the last step, RT-PSM extrapolates the expectation value for the maximum PSM score.

RT-PSM also employs various methods to achieve high performance of peptide identification. Unlike many other database searching software packages such as Mascot, RT-PSM preprocesses a protein database to generate a sorted tryptic peptide database. All the database searching operations are then done by searching against this sorted peptide database directly. In this way, the run time required for experimental spectrum identification can be reduced. Furthermore, in contrast with popular software packages such as PeptideProphet [21], multiple function modules including spectra filtering, database searching and PSM calculation and statistical significance estimation are all integrated in RT-PSM as a single piece of software. Thus, the cost of communication between different function modules is reduced significantly and hence the performance is improved. Most other peptide identification algorithms such as the X!Tandem are designed and developed for batch processing of MS/MS spectra. They are fast in terms of the average run time but slow for the

identification of individual spectra [9]. Moreover, few existing peptide identification software are open source. The source code of most software packages is proprietary and not available. Therefore, they cannot be modified to suit our purpose of achieving a real-time control process. In order to maximize the performance, RT-PSM is implemented as a random-access memory (RAM)-resident MS-Windows service. The executable file of PSM assignment and statistical assessment program and the peptide sequence database are loaded only once at the first launch of the service and remains in RAM afterwards for on-line spectrum identification. This design greatly helps to achieve real-time peptide identification and mass spectrometer acquisition control. Other common MS/MS spectrum identification programs like Sequest, Mascot and the X!Tandem are not implemented in this fashion.

2.2 Function Modules of RT-PSM

2.2.1 Experimental Spectrum Preprocessing

The quality of the experimental spectrum is a very important factor in determining how well it can be interpreted by an algorithm. Preprocessing of the raw experimental spectrum could improve identification. In general, there are two methods to deal with a raw experimental spectrum. One is to apply a classification algorithm on the criterion of quality of spectrum and then drop those spectra that are classified as poor quality. The other method is to preprocess the experimental spectra to improve their quality for further analysis. RT-PSM chooses the latter approach as it is usually less computationally expensive.

The experimental MS/MS spectrum of a precursor ion with mass $m(S_q)$ consists of a list of peaks, i.e., $S_q = \{(q_i, h_i) \ : \ 1 \leq i \leq m\}$, where (q_i, h_i) denotes the fragment ion i with m/z value q_i and intensity h_i. Peak intensities vary on the m/z values of peaks. There are several complex factors that can affect the intensities, such as composition-dependent fragmentation kinetics, precursor ion activation parameters, mass analyzer artifacts and collision energy [25]. But, considering all these factors will increase the computational complexity. Instead, RT-PSM uses a simple model to choose the N most intense peaks and remove the other peaks. Though this filtering algorithm is based on the assumption that the more intense peaks are more informative, it is true that peaks of relatively low intensity tend to be random noise [40]. The threshold method can help to select the most informative peaks for peptide identification. This variable N serves as an argument of the whole identification algorithm. If N is set to be a very small value, some informative peaks can be lost. Correspondingly, a large N can cause many noisy peaks to be included and in turn affect the matching score and computation time.

After filtering out those peaks of low intensities, RT-PSM ignores the intensity values of the selected ions. Only the m/z values of peaks are used to calculate PSM scores as ion intensities

are affected by many factors and hence very difficult to utilize. Furthermore, once intensities are involved, the computational complexity of the PSM score will grow significantly.

2.2.2 Candidate Peptides Selection

In general, an experimental MS/MS spectrum is used to search against the whole peptide sequence database sequentially to find the peptide sequence that can best explain the experimental spectrum. Modern mass spectrometers can provide the mass of the precursor ion together with the spectrum. With the mass of the precursor ion, we can refine the candidate peptides to a much smaller subset. RT-PSM computes the absolute difference in precursor mass within a tolerance window and uses this window as a filtering window to select candidate peptide sequences. RT-PSM chooses the set of candidate peptides P such that their masses $m(P)$ satisfy

$$|m(S_q) - m(P)| \leq \mu, \tag{2.1}$$

where $m(S_q)$ is the precursor mass of the experimental spectrum S_q, and μ is the tolerance value which serves as a threshold in determining candidate peptides. Comparing with the whole database, the number of candidate peptides is reduced remarkably by this method and thereby the run time of peptide identification is reduced.

2.2.3 Peptide-Spectrum Match Score

In order to calculate the PSM score, a theoretical spectrum should be calculated. A theoretical spectrum (denoted by S_p) of a peptide sequence P also consists of the m/z values of the ions being considered. Moreover these theoretical m/z values are calculated based on the peptide sequence, fragment patterns and amino acid residue masses. Although different software packages may consider different ions, RT-PSM considers the most common ions as shown in Table 2.1.

Ion type	m/z	Score weight
y^+	y	1
$y^+ - H_2O$	$y - 18$	0.4
$y^+ - NH_3$	$y - 17$	0.4
$y^+ - NH$	$y - 15$	0.4
b^+	b	1
$b^+ - H_2O$	$b - 18$	0.4
$b^+ - NH_3$	$b - 17$	0.4
$b^+ - CO$	$b - 28$	0.4

Table 2.1: These ions are involved in the similarity score in RT-PSM.

In Table 2.1, besides the ion types and the calculation of their corresponding m/z values, a score weight is assigned to each ion type. The score weight represents the significance of the

different ion types, as some fragment types are more likely than others in a MS/MS experiment. Therefore, some fragment ions are considered to provide stronger evidence in peptide-spectrum matching than other ions. As shown in this table, b-ions and y-ions are given the highest weight of 1 because they are most favored in the process of peptide fragmentation. The other ions are regarded as "supporting ions" and are given a score weight of 0.4. These score weights are later used to calculate peptide-spectrum matching scores.

Different mass analyzers have their own specification of the precision and accuracy on m/z value measurements. For example, the m/z value accuracy of ion trap mass spectrometers is around $0.005 - 0.05\ Thomson$ (a Thomson is a unit of mass-to-charge ratio). Therefore, an error tolerance is employed in determining if two m/z values are similar. We denote the error tolerance of the MS/MS instrument in use by δ. A peak q of an experimental spectrum S_q is said to be *matched* with a theoretical spectrum S_p when a predicted peak p exists such that $|q - p| \leq \delta$. The peaks in S_q which can be matched with peaks in S_p are denoted as a set S_{qp}. That is,

$$S_{qp} = \{q_i \in S_q : \text{there exists } p \in S_p \text{ such that } |q_i - p| \leq \delta\}. \tag{2.2}$$

Alternatively, for a peak $q \in S_q$, then $q \notin S_{qp}$, if there is no $p \in S_p$ that satisfies $|q - p| \leq \delta$. In general, the more elements contained in the set S_{qp}, the higher the probability that the experimental spectrum S_q was generated from the peptide P whose theoretical spectrum is S_p. Moreover, as different types of fragment ions provide different levels of evidence, the score weights of these ions are used in calculating the peptide-spectrum matching score. The setting of score weights in Table 2.1 is empirical. These score weights also serve as the arguments of the RT-PSM algorithm and can be fine tuned to different MS/MS instruments. The score weights in Table 2.1 were found to be most suitable for the LCQ ion trap spectra and actually used in the implementation of RT-PSM [40].

The peptide-spectrum matching score between an experimental MS/MS spectrum S_q and a theoretical spectrum S_p of a peptide sequence P from a peptide database is then defined as

$$Score'(q, p) = \sum_{q \in S_{qp}} W(q), \tag{2.3}$$

where we denote the score weight of the ion with m/z value q by $W(q)$. This score indicates the similarity between an experimental MS/MS spectrum and a theoretical spectrum of a peptide sequence. However, according to Equation 2.2, given an experimental MS/MS spectrum S_q, long peptide sequences are more likely to get a higher peptide-spectrum matching score as they consist of more theoretical ions. Because of this, the score calculated through Equation 2.1 is then normalized

by the length of peptide P as follows,

$$Score(q,p) = Score'(q,p)/length(P), \tag{2.4}$$

where $length(P)$ is the number of amino acid residues in peptide P.

Given a query experimental MS/MS spectrum and a set of candidate peptide sequences, the peptide-spectrum score between the experimental MS/MS spectrum and each peptide sequence are calculated and the results are sorted. The peptide corresponding to the highest score is considered to be the correct peptide for the experimental spectrum.

The algorithm of peptide-spectrum scoring function used by RT-PSM is shown in Algorithm 1.

2.2.4 Statistical Significance Score

As the total number of peptide sequences in a peptide dataset is bounded, given by an experimental MS/MS spectrum S_q, there always exists a highest peptide-spectrum matching score and a corresponding peptide. However, the peptide with the highest score may not be the right peptide that produced the experimental spectrum. Thus, other than the highest peptide-spectrum matching score, we need to know how confident we are about the match between the experimental MS/MS spectrum and the peptide with the highest score being the correct identification. This confidence level can be expressed by the expectation value (E-value). The E-value for a peptide-spectrum matching score h estimates the number of expected peptides with score greater than h. As the number of random peptide-spectrum matches in a database search has been proven to follow a Poisson distribution [32], the probability of finding exactly t peptides with score greater than or equal to h is given by

$$e^{-E(h)}\frac{(E(h))^{-t}}{t!}, \tag{2.5}$$

where we denote the expectation value of score h by $E(h)$. Particularly, the probability of having no peptides with peptide-spectrum matching score greater than or equal to h is $e^{-E(h)}$, so the probability of finding at least one such peptide is

$$p = 1 - e^{-E(h)}. \tag{2.6}$$

This is called the p-value for the peptide-spectrum matching score h. The p-value reflects the probability that the match occurs merely by chance. For example, a p-value of 0.05 implies that there is 5% probability that the match is random and thus probably not a true positive. Obviously, the smaller the p-value, the more confident we are that the match is correct.

Algorithm 1 [40]: RT-PSM uses the following peptide-spectrum scoring algorithm

Input: $spq[np]$: experimental spectrum which consists of np peaks
$pep[na]$: peptide sequence which consists of na amino acid residues
Err: error tolerance and total mass of the peptide $pepmass$
Output: $score$: peptide-spectrum matching

$b_{ion}[1] \leftarrow mass\ of\ hydrogen + mass\ of\ pep[1]$; // Calculate the mass of b_1 ion
for $i = 2$ to na **do**
  $b_{ion}[i] \leftarrow b_{ion}[i-1] + mass\ of\ pep[i]$; // Calculate the mass of all b_i ions
end for
$score \leftarrow 0$;
for $j = 1$ to na **do**
  if BinarySearch$(b_{ion}[i], spq, pepmass, Err)$ returns true **then**
    $score \leftarrow score + score\ weight\ of\ b\ ion$;
  end if
  if BinarySearch$(b_{ion}[i] - mass\ of\ H_2O, spq, pepmass, Err)$ returns true **then**
    $score \leftarrow score + score\ weight\ of\ b - H_2O\ ion$;
  end if
  if BinarySearch$(b_{ion}[i] - mass\ of\ NH_3, spq, pepmass, Err)$ returns true **then**
    $score \leftarrow score + score\ weight\ of\ b - NH_3\ ion$;
  end if
  if BinarySearch$(b_{ion}[i] - mass\ of\ CO, spq, pepmass, Err)$ returns true **then**
    $score \leftarrow score + score\ weight\ of\ a\ ion$;
  end if
  // Start to process y ion group
  $y_{ion} = pepmass - 2 * mass\ of\ Hydrogen - b_{ion}[i]$;
  if BinarySearch$(y_{ion}, spq, pepmass, Err)$ returns true **then**
    $score \leftarrow score + score\ weight\ of\ y\ ion$;
  end if
  if BinarySearch$(y_{ion} - mass\ of\ H_2O, spq, pepmass, Err)$ returns true **then**
    $score \leftarrow score + score\ weight\ of\ y - H_2O\ ion$;
  end if
  if BinarySearch$(y_{ion} - mass\ of\ NH_3, spq, pepmass, Err)$ returns true **then**
    $score \leftarrow score + score\ weight\ of\ y - NH_3\ ion$;
  end if
  if BinarySearch$(y_{ion} - mass\ of\ NH, spq, pepmass, Err)$ returns true **then**
    $score \leftarrow score + score\ weight\ of\ c\ ion$;
  end if
end for
Normalize $score$
return $score$

In order to calculate the p-value or E-value, we need to know the probability distribution of the scores for random peptide-spectrum matches. In general however, the distribution of random peptide-spectrum matching scores is not available in a parameterized form for experimental MS/MS datasets. Most common methods to calculate the distribution of peptide-spectrum matching scores for an experimental spectrum and peptide database are time-consuming. RT-PSM employs an approach that constructs a histogram of peptide-spectrum matching scores. During a search against a set of candidate peptide sequences, most scores will be results of random matches. The highest peptide-spectrum match score would probably be a correct match, if it is in the database. Then, the probability distribution of random scores can be estimated based on the histogram and so can the E-value function. This method has been used by Beavis and co-workers for Sonar [15] and GPM [14] and by Havilio [18] in their study.

Using the peptide-spectrum match score in Equation 2.4, it is observed experimentally that the relationship between $log(E(x))$ and score x fits a second-degree polynomial function. Therefore, RT-PSM fits the logarithm of the E-value distribution to a second-degree polynomial distribution model in scores. This model is estimated without the lowest and highest 10% of scores, which are regarded as noise. Then RT-PSM estimates the E-value of the highest peptide-spectrum matching score using the distribution model and calculates a p-value using Equation 2.6.

Combining all these function modules, the flow chart of processing an experimental MS/MS spectrum is shown in Figure 2.1.

2.3 Performance analysis of RT-PSM

In order to improve the speed of RT-PSM, profiling tests are required to find out the most time-consuming portion. As the goal of RT-PSM is to achieve real-time response, we are more interested in single spectrum identification than the batch processing of spectra. Thus, we must measure the run time per spectrum identification and not just the throughput of the whole system.

We used the dataset provided with the RT-PSM source package. The MS/MS spectrum dataset consists of 22577 peptide collision-induced dissociation spectra acquired using an LCQ DECA XP ion trap (ThermoElectron Corp.). The test protein dataset is a subset of the UniRef100 database (release 1.2) containing 44278 human protein sequences [1].

The run time of a single spectrum identification is rather short. So we need a method to measure time with a high degree of precision. The Windows operating systems offer such a high resolution timer. It is accessed through a Windows API, *QueryPerformanceCounter*. Because the counter frequency is generally the same as the CPU clock frequency, this API provides a very high resolution counter. Though some other Windows APIs can help to convert this counter value into the time unit of milliseconds, in our study we need only the proportion of run time of all these function

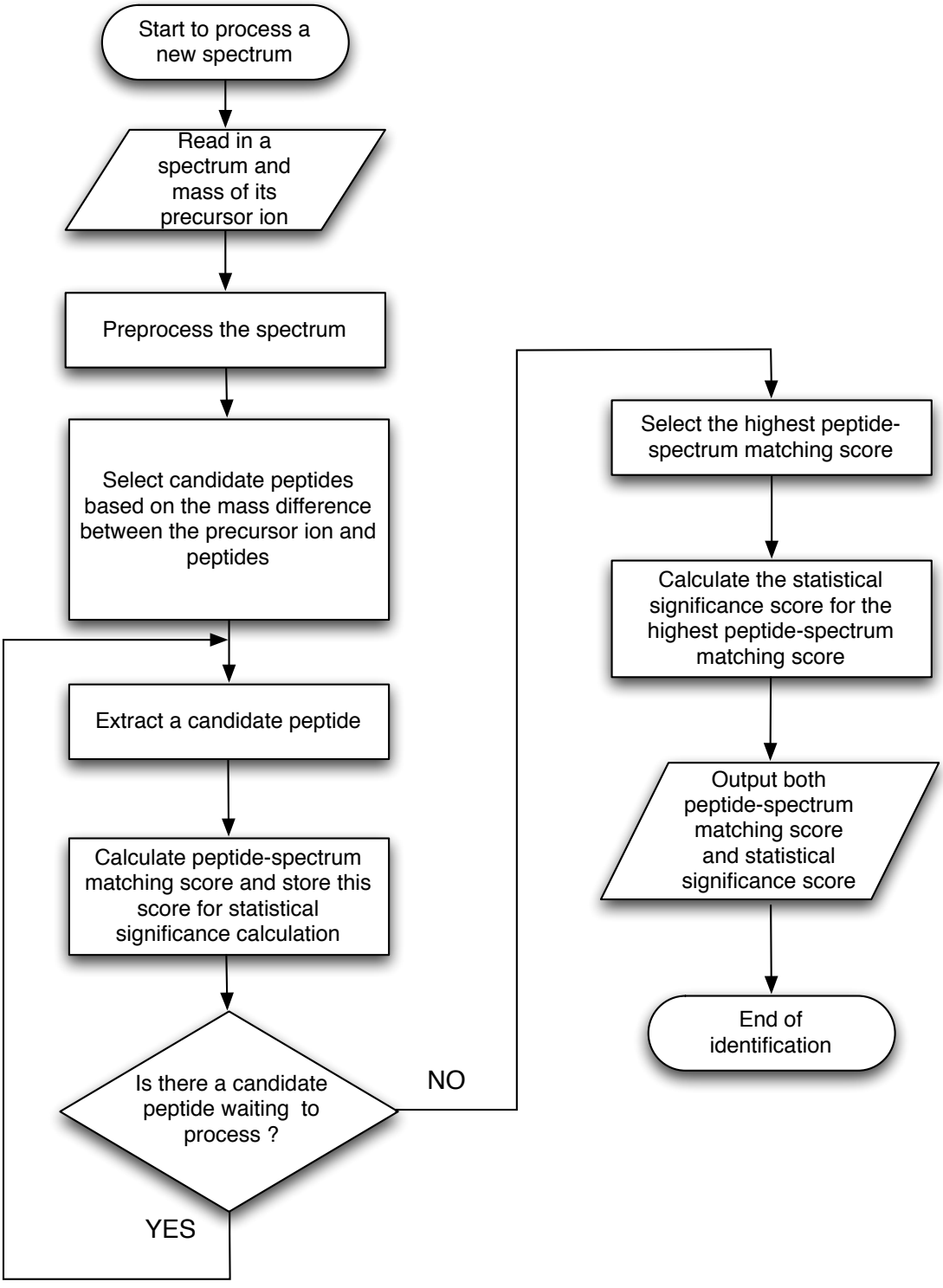

Figure 2.1: The flowchart illustrates the process of a MS/MS spectrum identification in RT-PSM.

