Speeding up Scoring Module of Mass Spectrometry Based Protein Identification by GPUs

Li You

Abstract

Database searching is a main method for protein identification in shotgun proteomics, and till now most research effort is dedicated to improve its effectiveness. However, the efficiency of database searching is facing a serious challenge, due to the ever fast increasing of protein and peptide databases resulting from genome translations, enzymatic digestions, and post-translational modifications. On the other hand, as a general-purpose and high performance parallel hardware, Graphics Processing Units (GPUs) develop continuously and provide another promising platform for parallelizing database searching based protein identification to increase its efficiency.

In this paper, we propose to systematically research on speeding up database search engines by GPUs for protein identification. Considering the scoring module is the most time-consuming part, we mainly utilize GPUs to speed it up. We choose two popular scoring method: firstly, SDP based method, which is chosen by X!Tandem, reaches a speedup of thirty to one hundred; secondly, KSDP, which is adopted by pFind, achieves a speedup of five to ten.

1. Introduction

Protein identification is the basis of proteomics, the main character of which is high-throughput, with tandem mass spectrometry based shotgun approach as the technique of choice. Compared with other data analysis methods, database search engines have been the most stably and widely utilized, such as Mascot (Perkins et al., 1999), SEQUEST (Eng et al., 1994), pFind (Fu et al., 2004; Li et al., 2005; Wang et al., 2007), X !Tandem (Craig and Beavis, 2004), OMSSA (Geer et al., 2004), and Phenyx (Colinge et al., 2003). While most of research effort targets to improve effectiveness by the designing new scoring and validating algorithms, the efficiency of the database search engines are facing a serious challenge, due to the following reasons:

Firstly, the number of entries in protein sequence database is keeping increasing. Take IPI.Human for example, from v3.22 to v3.49, the count of the protein has increased nearly by 1/3 times. Moreover, together with the evolution of genome sequencing technologies, proteogenomic research wishes to adopt genome translated protein sequences to identify more protein. As an example, the EST database (Human.12.06) will be translated into 8,163,883 protein sequences, over 100 times larger than the human proteome IPI.Human.v3.49, which has only 74,017 protein sequences.(Li et al., 2010)

Secondly, increasing importance of considering semi- or non-specific digestion leads to 10 or 100 times more digested peptides respectively than specific digestion, as is shown in Table 1.

Table 1. The scale of peptide sequences under tryptic digestion

Database	Proteins	Peptides (fully specific)	Peptides (non-specific)
Yeast	6,717	741,476	120,464,808
IPI-Human	74,017	7,412,821	1,230,715,950
Swiss-Prot	398,181	34,764,218	5,605,491,572

peptide mass = 800 Da–6000 Da, peptide length = 4–100 amino acids, non-specifically digested peptide length = 4–50 amino acids, max missed cleavage sites = 2.

Table 2. The number of post-translationally modified peptides

Modification sites	Num. modified peptides
0	3,309,085
1	25,197,765
2	133,063,810
3	477,180,661
4	1,361,747,010
5	3,395,725,099
6	7,823,314,004
7	17,606,043,889
8	41,148,061,489
9	99,244,365,518

Note: database = IPI-Human V3.49, fully tryptic digestion, peptide mass = 800 Da–6000 Da, peptide length = 4–100 amino acids, max missed cleavage sites = 2. Ten modifications are specified: Oxidation (M), Phosphorylation (S, T, Y), Methylation (K, R), di-Methylation (K, R), tri-Methylation (K), and Acetylation (K).

Thirdly, post-translational modifications (PTMs) generate exponentially more modified peptides. Till now, over 500 types of PTMs exist in Unimod database (http://www.unimod.org). If we choose ten common variable PTMs and limit the number of modification sites in a peptide to no larger than five, the number of tryptic peptides of the human proteome will be increased over 1000 times, as is shown in Table 2. At the same time, the generation speed of the mass spectrometer increases steadily.