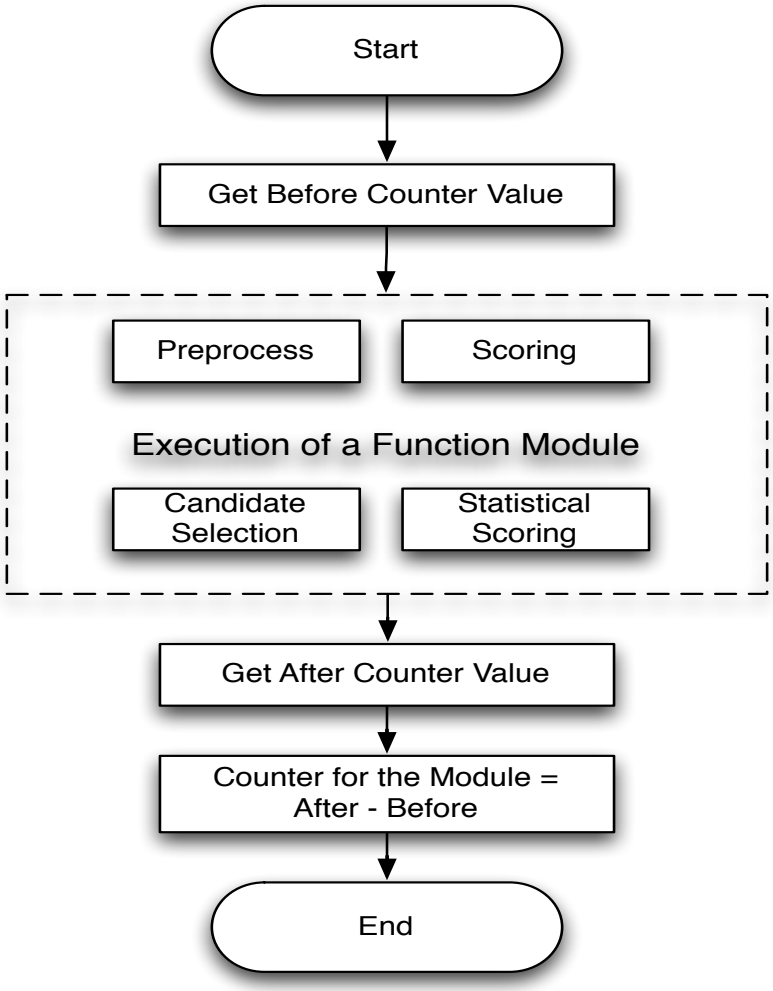

Figure 2.2: Function modules of RT-PSM are profiled using this procedure.

modules. Therefore, in our profiling test, we used *QueryPerformanceCounter* only. The software methodology for timing the function modules of RT-PSM is summarized in the flowchart of Figure 2.2. This is the profiling method we use with the Windows OS. In our work, we do profiling on Mac OS X as well. Then we use a different system API to do the time measurement following a similar workflow.

The exact counter value of the same function module varies on every iteration due to the difference of every spectrum, the different set of candidate peptides, operating system task switching and the hardware status of each iteration. We took the average run time of 1000 spectra searches against the same peptide database. Therefore, this profiling results should reflect the general use of a spectrum identification. The profiling results are shown in Figure 2.3. In this figure, the computation weight for each module has been converted to relative weights to facilitate the comparison of profiling results from different system and hardware.

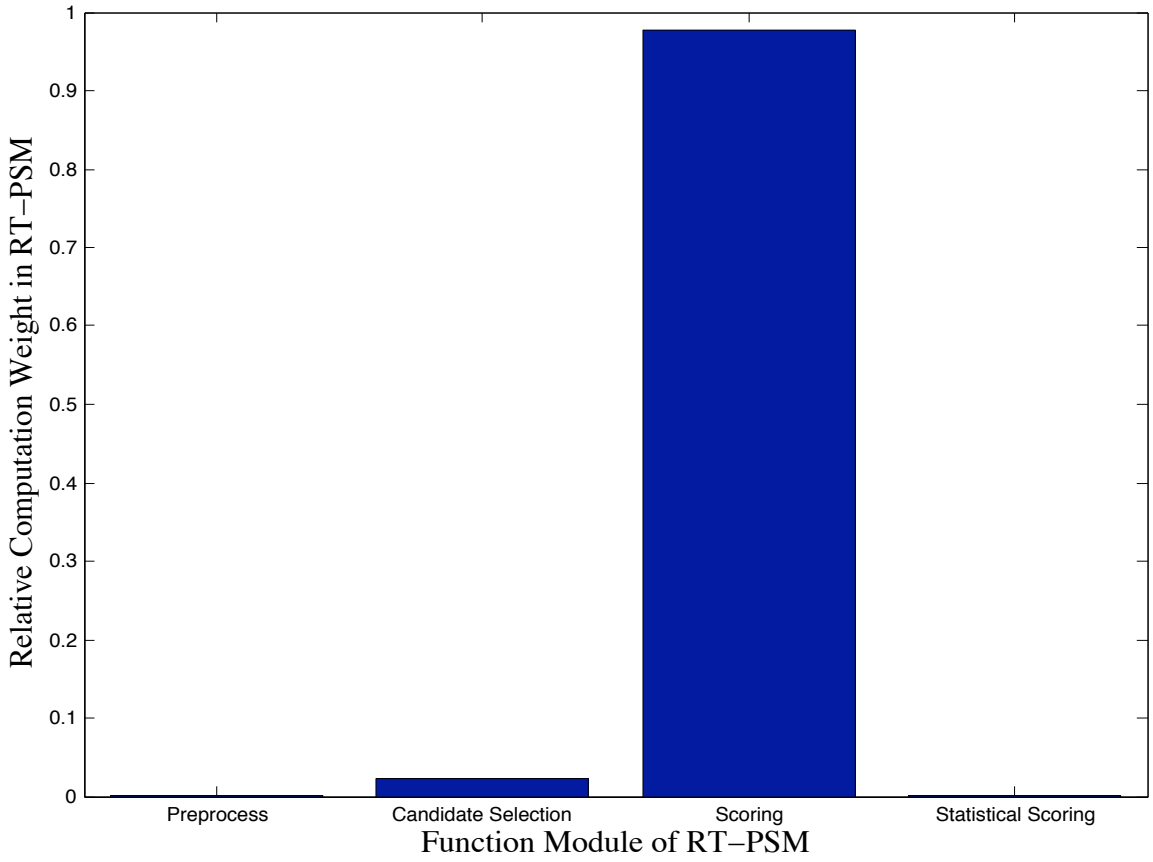

Figure 2.3: Profiling results of peptide identification indicates that the computation of the similarity scores consumes the most CPU time.

From this figure, we can clearly see that the scoring process is the most time consuming portion in the process of a spectrum identification. This observation is reasonable as a single experimental

spectrum has to be compared with thousands of peptide sequences from a database and the scoring function is invoked for each peptide-spectrum pair.

There are two directions to optimize the peptide-spectrum scoring process. One way is to reduce the number of the candidate peptides and hence improve the efficiency of a single spectrum identification. However, we cannot reduce the number of candidate peptides freely because we need to make sure that the peptide with the highest score will not be missed in the selection of candidate peptides. A sophisticated indexing system on the peptide database follows this direction. The other direction is to reduce the run time of calculating the score of each peptide-spectrum pair. This leads to efficient scoring algorithms and parallel computing methods.

Chapter 3

Parallelizing Peptide-Spectrum Matching Using SIMD Instructions

3.1 Parallel Computing in MS/MS Spectrum Identification

The focus of this chapter [1] is to try to optimize the computation of the identification algorithm for a spectrum. Parallel computing can be used to improve the speed of MS/MS spectrum identification using the database search approach. Several methods to parallelize the existing common software packages exist. One method called *Parallel Tandem* [11] was developed using the Parallel Virtual Machine (PVM) and Message Passing Interface (MPI). It combines a database search algorithm, X!Tandem, and a Linux cluster parallel computing environment with PVM and MPI. A parallel version of Sequest [33] was also introduced. Similar to Parallel Tandem, it was implemented on computer clusters using PVM as the inter-processor communication method as well. All these parallel algorithms are based on a computer cluster environment. The parallel computation is done by dividing the set of MS/MS spectra into subsets and then processing multiple MS/MS spectra in parallel with a peptide database. The parallelism is achieved at the level of spectra subsets and the speed-up can be obtained in terms of the identifications of a batch of spectra. RT-PSM, however, requires speed-up for the identification of a single spectrum since it is designed to provide the identification result of every spectrum to the dynamic exclusion list algorithm in real-time.

Unlike these parallel algorithms, our algorithm parallelizes the computation of matches between a single spectrum and a given peptide sequence from the database and hence improves the speed of individual spectrum identification.

[1] A shortened version of this chapter has been published in [41].

Manufacturer	Microprocessor	Name of the technology
Intel	Pentium MMX/II	MMX (MultiMedia eXtensions)
	Pentium III/Xeon	MMX,SSE (Streaming SIMD Extensions)
	Pentium IV	MMX,SSE, SSE2 (Streaming SIMD Extensions 2)
AMD	K6/K6-2/K6-III	MMX/3DNow!
	Athlon	Extended MMX/3DNow!
HP	PA-RISC	MAX-2 (MultiMedia Acceleration eXtensions)
SGI	MIPS	MDMX (MIPS Digital Media eXtensions)
Freescale	PowerPC G4/G5	Velocity Engine (AltiVec)
Sun	SPARC	VIS (Visual Instruction Set)
ARM	ARMv6	NEON
IBM	P6	VMX (with some extensions)
Sony/Toshiba	Cell (PPE)	AltiVec

Table 3.1: Microprocessors with SIMD technology (modified from [31]).

3.2 Overview of SIMD technology

3.2.1 Introduction

A more readily available form of parallel processing capability is available in *Single-Instruction, Multiple-Data* (SIMD) computers [36]. SIMD instructions were first designed to accelerate the performance of applications such as motion video, real-time physics and graphics. These applications always require repetitive operations on large arrays of numeric values. SIMD technology allows one instruction to operate on multiple data units at the same time, which allows to achieve higher performance by processing more data with fewer instructions. By providing this support, SIMD technology can capture some of the potential speed-up due to the parallel nature in these applications.

SIMD technology is not only available for the Intel architecture as "MMX", "SSE" or "SSE2" but also in a number of other architectures shown in Table 3.1. These SIMD technologies do not have the same capability and some of them are limited. For instance, Sun's VIS does not support floating point values in SIMD registers and provides 64-bit rather than 128-bit registers while Intel's SSE can support both floating point values and 128-bit registers [42]. However, all these SIMD technologies share most common ideas and general operations. Because of this, though an SIMD implementation of an algorithm cannot be used directly on different architectures, it is generally not difficult to port an algorithm to another SIMD architecture.

Figure 3.1 illustrates the use of SIMD instructions using the packed single-precision floating-point instructions available on the Intel SSE architecture. SIMD instructions on other architectures are similar. In the figure, both the source operand and destination operand are using 128-bit registers for this particular instruction. Every operand contains four 32-bit single-precision floating-

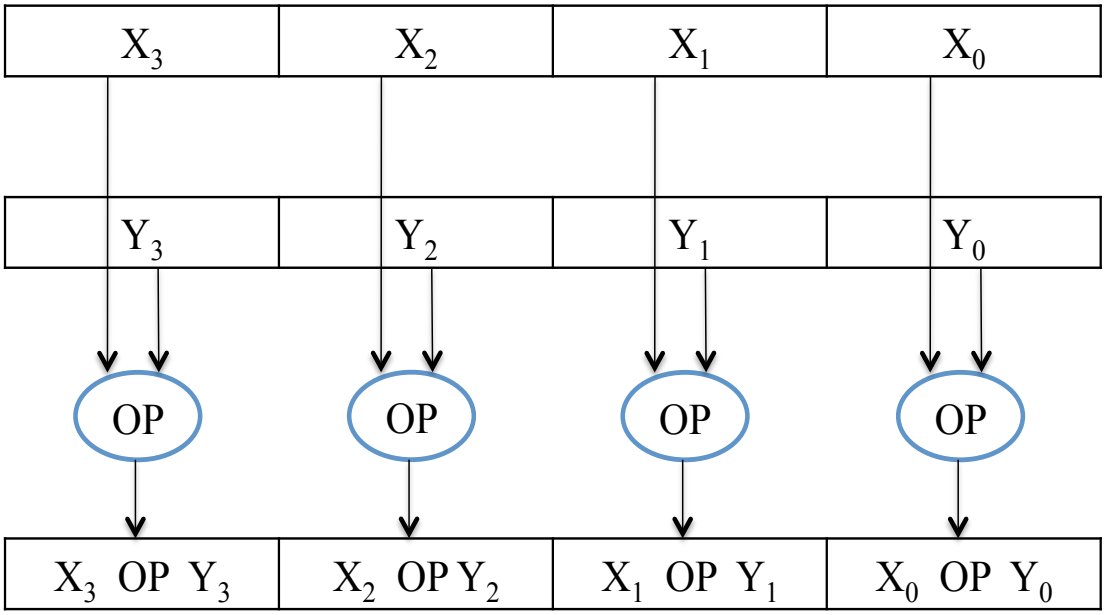

Figure 3.1: Packed single-precision floating point operation

point values. The destination operand holds the results of the operation (OP) performed in parallel on the corresponding values (X_0 and Y_0, X_1 and Y_1, X_2 and Y_2, and X_3 and Y_3) in each source operand. In this way, computation with SIMD enables processors supporting the SIMD technique to execute one instruction on multiple data units concurrently and thus increases the amount of data that can be processed in a given time interval. Many operations are supported by SIMD instructions including various comparisons, arithmetic, shuffle, conversion and logical operations.

3.2.2 Intel MMX and SSE instructions

Introduction

MultiMedia eXtension (MMX) is a SIMD instruction set designed by Intel, introduced in 1996 with their P5-based Pentium microprocessors [28]. It exploits the parallelism in various multimedia applications using SIMD techniques, yet maintains full compatibility with all previous existing Intel Architecture (IA) processors.

MMX technology defines a simple and flexible SIMD execution model to handle 64-bit packed integer data [28]. It defines eight 64-bit general-purpose registers that can hold MMX data. Although MMX registers appear as separate registers in the Intel architecture, these registers are physically shared with the floating point unit's (FPU) registers. Each of the eight MMX 64-bit registers is physically equivalent to the lower half 64-bits of each of the FPU's registers. Because

of this, a mode switch operation between MMX mode and FPU mode has to be performed before entering or exiting MMX mode. Though a single instruction is needed for this mode switch, the switch operation is very expensive. As there are many limitations of MMX, Intel later developed a much more powerful SIMD instruction set, Streaming SIMD Extension (SSE) [36].

The SSE extension was introduced as part of the Intel Pentium III processor family. These extensions are an update to MMX technology. The latest Intel processors supporting SSE also support the MMX instructions. The SSE extension adds eight new 128-bit registers known as XMM0 through XMM7 [24]. These registers can be accessed independently from FPU registers and other general-purpose registers. Each register can be divided into two 64-bit data units, four 32-bit data units, eight 16-bit data units or sixteen 8-bit data units. Furthermore, SSE supports integer, single-precision and double-precision floating point values. The SSE instruction set consists of 70 new instructions that can operate on the XMM registers, MMX registers and/or memory [24]. After SSE, SSE2 was introduced into the IA-32 architecture in the Pentium IV and Intel Xeon processors. SSE2 introduces SIMD computations on two double-precision floating point data elements and extends the instructions introduced in MMX and SSE. Intel keeps developing SSE technology and many new instructions are included in the latest version of SSE. But our SIMD implementation of peptide-spectrum matching uses instructions from SSE2 only.

Data Organization

When using SIMD instructions, the data alignment is another issue that must be considered. For SSE and SSE2 instructions that operate on 128-bit registers, data must be stored on 16-byte boundaries to take maximal advantage of SIMD instructions [24]. Though unaligned data is also allowed to be copied into and out of XMM registers, access to unaligned data with SSE instructions is much slower than aligned access. Furthermore, if the source data is contiguous, a whole vector can be loaded at once to achieve better memory access performance.

Comparison Result Format

Unlike ordinary comparison instructions, a SSE comparison operation results in an element mask corresponding to the length of the packed operands [42]. In an element mask, each packed data element contains either all 1's or all 0's. For instance, if the OP is a comparison operation in Figure 3.1, then the result is an element mask containing four sub-elements, with each sub-element consisting either of all 1's where the comparison condition is true, or all 0's where the comparison condition is false.

Branch Misprediction

Modern CPUs employ branch prediction in an attempt to predict the outcome of branches, and have special hardware for maintaining the branching history of many branch instructions. Conditional branch instructions are problematic in modern pipelined architectures because the CPUs do not know in advance which of the two possible outcomes of the comparison will happen. However, this causes a substantial delay due to mispredicted branches. A less obvious benefit of using the SSE instructions is that we can avoid conditional branch instructions by performing arithmetic on the results of the SIMD operations [42]. In this way, the delay caused by branch misprediction can be reduced.

3.3 Algorithm and Implementation

3.3.1 Comparison Operations in Peptide-Spectrum Matching

RT-PSM employs Algorithm 1 to calculate the matching score for each experimental spectrum and peptide sequence pair. This algorithm first calculates the theoretical m/z value of b-ions. The m/z values of other ions are calculated according to Table 2.1. Then each of these m/z values are used in searching against the peaks of the experimental spectrum to check if the value exists in these peaks. In order to obtain high efficiency, a binary search is performed in searching for a match. In spite of this, most of the time is spent on searching. More precisely, among all the primitive computation in this algorithm, comparison operations dominate the whole process.

The comparison operations on the pairs of predicted ions and the peaks of experiment spectra can be modelled with a peptide-spectrum matching matrix as Figure 3.2.

The row label indicates the various theoretical ions. Not all the ions considered in Table 2.1 are listed. The ions considered by RT-PSM on a given peptide sequence can be viewed as a vector of m/z values. Furthermore, the column label shows the peaks of an experimental spectrum and the peaks can be viewed as a vector of m/z values as well. Then each cell in this matrix represents a comparison between the m/z value of an experimental spectrum peak and a theoretical ion. The comparison is done with Equation 3.1,

$$|x - y| \leq \delta. \tag{3.1}$$

We denote the m/z value of a peak by x and the m/z value of a theoretical ion by y. Then, δ represents the mass tolerance of the instrument. Generally, two condition statements have to be used to compute this absolute difference in high level programming languages such as C/C++. Therefore, given a peptide sequence of N amino acids and an experimental spectrum consisting of

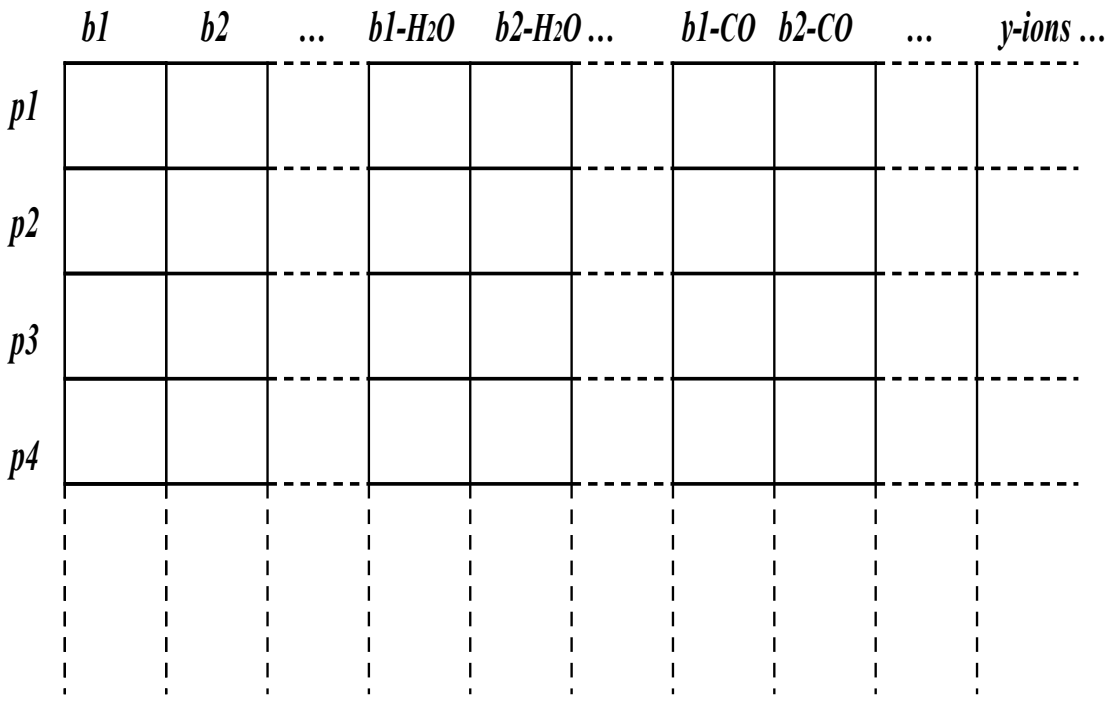

Figure 3.2: The comparison operations can be modelled with a Peptide-Spectrum matching matrix.

M peaks, RT-PSM will consider $N \times 8$ ions in total. As RT-PSM searches every ion among the peaks with a binary search algorithm, then the total number of comparison operations is $O(Nlog_2M)$. However, in practice, the number of peaks after preprocessing and the number of amino acids of the peptides are both small. Binary searches used on small-scale data tend to fail to bring significant improvements of speed because of the overhead of function calls, conditional branches and branch mispredictions.

3.3.2 Parallel Algorithm

Parallel Direction

The comparison operations can be parallelized with SIMD instructions so that multiple comparisons, that is multiple entries in the peptide-spectrum match matrix, can be performed simultaneously. RT-PSM has converted all the m/z values from floating point values to 32-bit signed integer values for computational efficiency. Therefore, we choose SSE registers (usually 128-bit wide),

which can be divided into four 32-bit signed integer units. With this division, many computations can be performed on the four data units independently with a single instruction. Though the SSE instruction set provides methods to load data from arbitrary locations in memory by executing data load and shuffle instructions, it is very inefficient to load data from non-contiguous addresses to a SSE register. Therefore, we choose to take data in contiguous addresses as vectors and perform load and set operations on those.

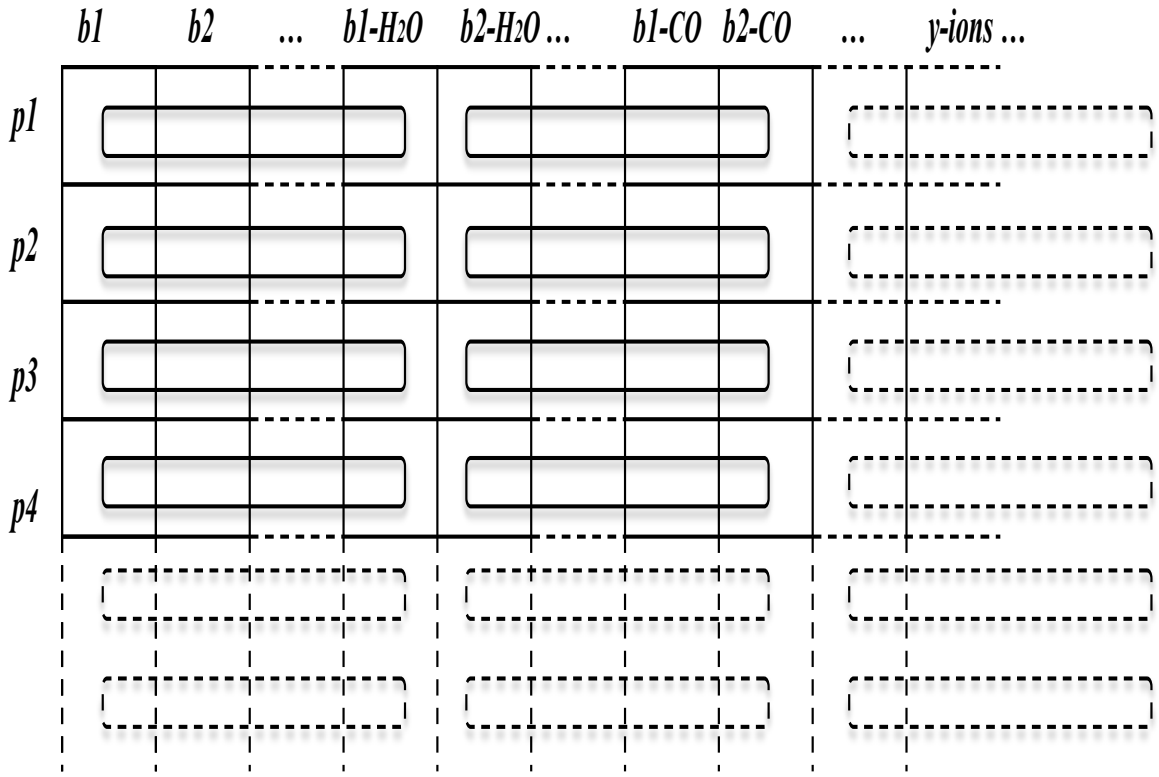

Figure 3.3: A possible vector arrangement uses vectors in parallel to theoretical ions.

As there are two natural directions for sequential search, the peptide-spectrum matching can be parallelized in two directions using SIMD instructions. One direction is to perform multiple comparison operations of vectors in parallel to the predicted ions (row vector) as shown in Figure 3.3. In this case, comparison operations are done between one SSE register containing four data units of the same m/z value of a peak and another SSE register holding four different m/z values of four different predicted ions.

The second way is to do multiple comparisons of vectors in columns. We group together multiple comparison operations in vectors in parallel to the peaks of the spectrum, as shown in Figure 3.4. In this case, each SIMD comparison is done between a register holding the same four m/z values

of a predicted ion and four different m/z values of four different peaks.

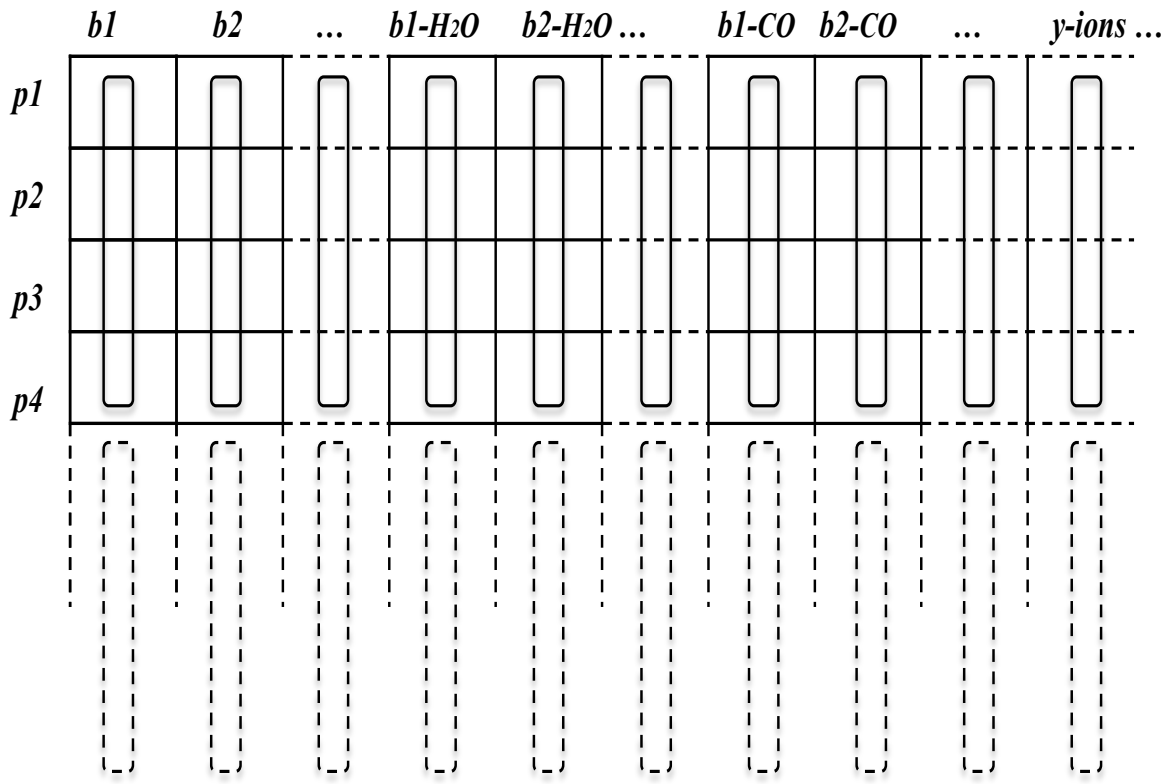

Figure 3.4: Another vector arrangements uses vectors in parallel to peaks of the experimental spectrum.

Comparison Operation Reduction

Both vector arrangements can lead to correct parallel algorithms. However, taking comparison vectors in parallel to the peaks can save many redundant comparison operations. Therefore, our algorithm uses this direction, searching vectors of the spectrum peaks in predicted ions. As the fragment ions can be divided into b-ion groups and y-ion groups, we discuss how the comparison operations can be saved with the example of the b-ion group only. But the same idea can be applied to the y-ion group with little modification.

Let b_i be the mass of a b-ion which consists of a total of i amino acid residues. As shown in Table 2.1, the mass of an a-ion with the same number of amino acids is $b_i - 28$. The b_i-ion may lose a water molecule or an ammonia to form the other two ions considered by RT-PSM and the calculations of these two ions are $b_i - 18$ and $b_i - 17$, respectively. We use notation $B(i)$, defined as follows:

$$B(i) = \{b_i, b_i - 28, b_i - 18, b_i - 17\}, \tag{3.2}$$

to denote the set of masses of the ions related to the b_i-ion. Let N_a be the total number of amino acids in a peptide. For peaks in an experimental spectrum, we use p_k to denote the value of the kth peak in the spectrum. Let N_p be the number of peaks in an experimental spectrum. If sequential search is used, the total number of comparison operations required for a b-ion group is $N_a \times 4 \times N_p$. A way to reduce the comparison operations is using the comparison result of p_k and b_i to eliminate the unnecessary comparison for every $b_i - 28, b_i - 18$ and $b_i - 17$ ion.

In the case of searching b-ions in peaks (corresponding to row vectors), only a few comparison operations can be reduced with searching results of b_i in peaks. The first step is to search b_i among the peaks sequence $\{p_k : 1 \leq k \leq N_p\}$. After this search, we will have the information about the lower bound and upper bound of b_i in p_k, $\{(k_{min}, k_{max}) : p_{k_{min}} < b_i < p_{k_{max}}\}$. Then for the other ions related to b_i, there is no need to compare them against $\{p_k : k_{max} \leq k \leq N_p\}$ because the mass of the $b_i - H_2O$, $b_i - NH_3$ and $b_i - CO$ ions are less than the mass of a b_i-ion. The upper bound k_{max} is also an upper bound of these ions. In this way, a few comparison operations can be saved depending on the search results of b_i. However, it is hard to find a lower bound for these ions given by k_{min} because the mass difference between adjacent peaks p_k and p_{k-1} is unpredictable. It can only be sure that $|p_k - p_{k-1}| \geq \delta$ where δ is the error tolerance of the instrument.

The other search direction, however, can help to reduce far more comparison operations. If a is an amino acid, then we use $m(a)$ to denote the mass of its residue. Let $P = a_1 a_2 ... a_k$ be the amino acid sequence of a peptide. The mass of the b-ion of P with i amino acids, denoted by b_i, can be computed with $b_i = 1 + \sum_{1 \leq j \leq i} m(a_j)$. As the minimum mass of the twenty common amino acid residues is 57.02 Daltons, the mass difference between two adjacent b-ions b_i and b_{i-1} is greater than or equal to 57.02 Daltons. Therefore, for a peak p_k in an experimental spectrum, after comparing it with b-ions, we can obtain the information about $\{x : b_{x-1} < p_k < b_x\}$. Then we need only compare p_k with $b_x - 28$, $b_x - 18$ and $b_x - 17$ respectively for potential matches with these ion types, if it is not matched with either b_{x-1} or b_x ions.

This b_x can be obtained during the first search for p_k in all the b-ions. So the total number of comparison operations is $(N_a \times 3) \times N_p$. With the SSE instruction set, the parallel version needs only $(N_a \times 3) \times N_p/4$ comparison operations as shown in Figure 3.5.

Elimination of Conditional Branches in Inner Loop

We abandon binary search in the SIMD version of search as binary search cannot bring significant improvements in efficiency when the underlying data set is small. Moreover, the numerous branches

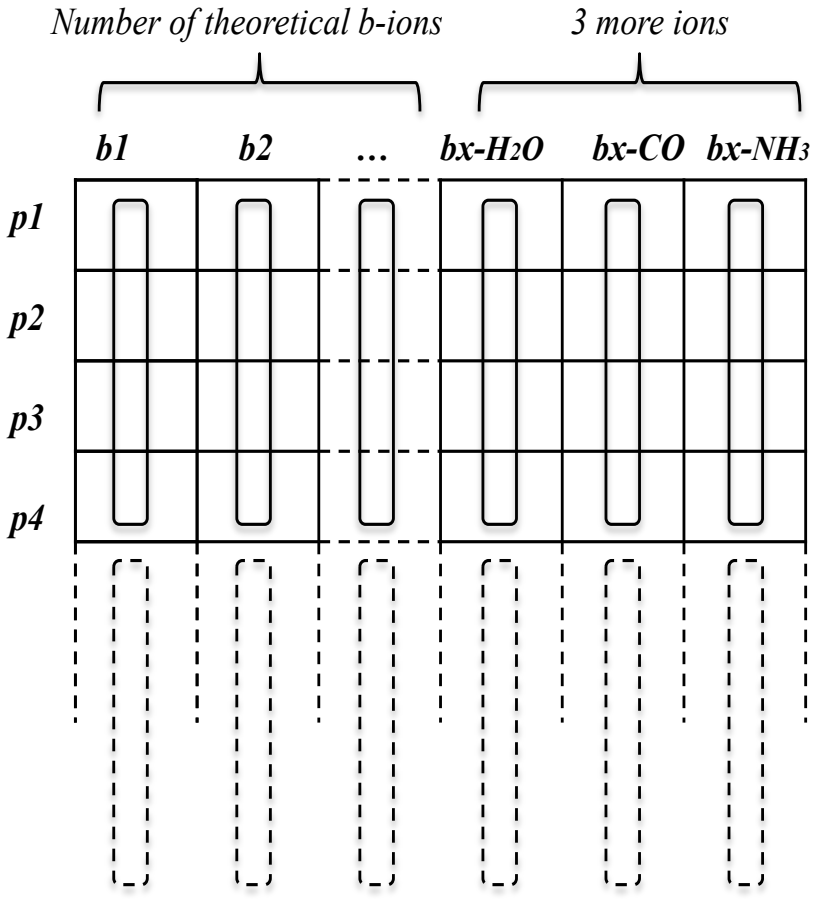

Figure 3.5: The number of comparison operations can be reduced using column vectors.

involved can slow down the program. Further, in order to implement binary search with the SSE instruction set, inefficient memory access operations like accessing non-aligned addresses, since shuffle loading must be used. The performance can be affected seriously as memory access latency is a very important factor for software performance nowadays.

It is not a difficult task to obtain the value of b_x during the searching for p_k among the b-ions. But the ordinary implementation with high level languages involves a few conditional branch statements and extra memory access. In our SIMD implementation, the corresponding vector b_x for four different peaks can be computed and stored in a SSE register throughout the inner loop of searching through b-ions without a single branch instruction. As the Intel SSE instruction set stores the comparison results of vector registers as the mask value, the b_x vector can be obtained using the SSE arithmetic, logic and comparison instructions. The data flow of computing for b_x is shown in Figure 3.6.

The ellipses in the figure represent the operations performed with two vector registers as the

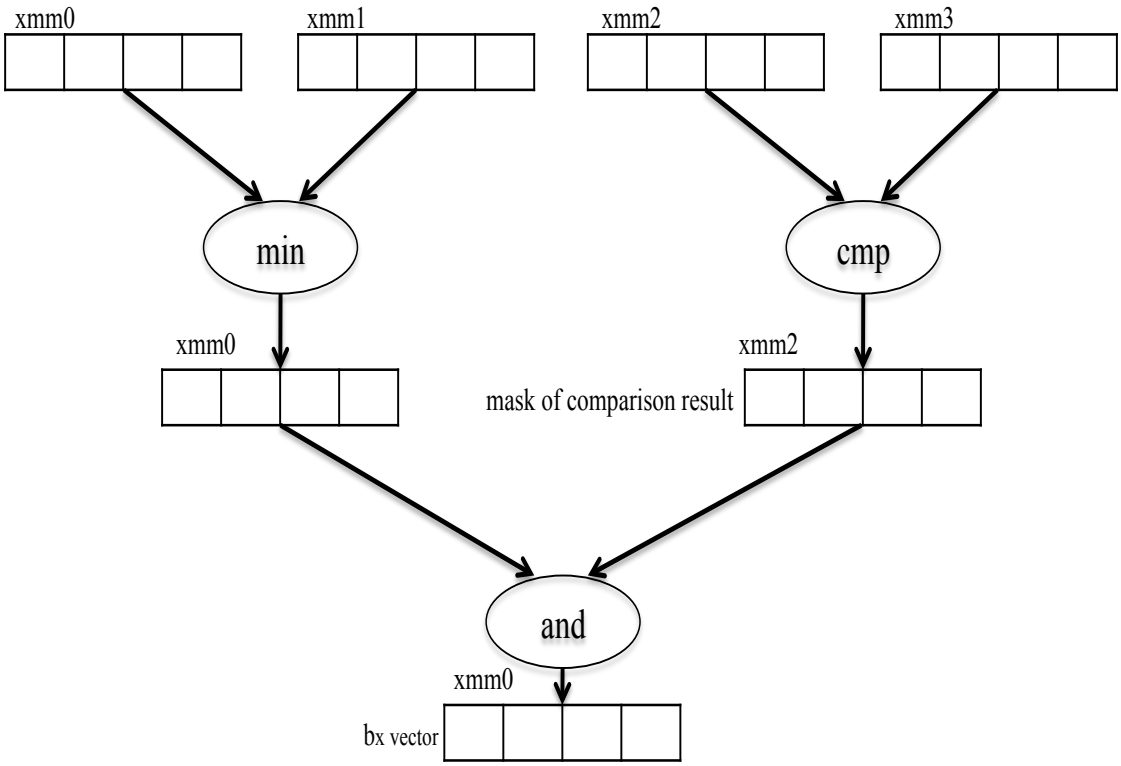

Figure 3.6: Data flow to compute the b_x vector.

operands. We denote the logic and operation by 'and' and the comparison operation by 'cmp'. The 'min' operation represents the computation taking the smaller values of the corresponding data elements in source operands and storing them in the destination operand.

The 'and' operation can be implemented directly using the SSE 'pand' instruction. The 'cmp' operation can also be implemented using the SSE instruction 'pcmpgtd', which will set the corresponding data element in the destination operand to all 1's if the data unit in the destination operand is greater than the data unit of the source operand. However, only the latest version of SSE supports the 'min' operation directly. For processors that do not support it, we can use a sequence of instructions according to the following algorithm from [39].

We define a 'doz' function to be the difference, or zero for signed arguments:

$$doz(x,y) = \begin{cases} x-y & x \geq y \\ 0 & x < y, \end{cases} \tag{3.3}$$

and the minimum can be computed with:

$$min(x, y) = x - doz(x, y). \tag{3.4}$$

This algorithm helps us to implement the 'min' operation in a branch-free way using a minimum number of registers.

General Code Optimization

In order to achieve higher speeds, we also did general optimization on our code. The memory used repeatedly in the calculation should preferably be contained in the first-level caches. So all the data needed for peptide-spectrum matching such as the experimental spectrum, theoretical spectrum and a few constants are grouped together to fit in the level-1 cache. Each data field has been aligned at addresses divisible by 16 for improving memory access speed.

A few vectors of constants are used in our code such as the error tolerance vector for comparison, the maximum signed integer vector for calculating b_x and the mass difference vector. Some vectors of constants are initialized outside the inner loop. We also use various instructions to generate trivial constants on the fly for reducing the memory access in the inner loop. As a result, there is no extra memory access in the inner loop except loading vectors of new peaks.

As the code using SSE instructions was written in assembly language, the use of registers in a sequence of instructions may cause dependency between instructions. We did some rearrangements on the instruction sequence to maximize the pipeline efficiency.

3.4 Evaluation and Results

The parallel algorithm we introduced in this chapter follows the similarity scoring function defined in RT-PSM precisely. It is only the implementation that differs from the original RT-PSM. Indeed, the implementation of the parallel peptide-spectrum matching using SIMD instructions produces identical results to the original RT-PSM. Therefore, we focus on the performance evaluation.

The speed of the new algorithm was evaluated using the exact same data set that RT-PSM used as we implemented this algorithm based on the original RT-PSM directly. We developed a parallel version of the function *cscore* with the exact same function prototype. So it is easy to just plug in the parallel version of *cscore*, build both the parallel and original version with the same compiler and measure performance of the two versions. The two programs were tested respectively on exactly the same computer and operating system with exactly the same set of MS/MS spectra and peptide database. The program was written in C++ with inline assembler code and was compiled with Microsoft Visual Studio 2008. The test computer is an iMac with an Intel Core 2 Duo 2.4GHz processor and 2GB RAM running Windows XP as the operating system. Both

programs were running in batch mode. The time measurement was done on the process time for every 1000 spectra. Furthermore, in our test, both versions of RT-PSM were tested on single CPU core only.

We use the same dataset provided with the RT-PSM source package. The MS/MS spectrum dataset consists of 22577 peptide collision-induced dissociation spectra acquired using an LCQ DECA XP ion trap (ThermoElectron Corp.). The test protein dataset is a subset of the UniRef100 database (release 1.2) containing 44278 human protein sequences [1].

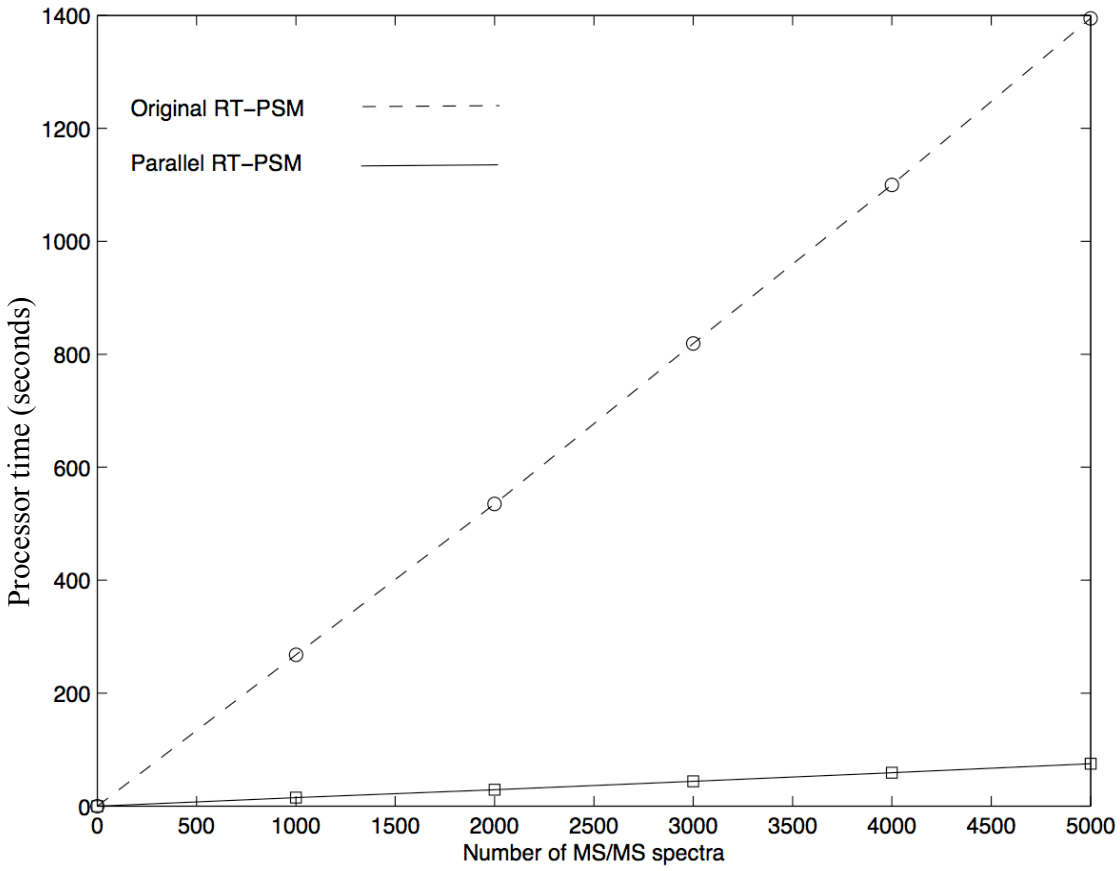

Figure 3.7: Speed of original RT-PSM and parallel RT-PSM

Plots of the number of spectra versus the process time for both programs are shown in Figure 3.7. In this figure, the processor time includes both user CPU time and system CPU time. On average, it takes the original RT-PSM approximately 268 seconds to process 1000 MS/MS spectra. Our implementation needs only 15 seconds, which is around 18 times faster than the original RT-PSM. In addition, the maximum processing time for a MS/MS spectrum is around 33 ms in our test.

As a significant speedup is obtained, we again performed profiling on the SIMD version of RT-PSM. The profiling results are shown in Figure 3.8. From this figure, we can see that the

parallel peptide-spectrum scoring algorithm using SIMD instructions is no longer the only dominant module in RT-PSM. As the computation time of the scoring module is decreased, the computation of candidate peptides selection becomes more important in the whole process.

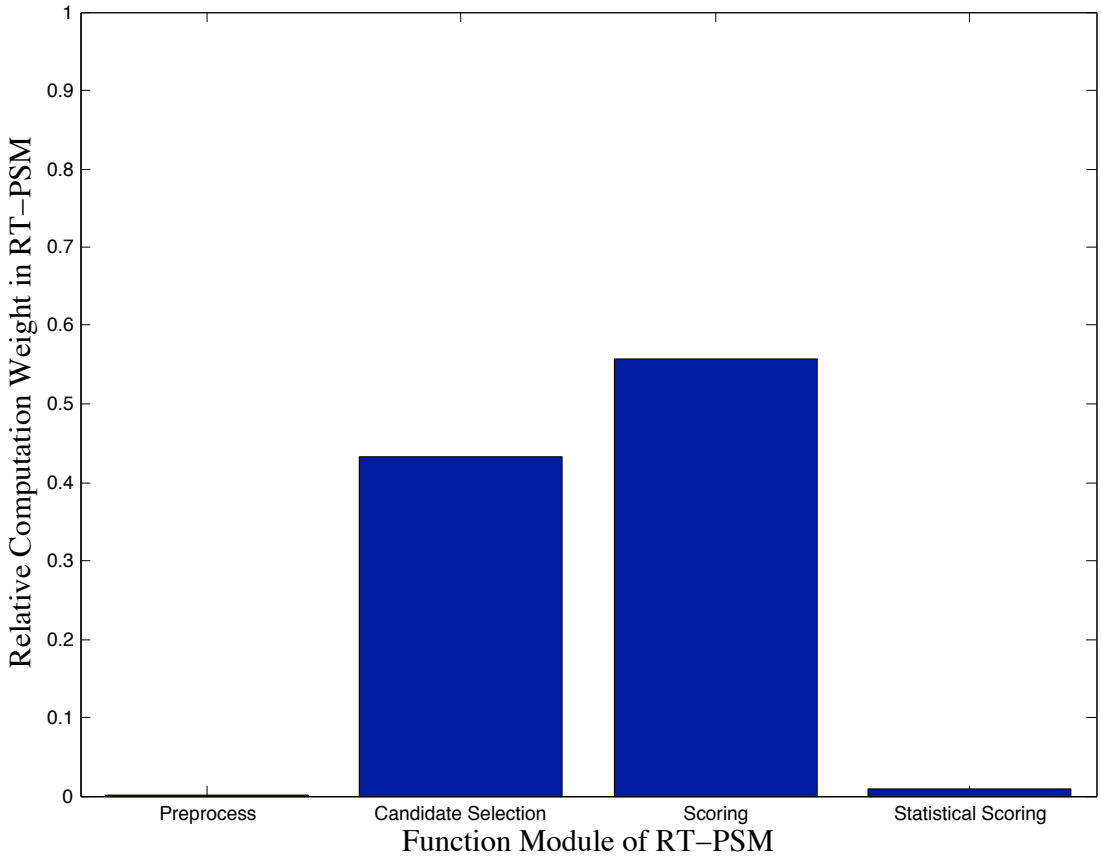

Figure 3.8: Both the candidate peptide selection and peptide-spectrum scoring modules are dominant in the SIMD version in contrast to Figure 2.3 of Chapter 2.

Chapter 4

Parallelizing RT-PSM with NVIDIA CUDA Technology

Parallelizing RT-PSM using the SIMD instruction set turns out to be very effective. However, it has its own limitations. Programming with SIMD instructions is difficult as we have to learn the SIMD instruction set and program at the instruction level. Moreover, the performance of the parallel algorithm using SIMD instructions is limited by the vectors supported by processors. For example, Intel SSE can only support 128-bit vector operations. However, the *Graphical Processing Units* (GPUs) are more powerful computation devices which overcome these limitations and offer a more powerful programming architecture in many circumstances.

4.1 Introduction

In recent years, GPUs have gained the attention of various application developers as common data-parallel coprocessors. Though the development of GPUs was originally driven by the 3D gaming industry for high-performance, real-time graphics engines, today GPUs have become accessible for general purpose computations. The new generations of GPU architectures offer easier programmability and increased generality. In addition, due to the nature of graphics computation, modern GPUs have evolved into highly parallel, many-core processors with tremendous computational power. This allows GPUs to be especially suited to address many problems that can be expressed as data-parallel computations.

In 2006, NVIDIA introduced CUDA, a *general purpose parallel computing architecture*, which enables NVIDIA GPUs to solve many complex computational problems using data-parallel algorithms [8]. Though similar technology is also available on other brands of graphic cards such as AMD, in this research we exclusively focus on NVIDIA CUDA only.

4.1.1 Hardware Architecture of CUDA-capable GPUs

Many graphic applications demand for compute-intensive, highly parallel computations such as per pixel or per vertex computations. The same algorithm is applied to every pixel in a image,

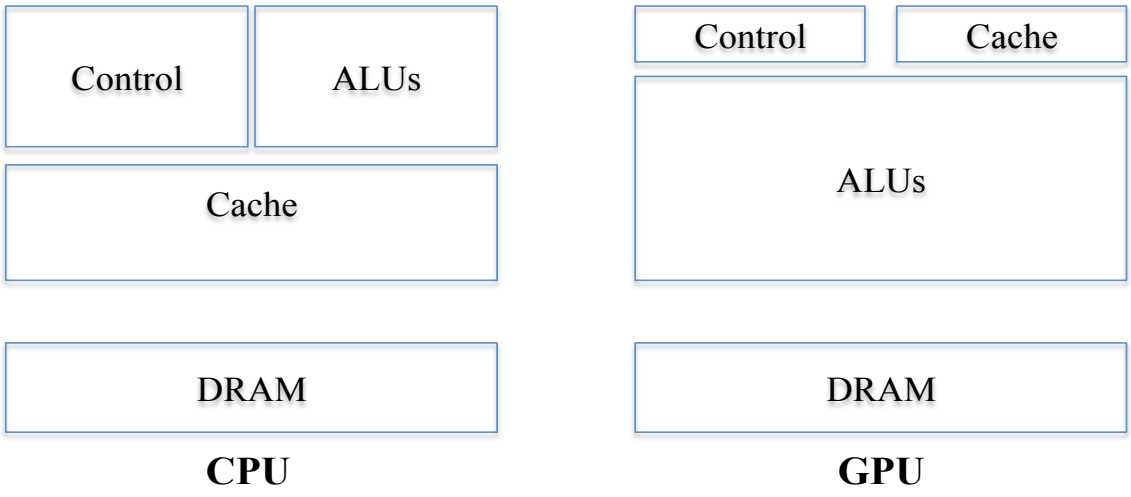

Figure 4.1: The GPU design in contrast to CPU design (modified from [8]).

or vertex in the geometric models. Therefore, GPUs are designed to devote more transistors to data processing instead of data caching and flow control [6]. This constitutes the major difference between the CPU and GPU designs shown in Figure 4.1.

This figure illustrates that GPUs contain far more arithmetic logic units than general CPUs. Therefore, the same program with high arithmetic intensity can be executed on many data elements in parallel, highly efficiently. Another important benefit of this architecture is that the memory access latency can be hidden by calculations if the program includes far more computation than memory access.

Figure 4.2 depicts a typical hardware architecture of a CUDA-capable NVIDIA GPU. A CUDA device has a set of *streaming multiprocessors*. Each multiprocessor contains a set of 32-bit *streaming processors* with a SIMD architecture [8]. The processors in the same multiprocessor share the same instruction unit. The instruction unit broadcasts the current instruction to these processors and each processor executes the instruction in a SIMD fashion.

Each multiprocessor has a large set of registers which can be dynamically allocated among threads running on it. Besides registers, each multiprocessor also has on-chip shared memory, a read-only constant cache and a read-only texture cache. The shared memory can be used for data sharing among threads running on the same multiprocessor. The constant and texture cache can speed up access to constant and texture memory on device memory.

4.1.2 Programming and Execution Model

The CUDA programming model extends the standard C language with additional keywords and structures that allow developers to implement computation directly on the GPU in a familiar C

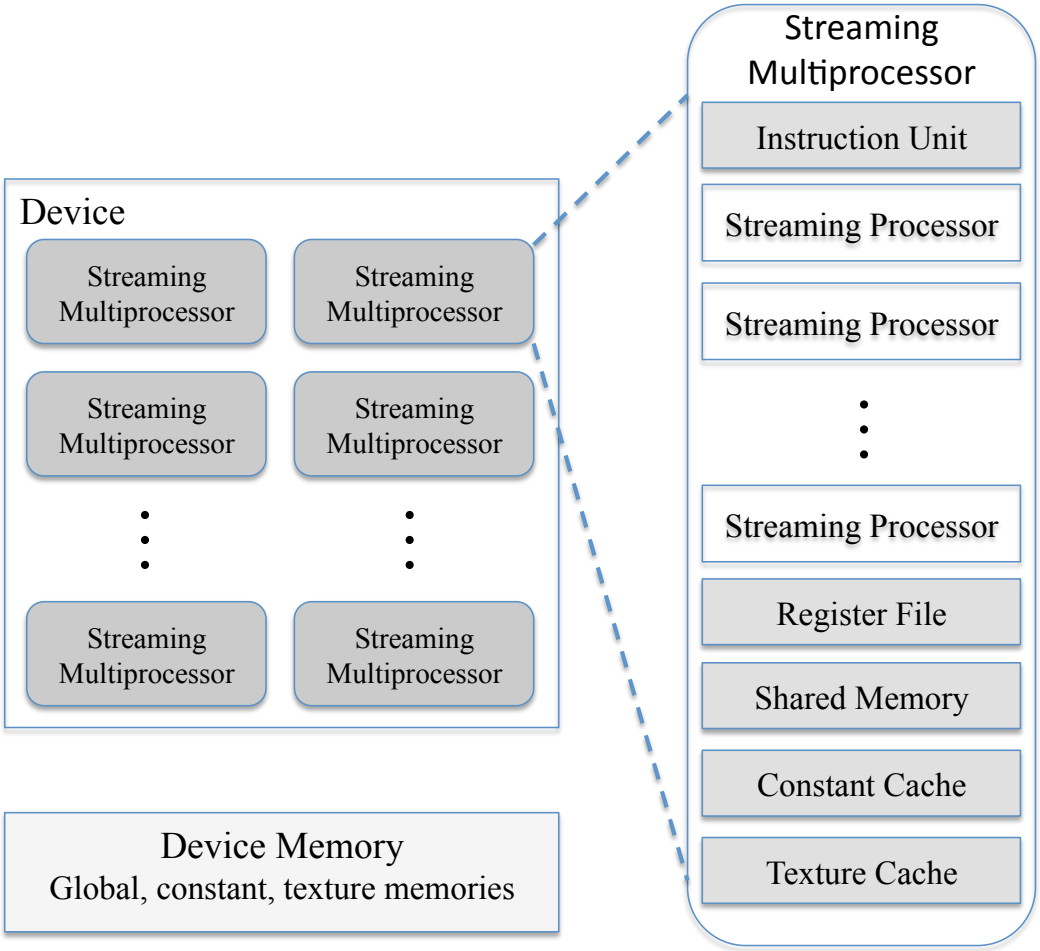

Figure 4.2: This figure illustrates the typical architecture of CUDA device (modified from [8]).

programming style. Developers can define C functions that are so-called *kernels*, which specify the code to be executed in parallel by different CUDA *threads*. In general, it is the data-parallel portion of an application that is executed on the CUDA device as kernels.

When a kernel is invoked, it is executed as a grid of parallel threads. Each grid typically contains thousands of CUDA threads. Threads in a grid are further grouped into a second level hierarchy, so-called *thread blocks* shown in Figure 4.3. Each grid consists of one or more thread blocks and all blocks in a grid have the same number of threads. Developers can specify the configuration of blocks and threads when invoking the kernel. All threads share the device data memory space. Only threads in the same thread block can cooperate with each other by efficiently sharing data through a shared memory. Two threads from two different blocks cannot cooperate.

The CUDA run-time system creates a grid of threads when a kernel is launched. These threads are assigned to hardware execution resources on a block-by-block basis. Each thread block is

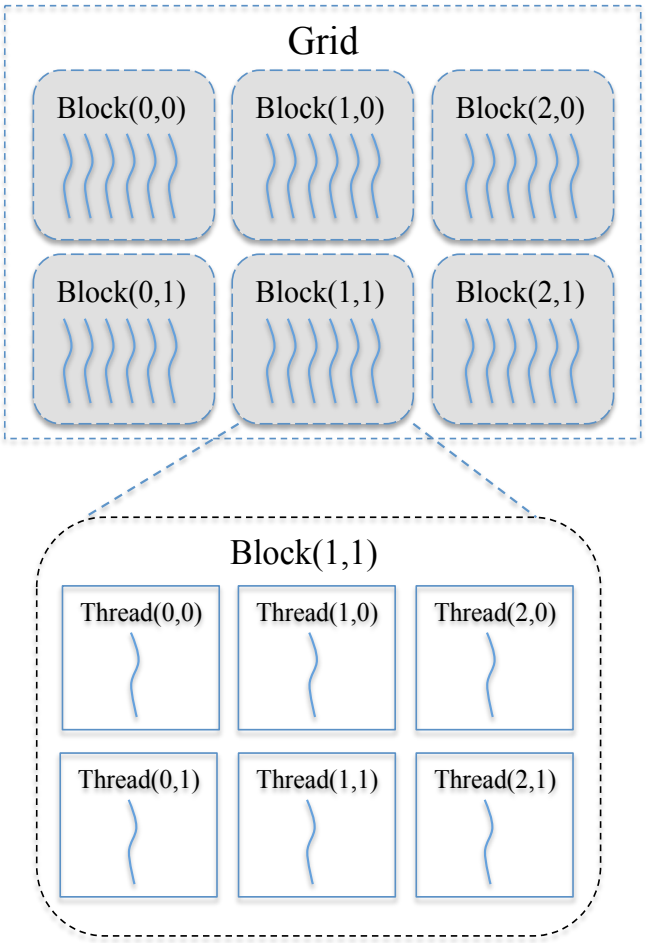

Figure 4.3: A kernel is executed as a grid of thread blocks (modified from [8]).

mapped to single streaming multiprocessor. When the number of blocks is more than the number of streaming multiprocessors, multiple blocks will be assigned to a single streaming multiprocessor. There is a limit on the number of blocks that can be assigned to a multiprocessor. In practice, this number is also limited by the resources needed for the blocks.

Though the exact thread scheduling depends on the implementation of the hardware, the basic concept is common in most existing CUDA GPUs. Once a thread block is assigned to a multiprocessor, it is further divided into 32-thread units called *Warps*. Warps are the basic units of thread scheduling in a multiprocessor. At any point in time, only one of the warps in a streaming multiprocessor will be actually executed.

The hardware of streaming multiprocessors are implemented to be able to perform zero-overhead warp scheduling. This enables streaming multiprocessors to hide the long latency of operations such as access the global memory. When a warp needs to wait for the result of a previously initiated long-latency operation, the warp is then placed into a waiting area and another ready warp is

selected to run without switching cost. The multiprocessor will keep busy in this way without stalling.

CUDA employs a barrier synchronization method to coordinate parallel activities. Threads in the same block can cooperate using the function *syncthreads*. When this function is called in a kernel, all threads in the same block will be held at the calling location until every other thread reaches the point. However, threads from different blocks can only interact through global memory. More details of the CUDA programming model can be found in [6, 7, 8].

4.1.3 CUDA Memories

As shown in Figure 4.2, CUDA offers various types of memories. CUDA threads may access data from multiple memory spaces during their execution. However, different types of memory have different properties such as location, size, access latency and program scope. Table 4.1 summarizes the types of memories and their properties.

Memory	Location	Cached	Access	Access Time	Scope
Global	Off-chip	No	Read/Write	Slow	All threads
Local	Off-chip	No	Read/Write	Slow	One thread
Shared	On-chip	N/A - resident	Read/Write	Fast	All threads in a block
Constant	Off-chip	Yes	Read	Slow	All threads
Texture	Off-chip	Yes	Read	Slow	All threads

Table 4.1: CUDA provides various types of memories (modified from [8]).

Besides these memories, each CUDA thread can access registers allocated to it. Access to registers is the fastest data operation. Accessing shared memory is almost as fast as access registers while access to global memory and local memory has a long latency. The latency of reading constant or texture memory depends on whether the data is already in cache or not. If the data is in cache, the speed of reading is as fast as access registers. If it is not in the cache, reading it becomes very slow. The size of these memories are different too. The specification of the sizes of various memory depends on the CUDA device. In general, the size of shared memory and cache are far less than the size of global and local memory. Due to the limits of size and access time, memory usage plays a very important role in algorithm design for CUDA.

4.2 Method

4.2.1 Data Decomposition

CUDA offers a very flexible and complex architecture to achieve parallelism at the data level. There could be various ways to decompose the data for parallel computing in RT-PSM. We could

design CUDA kernels to achieve parallelism in a comparison operation matrix to compute the similarity score as we used in Chapter 3. However, this is not an efficient method with the CUDA architecture as the implementation would involve many synchronization operations and would tend to cause branch divergence (discussed in next section). Instead, we choose a more natural approach, which is to assign each CUDA thread to calculate the similarity score of a query spectrum and a peptide as shown in Figure 4.4.

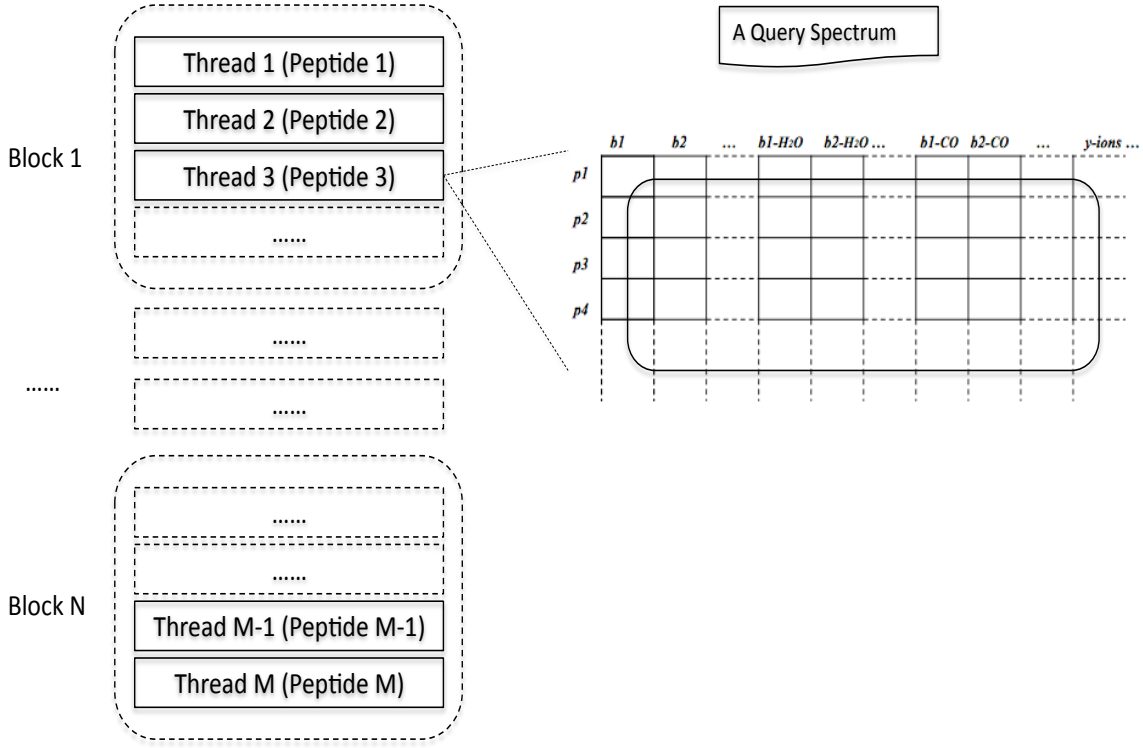

Figure 4.4: Instead of parallelizing the match matrix, each CUDA thread processes a peptide and the query spectrum independently.

A benefit of this parallel approach is that there is less requirement of cooperation among threads. Each thread takes the query spectrum and a peptide to calculate the score and writes it back to memory independently. One important motivation of RT-PSM is to provide real-time identification results of MS/MS spectra. This data decomposition scheme also meets this requirement as all the CUDA threads work on the same MS/MS spectrum simultaneously.

4.2.2 CUDA Kernels

The computation of the similarity score between the spectrum and a peptide sequence is implemented as a CUDA kernel function. There are also various algorithms for calculating a similarity score. As CUDA threads are executed in a SIMD fashion, and for the same reason described in

Chapter 3, we decided not to use binary search in the kernel function. As a means of exploring the applicability of CUDA in RT-PSM, we have investigated two different approaches to implement CUDA kernel functions for RT-PSM.

One approach is the straightforward algorithm similar to Algorithm 1 in Chapter 2. The masses of all theoretical ions are calculated based on a peptide sequence and sequentially compared with peaks in a spectrum. We denote this kernel by *Kernel 1* and the pseudo code for this kernel is shown in Algorithm 2.

Algorithm 2 Kernel 1

Input: *dgpeps* : an array of Peptide structure
dcspectrum : an array of peaks in query spectrum
speclen : the length of spectrum
Output: *g_cscore*: an array of similarity scores of the spectrum and peptides

```
idx ← blocksize * blockIdx.x + threadIdx.x; // global index of the thread
score ← 0;
k ← 0;
aa ← dgpeps[idx].str[k]; // extract the first amino acid
while aa ≠ 0 do
  bmass ← mass of b_k ion; // calculate the mass of b_k ion given by aa
  for j = 0 to speclen do
    peak ← dcspectrum[j];
    for all δ-ion such that δ ∈ the set of ions considered by RT-PSM do
      ionmass ← the mass of δ-ion calculated with bmass
      if |ionmass − peak| ≤ error tolerance then
        score ← score + weight for this match
      end if
    end for
  end for
  k ← k + 1;
  aa ← dgpeps[idx].str[k]; // get next amino acid
end while
normalize score
__syncthreads
g_cscore[idx] ← score
```

The other approach employs the similar optimization described in Chapter 3, which is to compare peaks against theoretical ions and utilize the bound to reduce comparison operations. We denote this kernel by *Kernel 2* and the pseudo code of this kernel is shown in Algorithm 3.

The pseudo code illustrates the fundamental algorithms of both kernels. In our implementation, we have further optimized these algorithms. We discuss the optimization methods we used in detail in the next section.

Algorithm 3 Kernel 2

Input: *dgpeps* : an array of peptide structure in global memory
dcspectrum : an array of peaks in query spectrum
speclen : the length of spectrum

Output: *g_cscore*: an array of similarity score of the spectrum and peptides

```
__shared__ speps[blocksize][maxLength] // two dimension array of peptide sequences
__shared__ spepmass[blocksize] // an array of the mass of peptides

idx ← blocksize * blockIdx.x + threadIdx.x; // global index of the thread
i ← threadIdx.x // local index in the block
for j = 0 to maxLength do
  speps[i][j] ← dgpeps[idx].str[j] // copy peptide sequences to shared memory
end for
spepmass[i] ← dgpeps[idx].mass
score ← 0
for j = 0 to speclen do
  peak ← dcspectrum[j];
  k ← 0
  aa ← speps[i][k]
  while aa ≠ 0 do
    bmass ← mass of b_k ion; // calculate the mass of b_k ion given by aa
    ymass ← mass of the complement y ion // calculate the y ion with spepmass and b_k
    if peak < bmass < bupper then
      bupper ← bmass // determine the upper bound
    end if
    if peak < ymass < yupper then
      yupper ← ymass // determine the upper bound
    end if
    if |bmass − peak| ≤ error tolerance then
      score ← score + weight for this match
    end if
    if |ymass − peak| ≤ error tolerance then
      score ← score + weight for this match
    end if
    k ← k + 1
    aa ← speps[i][k]
  end while
  for all δ-ion such that δ ∈ the set of ions considered by RT-PSM except b, y ions do
    ionmass ← the mass of δ-ion calculated with bupper and yupper
    if |ionmass − peak| ≤ error tolerance then
      score ← score + weight for this match
    end if
  end for
end for
normalize score
__syncthreads
g_cscore[idx] ← score
```

4.2.3 Kernel Optimization

Optimization plays a very important role in CUDA algorithm design due to the very complex parallel architecture provided by CUDA. Optimization methods are also different from those methods for CPUs because of the difference of architectures.

Optimizing For Occupancy By Recomputing

Kernel 2 uses the relationship of theoretical ions to get an upper bound and lower bound of the peak in searching as described in Chapter 3. Intuitively, it should be far more efficient than *Kernel 1* of the common sequential search as *Kernel 2* involves fewer comparison operations. However, we found that the straightforward implementation of *Kernel 2* is much slower than *Kernel 1*, even after our optimization. There are many factors that may affect the performance. Therefore, we applied NVIDIA visual profiler on both implementation of kernels. The key counters of the profiling output is shown in Table 4.2.

Kernel	GPU Time	Occupancy	Instructions
Kernel 1	452.625	0.5	97368
Kernel 2 (*straightforward implementation*)	991.872	0.031	58456

Table 4.2: The difference of occupancy causes the difference of performance.

We found that this performance difference between the two kernels was largely caused by *occupancy*. Occupancy is the metric related to the number of active warps on a multiprocessor and therefore it indicates how effectively the hardware is kept busy. Higher occupancy does not imply higher performance. However, low occupancy affects the ability to hide memory latency [7].

In order to implement the searching of a peak in the masses of ions, the masses of theoretical ions are calculated once and stored in memory so that searching for other peaks can reuse these values. This is the natural implementation and is efficient on CPUs. In CUDA programming architecture, the fast memory to hold these values is the shared memory. Nevertheless, the shared memory is a limited resource. The availability of shared memory is one of the important factors that determines occupancy. It turns out that the occupancy is significantly lowered by holding the mass values of theoretical ions in shared memory. We tried to use the global device memory to store the mass values. However, although the occupancy increased, it made the kernel even slower because of the high latency to access global memory.

According to the profiling output, the number of instructions executed by *Kernel 2* are far less than *Kernel 1*. Therefore, it could be feasible to trade some additional run time for less shared memory usage and in turn higher occupancy. We developed a solution that recomputes the mass values every time a peak is being processed. Therefore, we need only store the peptide sequence in

shared memory instead of the mass values. For a peptide consisting of N amino acids, it needs only N bytes to store the sequence while it takes $4 \times N$ bytes to store the masses of corresponding b-ions. The disadvantage of this method is that more computation is required. However, test results show that this optimization is very effective. It essentially enables *Kernel 2* to outperform *Kernel 1*.

Memory Allocation

CUDA threads can access data from various memory spaces during their execution. CUDA architecture offers this complex memory hierarchy to facilitate memory access. As shown in Table 4.1, the access speed and size vary according to the types of memory. Therefore, memory allocation for data storage is an important factor for performance. Table 4.3 illustrates the data allocation in our implementation of both kernels.

Data	Location in Kernel 1	Location in Kernel 2
candidate peptide sequences	global memory	global memory and shared memory
masses of candidate peptides	global memory	global memory and shared memory
masses of amino acids	constant memory	constant memory
query spectrum	constant memory	constant memory
array of scores	global memory	global memory

Table 4.3: Different kernels have different memory allocation schemes.

The data of candidate peptides is placed in the global device memory for both kernels because this piece of data is large and is sent from the host CPU. The constant memory holds the query spectrum and the masses of amino acid residues as they have small size. Furthermore, as they are accessed frequently, the cache for constant memory can facilitate access to them. *Kernel 1* does not move the peptide data to the shared memory as it turns out that each amino acid of a peptide sequence need be read only once in the algorithm of *Kernel 1*. In contrast, *Kernel 2* keeps a copy of peptide data in the shared memory as peptide sequence is accessed frequently. The array of similarity scores is placed in global device memory as it is transfered to the host when the execution of the kernel is completed.

Branch Divergence Reduction

As streaming multiprocessors execute CUDA threads in SIMD fashion, flow control instructions (e.g., if, switch, do, for, while) can significantly impact the effective instruction throughput by causing threads of the same warp to diverge [8]. In our work, the branch divergence is reduced by two approaches.

One is to remove data dependent controlling conditions by executing redundant instructions. For example, it is common to stop searching when a match is found. Instead, in our implementation,

we continue to perform redundant comparison operations even after a peak is matched to an ion. In this way, all the threads will execute the same path in processing a peak and much potential branch divergence is avoided.

The other approach is to use *branch predication* supported by CUDA [8]. When using branch predication, a per-thread condition code is associated with all the instructions whose execution depends on the controlling condition. None of the instructions gets skipped. Instead, each of them gets scheduled but only the instructions with a true condition code are actually executed. However, the compiler can only apply branch predication when the conditional statements are of a very simple form [7]. Therefore, in order to utilize branch predication, we use multiple simple flow control statements to replace those complex flow control statements.

4.2.4 Integration in RT-PSM

A minor modification of the RT-PSM algorithm is needed to integrate the CUDA peptide-spectrum matching code. Instead of extracting the peptide from the set of candidate peptides one by one, the whole set of candidate peptides are located and downloaded to the GPU by calling the CUDA API. The peptide-spectrum matching scores calculated by CUDA threads are copied back to host memory after the kernel execution is completed. Some other code for setting up the CUDA environment, such as the data copy and kernel launch also needs to be added into the RT-PSM program. The rest of the RT-PSM algorithm remains the same.

4.3 Results and Discussion

4.3.1 Experimental Setup and Dataset

These two parallel algorithms were developed with NVIDIA CUDA Toolkit 3.1. We performed tests and time measurements on a Mac Pro. This machine has an Intel Quad Core Xeon processor with a clock rate of 2.26 GHz. The CPU version of RT-PSM was tested on single core only. It is equipped with 6GB 1066 MHz DDR3 RAM and runs the Mac OS X (10.6.4) operating system. The graphics card is a NVIDIA GeForce GTX 285 card and the specification of this card is given by Table 4.4.

The dataset we used is the same dataset provided as the RT-PSM source package. The MS/MS spectrum dataset consists of 22577 peptide collision-induced dissociation spectra acquired using an LCQ DECA XP ion trap (ThermoElectron Corp.). The test protein dataset is a subset of the UniRef100 database (release 1.2) containing 44278 human protein sequences [1].

Graphic Card	GeForce GTX 285
CUDA Driver Version:	3.10
CUDA Runtime Version:	3.10
Total amount of global memory:	1,073,414,144 bytes
Number of multiprocessors:	30
Number of cores:	240
Total amount of constant memory:	65,536 bytes
Total amount of shared memory:	16,384 bytes
Total number of registers available	16,384
Warp size:	32
Clock rate:	1.48 GHz

Table 4.4: Both kernels are tested on an GTX 285 graphics card.

4.3.2 Speedup on the Similarity Score and the Kernel Comparison

The parallel algorithm on NVIDIA CUDA architecture also follows the similarity scoring function defined in RT-PSM exactly and produces the identical results as the original RT-PSM implementation. Therefore, we focus on the performance evaluation.

We first evaluated the speed of calculating similarity scores for a set of candidate peptides. The original CPU version of RT-PSM and the GPU version of both kernels are compared. As our parallel algorithms make no assumption on the dataset, we randomly sample some spectra and the corresponding candidate peptides. Then, we make an off-line program using the three versions respectively and tests them on the data. Table 4.5 lists the number of peptides processed and the computation time of CPU and GPU algorithms.

No. of peptides	CPU (original RT-PSM)	GPU Kernel 1	GPU Kernel 2
3072	77.715 ms	0.502 ms	0.410 ms
3328	81.395 ms	0.515 ms	0.417 ms
4096	83.853 ms	0.520 ms	0.432 ms

Table 4.5: The run time comparison illustrates the substantial speedup.

Both CUDA versions substantially outperformed the CPU implementation. *Kernel 1* obtains more than 150-fold speedup while *Kernel 2* achieves more than 190-fold speedup. Figure 4.5 shows the performance comparison between the two kernels.

4.3.3 Speedup on RT-PSM and New Profiling Result

As *Kernel 2* achieves the highest speedup, we integrated *Kernel 2* into RT-PSM and tested the throughput of the new version of RT-PSM. The comparison with the original RT-PSM is shown in Figure 4.6.

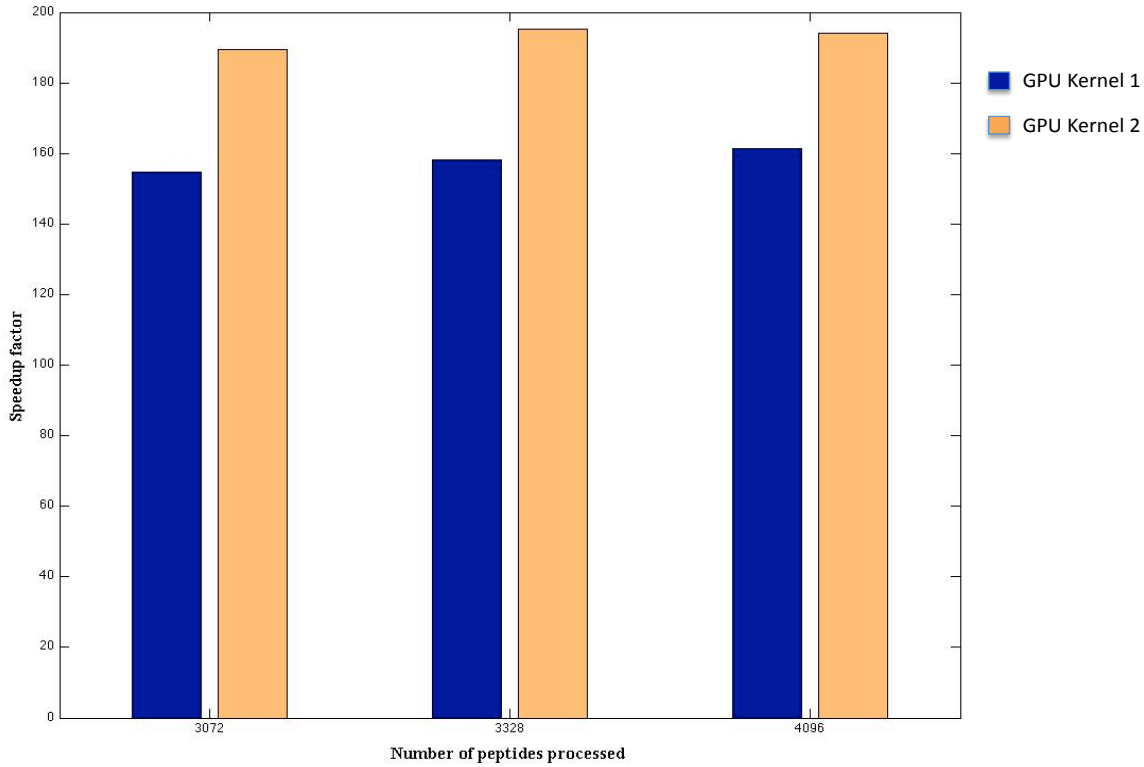

Figure 4.5: Speedup factors with respect to CPU version for both kernels are reported.

In Figure 4.6, the processor time includes both user CPU time and system CPU time. It takes the CPU version of RT-PSM approximately 198 seconds on average to process 1000 spectra while the CUDA version needs only approximately 7.5 seconds to do the same task. A 26-fold speedup has been achieved.

We noticed that the speedup factor on throughput is much lower than the speedup factor on the scoring module only. We also investigated the cause of this by profiling the CUDA version of RT-PSM. The profiling results are shown in Figure 4.7. From the figure, we can see that the candidate selection module consumes almost 90% of the computation time. The new profiling results clearly show that the scoring module is no longer the bottleneck of the whole process. The function module of selecting candidate peptides becomes the dominate module in the CUDA version. Our parallel scoring algorithm using CUDA technology makes the scoring module sufficiently fast for RT-PSM.

4.3.4 Instruction and Memory Bandwidth Analysis

In this section, we perform an estimation on instruction and memory bandwidth analysis of the *Kernel 2* algorithm. An estimation method was used in [22] and we will perform a similar analysis. Then we can find an upper bound on the performance of the algorithm running on CUDA. In doing

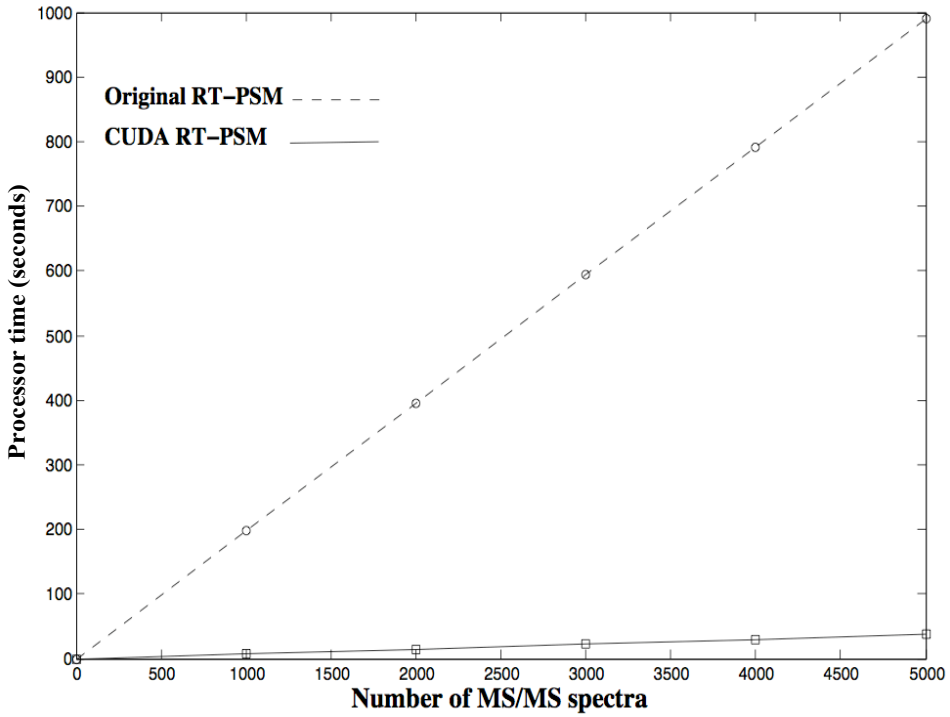

Figure 4.6: Parallel RT-PSM using CUDA technology achieves 26-fold speedup.

this estimation, we first obtain the memory bandwidth and the capability of instruction execution from the hardware specification. Then we calculate the number of memory access operations and estimate the number of instructions in the algorithm. With this information, we can estimate the peak performance that can be achieved on the hardware.

According to the official specification, the memory bandwidth of the NVDIA GeForce GTX 285 graphic card is 159.0 GB/sec. In the algorithm, the global memory is accessed at two points. One is to read in the peptide sequence into shared memory and the other is to output the similarity score. In our algorithm, we specify the maximum length of a peptide sequence as 32 Bytes. This is enough for peptides from digestion by Trypsin. In addition, as the query spectrum is stored in the constant memory which has a corresponding on-chip cache, access to the query spectrum is irrelevant to the global memory bandwidth. Therefore, the total memory access for a single peptide-spectrum pair is $32 + 4 = 36$ Bytes. Then, the memory bandwidth-bound is $1.59 \cdot 10^{11} \div 36 = 4.4167 \cdot 10^9$ Hz. This indicates that in terms of memory bandwidth and the algorithm, a total of at most $4.4167 \cdot 10^9$ peptide-spectrum pairs can be processed.

In order to estimate the instruction bound, we use the compiler for CUDA to generate the *Parallel Thread Execution* (PTX) code of our algorithm. Though PTX code is not the actual object code executed by the hardware, it is a low level media code and can be considered as a good

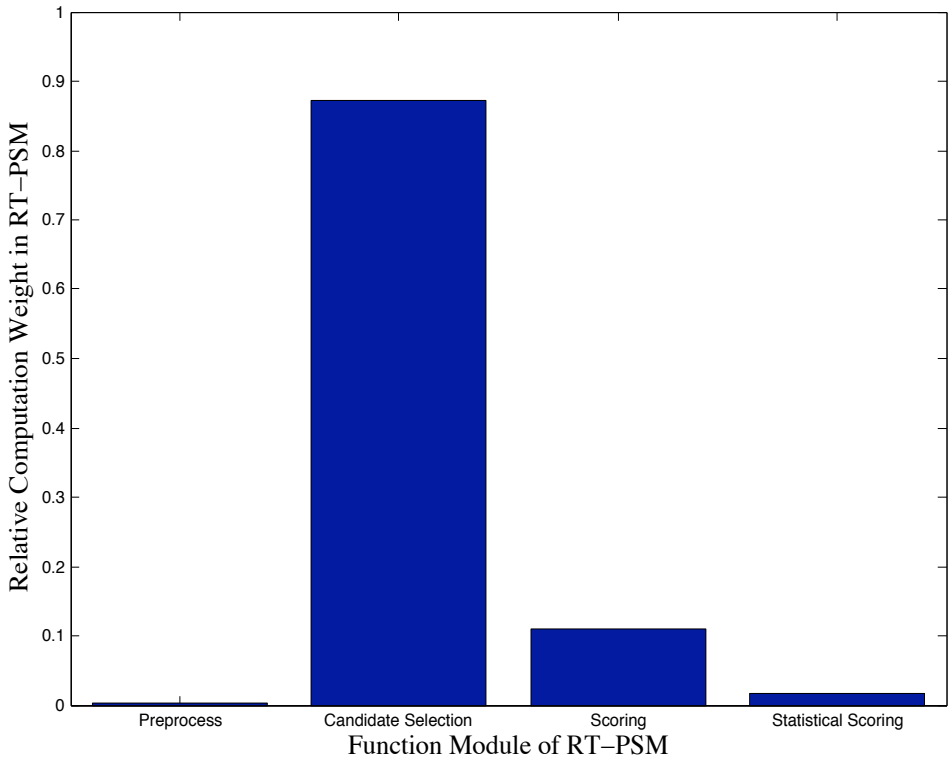

Figure 4.7: The profiling results show that candidate peptides selection becomes the bottleneck in the case of CUDA RT-PSM. This is in contrast to Figure 2.3 in Chapter 2.

reference to estimate the instruction bound.

According to the PTX code of the algorithm, there are a total of 93 instructions before the first layer loop. As shown in Algorithm 3, the algorithm consists of two nested loops. The second depth loop consists of a total of 76 instructions. Though there are some branch instructions executed, in order to estimate the upper bound, we assume all the instructions have to be executed in each run. There are 34 more instructions outside the inner loop and inside the outer loop. Finally, there are 8 instructions after the main loop. The iteration time of the outer loop depends on the number of peaks of the spectrum that the algorithm will consider. We denote the number of peaks in a query spectrum by N. The iteration time of the inner loop is fixed as the maximum length of peptides that the algorithm will consider. In our algorithm, it is set to 32. Thus, the total number of instructions required to execute the whole algorithm is $N \times 32 \times 34 + N \times 76 + 93 + 8$.

The NIVIDIA GeForce GTX 285 graphic card consists of 240 CUDA cores. The clock of the processor is 1476 MHz. Given by the hardware specification and our estimation of instructions, the compute-bound can be estimated by

$$\text{compute-bound} = \frac{1.476 \cdot 10^9 \times 240}{N \times 32 \times 34 + N \times 76 + 93 + 8}. \tag{4.1}$$

As the compute-bound varies on the number of peaks of a query spectrum, we calculated the compute-bound for several different sizes of spectra. The estimation results are listed in Table 4.6.

No. of peaks	Compute-bound	Bandwidth-bound
10	$3.0171 \cdot 10^7$ Hz	$4.4167 \cdot 10^9$ Hz
20	$1.5151 \cdot 10^7$ Hz	$4.4167 \cdot 10^9$ Hz
30	$1.0115 \cdot 10^7$ Hz	$4.4167 \cdot 10^9$ Hz
40	$7.5918 \cdot 10^6$ Hz	$4.4167 \cdot 10^9$ Hz
50	$6.0761 \cdot 10^6$ Hz	$4.4167 \cdot 10^9$ Hz

Table 4.6: Estimated compute-bounds and bandwidth-bounds.

From the estimation results, we can clearly see that the compute-bound numbers are much smaller. Though this analysis is highly idealized, it indicates that our parallel algorithm of similarity scoring is compute-bound on our test CUDA hardware. From this observation, further optimization should be focused on reducing computation complexity rather than memory access.

Chapter 5

Indexing the Peptide Database With Metric Data Structures

Chapters 3 and 4 provide significant speedups while still giving identical results from RT-PSM. In this chapter, we commence a preliminary investigation of other speedup approaches that are possible, but sacrifice some accuracy for additional speed gains. The chapter is meant only as a preliminary discussion of some approaches that are possible.

Another approach to improve the performance of tandem mass spectrum identification is to reduce the number of candidate peptides. RT-PSM already employs a mass difference filter (Equation 2.1) to choose a subset of the whole peptide database as candidate peptides. The amount of candidate peptides selected by the mass difference filter is already far less than the total number of peptides in the database. Hence, speedup with this approach can only be obtained by reducing the number of candidate peptides further.

In order to improve the speed further, we would need to filter out more peptides which are not likely to give high similarity scores. Filtering methods based on the so-called, "metric space indexing" have been introduced in mass spectrometry algorithms. *Multiple Vantage Point* (MVP) trees were introduced in [30]. Another different indexing method based on local sensitivity hashing was introduced in [12]. In this chaper, we introduce our evaluation methods and results of applying metric indexing methods to RT-PSM.

Unlike other common peptide identification software, RT-PSM has its own special requirements for identification of MS/MS spectra and for candidate peptide selection. Firstly, RT-PSM has a self-defined similarity scoring function. Secondly, as RT-PSM was developed with the goal of implementing real-time control of data acquisition, the run time is argubly more important than the accuracy, compared to other Tandem MS algorithms. Thus, we seek methods that can reduce the number of candidate peptides further based on the candidate peptides selected by the mass difference filter. Methods that may generate a *larger* number of candidate peptides than the current method are not considered. Moreover, the candidate peptide selection itself should not consume too much run time. Thirdly, statistical significance scores play an important role in the RT-PSM algorithm. According to the algorithm of calculating statistical significance scores, it needs not only

the highest matching score but the set of other matching scores as well. Those random matching scores are indispensable for calculating statistical significance scores. Because of this, a set of candidate peptides that is too small may affect the accuracy of statistical significance scores. We need to maintain a suitable number of candidate peptides.

Due to these requirements, we choose to evaluate two indexing methods. Both are similarity distance based indexing methods. One uses the *triangle inequality* based branch-and-bound algorithm in searching, and an MVP tree is a typical indexing structure of this kind. The other provides a heuristic searching algorithm without using the triangle inequality and it is called SASH [19].

5.1 Similarity Distance function

The reason we choose distance-based indexing methods is that it is easy to customize the similarity distance function. As our goal is to apply these indexing methods in RT-PSM, and RT-PSM defines its own similarity scoring function, we define the similarity distance function based on the similarity scoring function already used in RT-PSM. The candidate peptide selection is to select a certain number of peptides, which are most likely to give high similarity scores. In other words, they are the set of nearest neighbours to the query spectrum based on our similarity distance definition. Thus, the candidate peptide selection can be modelled by the *k-Nearest Neighbours* (k-NN) search. Furthermore, by choosing a distance function similar to the similarity scoring function of RT-PSM, we can reuse the calculation results of the similarity distance calculation in the k-NN search.

In Chapter 2, we introduced the similarity score used in RT-PSM. Though the scoring function employs a weighted summation and normalization, it largely relies on the shared peak counts. Therefore, we define our similarity distance function based on shared peak counts as well. In addition, the similarity distance function is also defined to follow the rule that the shorter the similarity distance between two data points, the more similar they are as part of the distance function.

We define the similarity distance of two spectra A and B as

$$Dist(A, B) = ||A|| - SPC(A, B) + ||B|| - SPC(A, B), \tag{5.1}$$

where, $SPC(A, B)$ denotes the number of shared peak counts between the two spectra A and B, and the $||A||$ and $||B||$ denote the total number of peaks in A and B, respectively.

5.2 Evaluation of Triangle Inequality Based Methods

5.2.1 Introduction of Pruning Rules and Evaluation Method

We first evaluate the feasibility of indexing methods using branch-and-bound searching algorithms based on the triangle inequality. The performance of the branch-and-bound searching algorithm depends on how many elements can be pruned in a search. The triangle inequality is the basis for the pruning rules of most distance-based index structures. A comprehensive introduction to pruning rules can be found in [35]. We describe an example of pruning rules based on the triangle inequality as shown in Figure 5.1.

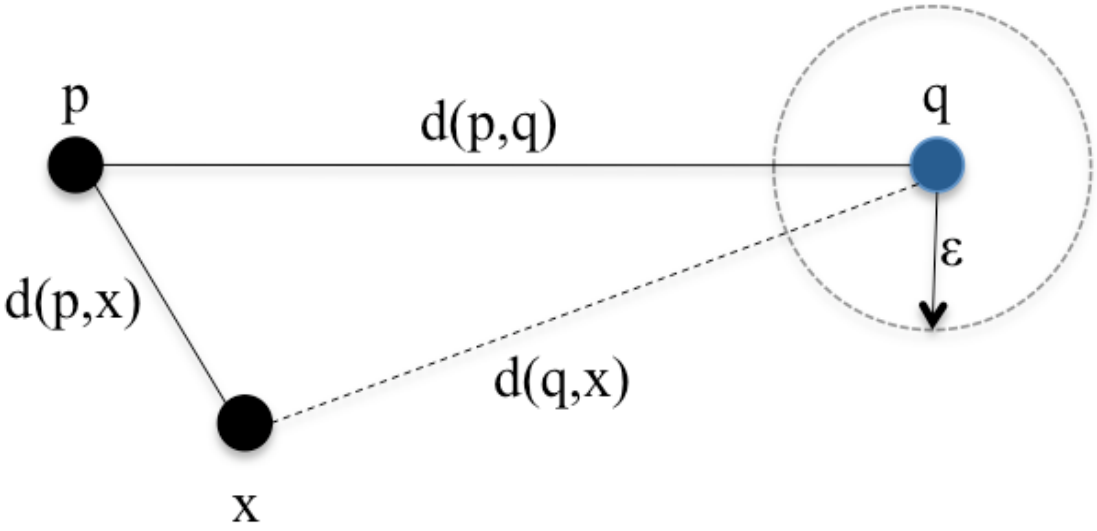

Figure 5.1: Some data points can be eliminated by pruning based on the triangle inequality.

In this figure, the data point q represents the query object. We let p and x denote two other data points and we use d to denote a distance function assuming the triangle inequality holds as well. Then, $d(p, q)$ is the distance from q to p and $d(p, x)$ is the distance from x to p. Both of them are known. But $d(q, x)$, the distance from q to x, is unknown. Most k-NN search algorithms maintain a set of the current k candidate nearest neighbours, and the distance of the farthest of the current k candidate nearest neighbours can be calculated and used in pruning rules. In Figure 5.1, we denote this value by ε.

According to the triangle inequality, we know that $d(q, x) \geq |d(p, q) - d(p, x)|$. As $d(p, q)$ and $d(p, x)$ are known, we can compute the value of $|d(p, q) - d(p, x)|$. If $|d(p, q) - d(p, x)| \geq \varepsilon$, then $d(q, x) \geq |d(p, q) - d(p, x)| \geq \varepsilon$. This means that x cannot be a member of the k-Nearest Neighbours

of q. Therefore, we can eliminate x from consideration and avoid the explicit computation of the distance from x to q.

The more data points that can be eliminated from consideration, the more efficient that the branch-and-bound searching becomes. However, if the pairwise distances between these points are very close, the pruning rules will hardly have an opportunity to be applied. For example, if $d(p,q) = d(p,x)$, we can only conclude that $d(q,x) \geq 0$ based on the triangle inequality. In this case, no matter what the value of the ε is, we cannot eliminate x based on the estimation of $d(q,x)$.

This is a typical problem called *the curse of dimensionality* [4] when dealing with high dimensional data. From a statistical perspective, if the probability density function of the pairwise distances of various data points is too concentrated, the nearest neighbours searching will be quite inefficient. A comprehensive discussion of the curse of dimensionality can be found in [35]. In other words, we can estimate the efficiency of the branch-and-bound searching algorithm by checking the variance of the pairwise distances. If the variance is small, then the triangle inequality based searching strategy will not be efficient.

5.2.2 Evaluation Experiment and Results

As our purpose is to reduce the number of candidate peptides further, we work on the current candidate peptides selected by the mass difference filter of RT-PSM. Thus, in the evaluation experiment, we first arbitrarily choose an experimental MS/MS spectrum. Then we calculate the pairwise distances on the candidate peptides selected by the mass difference filter and give the distribution graph of these distances. A typical distribution of the pairwise distances of a candidate peptide set is shown in Figure 5.2.

From the graph, we can see that the distribution of pairwise distances is mostly concentrated on the small range from 200 to 300. This indicates that the distribution of pairwise distance has a very small variance. We derived very similar results on several other MS/MS spectra. Furthermore, our results are similar to the paper [30] though we use a different distance function. Although this only represents a preliminary evaluation, we suspect that the approach might not be suitable for RT-PSM. However, further testing with a full implementation with different distance functions would be required to conclusively rule out the approach as a whole. This is beyond the scope of our thesis, and we leave further analysis as part of the future directions.

5.3 Evaluation of SASH

5.3.1 Introduction of SASH

Spatial Approximation Sample Hierarchy (SASH) [19] is another algorithmic approach that employs

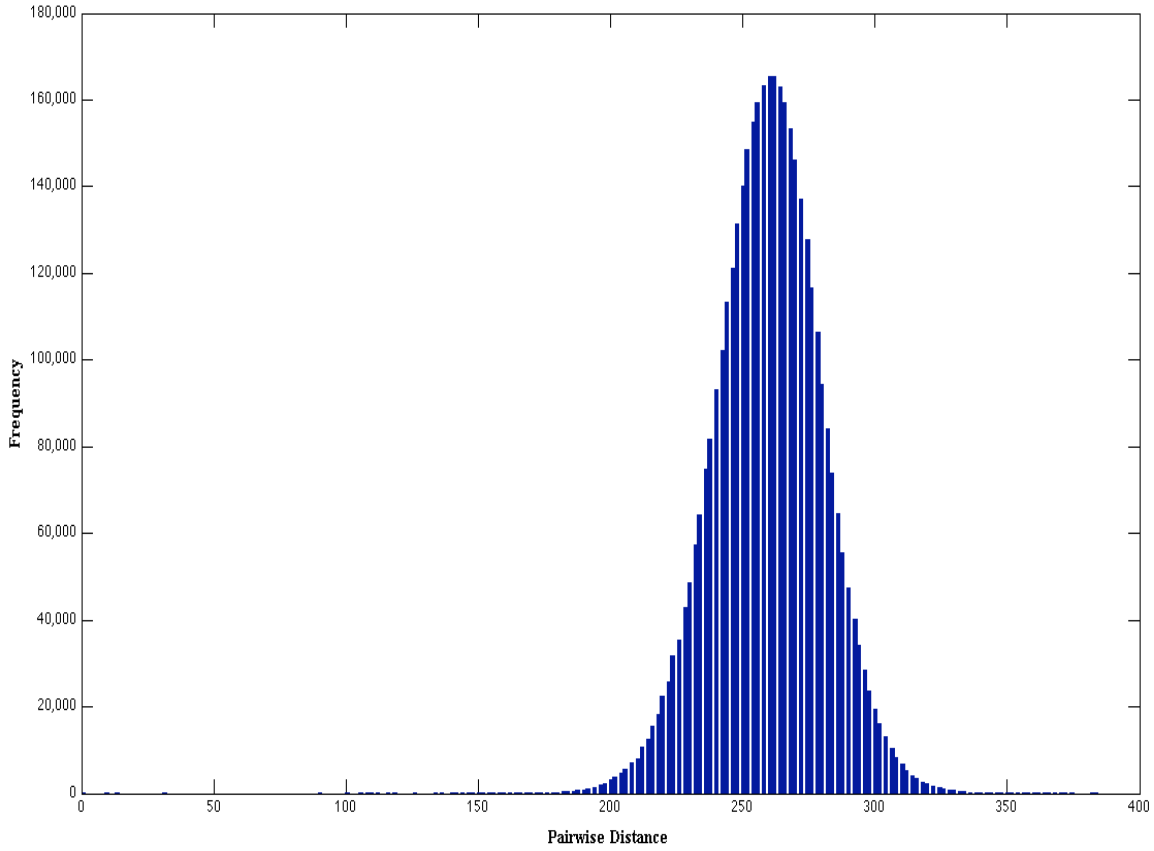

Figure 5.2: This is a frequency plots of the pairwise distances of a set of candidate peptides. The variance of pairwise distances is small.

an approximate search strategy without using the triangle inequality. Instead, SASH relies on heuristic information. It attempts to establish links to objects that are likely to be neighbours of the query object regardless of whether or not the triangle inequality is satisfied for the objects. When the k-NN search is performed on SASH, the objects that are not likely to be nearest neighbours are eliminated based on the established similarity links. In this way, some distance computation during the search can be avoided.

A SASH is a multi-level search structure constructed by building a smaller SASH on a random sample of roughly half the data objects. Then each remaining object is connected to several of their approximate neighbours from within the samples that are recursively determined using the smaller SASH. The k-NN search is processed by recursively locating approximate neighbours within the sample and then discovering neighbours within the remainder of the data set using the pre-established connections. Unlike MVP trees, the distance measure is used strictly for relevance ranking only. SASH generates neighbourhoods by refining an approximate neighbourhood set over an expanding progression of samples. A detailed introduction of SASH can be found in [19].

5.3.2 Evaluation of SASH

SASH provides an approximate nearest neighbours search strategy without applying the triangle inequality based pruning rules. Therefore, the curse of dimensionality might not affect the search strategy for SASH significantly. We implemented the SASH structure and tested it on the set of candidates selected by the mass difference filter as well. As SASH is an approximate method, it cannot ensure that the nearest neighbours is found correctly every time. This failure is significant for MS/MS spectrum identification as there can be only one true matching peptide for a spectrum. In RT-PSM, however, a low probability of incorrect identification may be acceptable as the identification result given by RT-PSM is mainly for real-time data acquisition. More accurate results can be obtained by post-processing using off-line identification software.

Therefore, we test the accuracy against the reduced number of candidate peptides. There are parameters in the implementation of SASH that can be used to control the number of distance computations in a k-NN search. Then the test proceeds as follows:

1. Perform sequential search with the similarity distance function to find the nearest peptide given by an arbitrary experimental spectrum.
2. Build a SASH index and perform a k-NN search to see if the nearest peptide is found in the search results. Record the number of peptides checked during the search.
3. Do step 2 many times because the SASH index is built with random sampling process.
4. Compute the probability that the nearest one is found and take the average value of the number of peptides checked in a search. Store the result with corresponding parameter settings.
5. Vary the parameter settings to control the number of checked peptides, and perform the same test and collect the results.

No. of trial	Total number of peptides	Number of peptides checked	Probability of returning the true nearest neighbor
1	818	252	0.3
2	818	512	0.65
3	818	628	0.80

Table 5.1: The probability of finding the nearest neighbour is proportional to the number of peptides checked.

From the test result shown in Table 5.1, we find that the probability of finding the true nearest peptide is proportional to the number of peptides checked. This indicates that our implementation of the SASH index does not help to improve the accuracy. We also arbitrarily chose several more

MS/MS spectra to run the same evaluation test and obtained similar results. Although these results are not conclusive, and additional parameters and datasets would need to be used for a thorough analysis, the preliminary results suggest that the SASH method might not be better than random selection.

In this chapter, we introduced our work on exploring other approaches to improve the speed of RT-PSM further besides the parallel algorithms. Particularly, we introduced two potential methods to reduce the number of candidate peptides with different metric space indexing structures and described the preliminary evaluation results. Though we obtained negative results in our preliminary evaluation, our work can be helpful for future work on how to reduce the number of candidate peptides to be checked while improving the runtime, without severely impairing the accuracy.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we introduced profiling results of the RT-PSM function modules and found that peptide-spectrum scoring is the most time-consuming module. Then we explored two approaches to address the performance problem of RT-PSM according to the profiling results.

One approach is to reduce the number of candidate peptides. Following this approach, we evaluated the feasibility of applying a metric index structure such as a MVP tree, in which pruning rules based on the triangle inequality can be applied. We also implemented another metric data structure, SASH, which can facilitate search without using the triangle inequality. In Chapter 5, we described the evaluation method and gave the evaluation results of these two index structures. The results show that both indexing methods are potentially not feasible or practical for RT-PSM, although this analysis was only a preliminary investigation.

Rather than reducing the number of candidate peptides, improving the speed of peptide identification for MS/MS spectra is another approach to optimizing RT-PSM. In this thesis, we introduced two parallel algorithms for peptide identification in RT-PSM using different technology.

In Chapter 3, we introduced a parallel peptide-spectrum matching algorithm using SIMD instructions. We designed and implemented this algorithm on the Intel SSE architecture. Intel SSE registers are divided into four 32-bit data units so that four peaks in an experimental spectrum can be processed in parallel. Furthermore, we introduced an algorithm to reduce the comparison operations significantly. Combined with other code level optimizations, the test results show that an 18-fold speed increase over the original version of RT-PSM was obtained.

The other parallel algorithm using NVIDIA CUDA technology was introduced in Chapter 4. A powerful and flexible programming environment is provided by the CUDA architecture. Based on the CUDA architecture, we employed a different data decomposition scheme. Instead of parallelizing the processing of peaks, we parallelize the scoring for every spectrum-peptide pair. We described two different algorithms as the kernel functions in CUDA. Furthermore, we discussed various optimizations we did on these two kernels to improve performance. The test results show that a 190-fold speedup on the similarity scoring module is achieved while a 26-fold speedup on

the entire process is obtained. In Chapter 4, we also presented new profiling results of the CUDA version. The scoring module is no longer the dominant module. The candidate peptide selection module becomes the bottleneck of the whole process. In addition, we also performed an instruction and memory bandwidth analysis. The estimation reveals that our algorithm is compute-bound in the CUDA architecture.

Though our work on parallelization is completely based on RT-PSM, the peptide identification algorithm is actually similar to algorithms used by other common software packages. Therefore, our work can also be applied to other peptide identification software with minor modification.

In this thesis, we proposed two different parallel computing methods to optimize RT-PSM. One important motivation is to provide real-time identification results to the dynamic exclusion list algorithm implemented in mass spectrometers. Thereby, these two methods we introduced are actually two different solutions with a different cost and performance for mass spectrometer instrument makers. As SIMD instructions are now widely available on most modern processors, the method using SIMD is a low cost solution. However, this method is not flexible and its performance is not as good as the method using CUDA. The parallel algorithm using CUDA technology is far more flexible and scalable. The same algorithm can easily be used on a more powerful GPU without modification to achieve better performance. It is also very convenient to modify the algorithm for different requirements. However, the instrument makers must invest in the hardware for CUDA and this method could be a costly solution.

6.2 Future Work

There are several possible directions to extend our work.

1. In this thesis, we have evaluated two indexing structures based on peptide databases and received preliminary negative results. However, there are various other indexing methods for high dimensional data. In particular, if we can add in the information of additional properties of the data, it may be possible to create some heuristic indexing structures for k-Nearest Neighbours search.

2. The profiling results on the CUDA version of RT-PSM show that the candidate peptide selection has become the bottleneck of the whole process. We may work on optimizing this function module to achieve better performance.

3. The parallel algorithms we proposed in this thesis make the similarity scoring computation sufficiently fast. This gives much room for applying more complicated and sophisticated scoring computation for better accuracy.

References

[1] Universal Protein Resource. http://www.uniprot.org.

[2] DeconMSn. Pacific Northwest National Laboratory, 2010. http://omics.pnl.gov/software/DeconMSn.php.

[3] C. Bartels. Fast algorithm for peptide sequencing by mass spectroscopy. *Biomed. Environ. Mass Spectrom.*, 19:363–368, 1990.

[4] R. E. Bellman. *Adaptive Control Processes*. Princeton University Press, Princeton, NJ, 1961.

[5] Carl Branden and John Tooze. *Introduction to Protein Structure*. Garland Publishing, Inc., New York, 1998.

[6] NVIDIA Corp. *NVIDIA CUDA Architecture*. 2009. http://developer.download.nvidia.com/compute/cuda/docs/CUDA_Architecture_Overview.pdf.

[7] NVIDIA Corp. *NVIDIA CUDA C Best Practices Guide*. 2010. http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CUDA_C_Best_Practices_Guide.pdf.

[8] NVIDIA Corp. *NVIDIA CUDA C Programming Guide*. 2010. http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/NVIDIA_CUDA_C_ProgrammingGuide_3.1.pdf.

[9] Robertson Craig and Ronald C. Beavis. A method for reducing the time required to match protein sequences with tandem mass spectra. *Rapid Commum. Mass Sepctrom*, 17:2310–2316, 2003.

[10] Robertson Craig and Ronald C. Beavis. Tandem: matching proteins with tandem mass spectra. *Bioinformatics*, 20:1466–1467, 2004.

[11] Dexter T. Duncan, Robertson Craig, and Andrew J. Link. Parallel Tandem: a program for parallel processing of tandem mass spectra using PVM or MPI and X!Tandem. *Journal of Proteome Research*, 4:1842–1847, 2005.

[12] Debojyoti Dutta and Ting Chen. Speeding up tandem mass spectrometry database search: metric embeddings and fast near neighbor search. *Bioinformatics*, 23:612–618, 2007.

[13] Jimmy K. Eng, Ashley L. McCormack, and John R. Yates III. An approach to correlate tandem mass spectral data of peptides with amino acid sequence in a protein database. *American Society for Mass Spectrometry*, 5:976–989, 1994.

[14] David Fenyö and Ronald C. Beavis. A method for assessing the statistical significance of mass spectrometry-based protein identifications using general scoring schemes. *Anal. Chem.*, 75:768–774, 2003.

[15] Helen I. Field, David Fenyö, and Ronald C. Beavis. Radars, a bioinformatics solution that automates proteome mass spectral analysis, optimises protein identification, and archives data in a relational database. *Proteomics*, 2:36–47, 2002.

[16] Christine L. Gatlin, Jimmy K. Eng, Stacy T. Cross, James C. Detter, and John R. Yates III. Automated identification of amino acid sequence variations in proteins by HPLC/Microspray tandem mass spectrometry. *Analytical Chemistry*, 72:757–763, 2000.

[17] C.W. Hamm, W.E. Wilson, and D.J. Harvan. Peptide sequencing program. *Computer Applications in the Biosciences*, 2:365, 1986.

[18] Moshe Havilio, Yariv Haddad, and Zeev Smilansky. Intensity-based statistical scorer for tandem mass spectrometry. *Anal. Chem.*, 75:435–444, 2003.

[19] Michael E. Houle. Fast approximate similarity search in extremely high-dimensional data sets. *Processdings of the 21st International Conference on Data Engineering*, pages 619–630, 2005.

[20] Lawrence Hunter. Artificial intelligence and molecular biology. *AI Magazine*, 11:27–36, 1990.

[21] Andrew Keller, Alexey I. Nesvizhskii, Eugene Kolker, and Ruedi Aebersold. Empirical statistical model to estimate the accuracy of peptide identifications made by MS/MS and database search. *Anal. Chem.*, 74:5395–5392, 2002.

[22] Theodore Kim. Hardware-aware analysis and optimization of stable fluids. *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pages 99–106, 2008.

[23] Michael Kinter and Nicholas E. Sherman. *Protein Sequencing and Identification Using Tandem Mass Spectrometry*. John Wiley & Sons, Inc., New York, 2000.

[24] Alex Klimovitski. Using SSE and SSE2: Misconceptions and Reality. *Intel Developer Update Magazine*, pages 1–8, 2001.

[25] Bin Ma. Challenges in computational analysis of mass spectrometry data for proteomics. *Science and Technology*, 25:107–123, 2010.

[26] Bin Ma, Kaizhong Zhang, and Chengzhi Liang. An effective algorithm for the peptide de novo sequencing from MS/MS spectrum. *Journal of Computer and System Sciences*, 70:418–430, 2005.

[27] Scott McGinnis and Thomas L. Madden. BLAST: at the core of a powerful and diverse set of sequence analysis tools. *Nucleic Acids Research*, 32:20–25, 2004.

[28] Alex Peleg and Uri Weiser. MMX technology extension to the Intel architecture. *Micro, IEEE*, 16:42–50, 1996.

[29] David N. Perkins, Darryl J. C. Pappin, David M. Creasy, and John S. Cottrell. Probability-based protein identification by searching sequence databases using mass spectrometry data. *Electrophoresis*, 20:3551–3567, 1999.

[30] Smriti R. Ramakrishnan, Rui Mao, Aleksey A. Nakorchevskiy, John T. Prince, Willard S. Willard, Weijia Xu, Edward M. Marcotte, and Daniel P. Miranker. A fast coarse filtering method for peptide identification by mass spectrometry. *Bioinformatics*, 22:1524–1531, 2006.

[31] Torbjørn Rognes and Erling Seeberg. Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics*, 16:699–706, 2000.

[32] Altschul S. The statistics of sequence similarity scores. NCBI: Cold Spring Harbor, 2004. `http://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html`.

[33] Rovshan G. Sadygov, Jimmy Eng, Eberhard Durr, Anita Saraf, Hayes McDonald, and Michael J. MacCoss. Code Developments to Improve the Efficiency of Automated MS/MS Spectra Interpretation. *Journal of Proteome Research*, 1:211–215, 2002.

[34] T. Sakurai, T. Matsuo, H.Matsuda, and I. Katakuse. A computer program to determine probable sequence of peptides from mass spectrometric data. *Biomed. Mass Sepctrom*, 11:396–399, 1984.

[35] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures.* Morgan Kaufmann, San Francisco, 2006.

[36] Nathan T. Slingerland and Alan Jay Smith. Multimedia extensions for general purpose microprocessors: A survey. *Microprocessors and Microsystems*, 29:225–246, 2005.

[37] A. Peter Snyder. *Interpreting Protein Mass Spectra: A Comprehensive Resource.* Oxford University Press, Oxford, New York, 2000.

[38] J. Alex Taylor and Richard S. Johnson. Implementation and uses of automated de novo peptide sequencing by tandem mass spectrometry. *Anal. Chem*, 73:2594–2604, 2001.

[39] Henry S. Warren. *Hacker's Delight.* Addison-Wesley Publishing Company, Boston, 2002.

[40] FangXiang Wu, Pierre Gagne, Arnaud Droit, and Guy G. Poirier. RT-PSM, a real-time program for peptide-spectrum matching with statistical significance. *Rapid Commum. Mass Sepctrom*, 20:1199–1208, 2006.

[41] Jian Zhang, Ian McQuillan, and FangXiang Wu. Speed Improvements of Peptide-Spectrum Matching Using SIMD Instructions. *IEEE BIBM2010 International workshop on Computational Proteomics, accepted*, 2010.

[42] Jingren Zhou and Kenneth A. Ross. Implementing database operations using SIMD instructions. *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 145–156, 2002.