One of the direct results from the above four increase is the large scale number of scoring between peptide and spectrum, which is the most compute intensive and time consuming part in the whole flow of protein identification. Profiling analysis shows that scoring module takes more than 90% of total identification time in both pFind and X!Tandem. Thus, speeding up scoring module is a promising method to increase the efficiency of protein identification, which could be conducted by parallelizing the scoring function.

Recently, Graphics Processing Units (GPUs), which has become a general-purpose and high performance parallel hardware, develop continuously and provide another promising platform for parallelizing scoring function. GPUs are dedicated hardware for manipulating computer graphics. Due to the large demand for computing real-time and high-definition 3D graphics, the GPUs have evolved into highly parallel many-core processors. The advantages of computing power and memory bandwidth in GPUs have driven the development of general-purpose computing on GPUs (GPGPU).

To the best of our knowledge, no research has ever attempted to parallel database search engine by GPUs, which could work as a small cluster or a much higher performance node inside a cluster. Thus, in this paper, we choose two popular scoring methods: SDP based method, chosen by X!Tandem; KSDP, adopted by pFind, and conduct systematic research on parallelizing the scoring function by using a general-purpose parallel programming model, namely Compute Unified Device Architecture (CUDA). Our first contribution is firstly applying GPUs to speed up the protein identification. Our second contribution is the observation that the spectrum-peptide matching distribution is an important factor to be considered, based on which we design, implementation, and evaluation of two different strategies. For the spectrum which does not share matched peptides with other spectra, we mainly utilize the GPU on-chip registers and texture to minimize the memory access latency. For the spectra which share the same set of matched peptides, we design a novel and highly efficient algorithm that treats the scoring module as matrix multiplication, and then makes use of GPU on-chip shared memory together with on-chip registers. As a result, SDP gets a speedup of thirty to one hundred; KSDP achieves a speedup of five to ten.

The rest of this paper is organized as follows. Section II introduces some existing speedup methods, the GPU architecture and GPU application in bioinformatics. Section III presents our design of parallel scoring algorithm on GPUs. Section IV presents our experimental results, and Section V concludes the paper and presents some future work.

2. Background and Related work

We firstly introduce the background knowledge for scoring method, present the existing speedup method, and illustrate the basic architecture of GPU.

2.1. Spectrum and Fragment ions

A peptide is a string of amino acid residues joined together by peptide bonds. In the mass spectrometer, peptides derived from digested proteins are ionized. Peptide precursors of a specific mass-charge ratio (*m/z*) are selected and further fragmented by collision-induced dissociation (CID). Product ions are detected. The measured *m/z* and intensity of the product ions form finally the peaks in the tandem mass spectrum (MS/MS spectrum), as shown in Fig 1. By CID, three kinds of backbone cleavages on peptide bonds can produce six series of fragment ions, denoted by N-terminal *a*, *b* and *c* type fragments and C-terminal *x*, *y* and *z* type fragments, as shown in Fig.2.

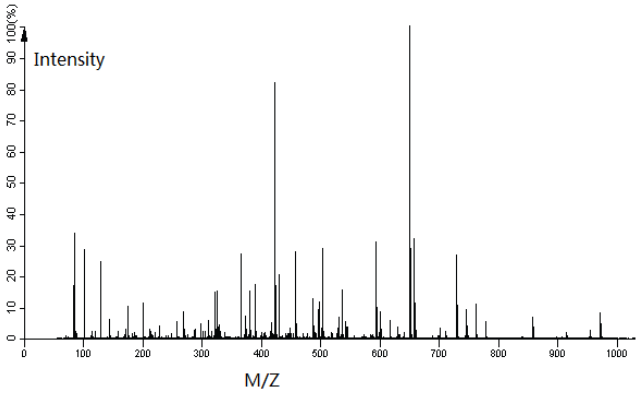

Fig. 1. An example of MS/MS Spectrum

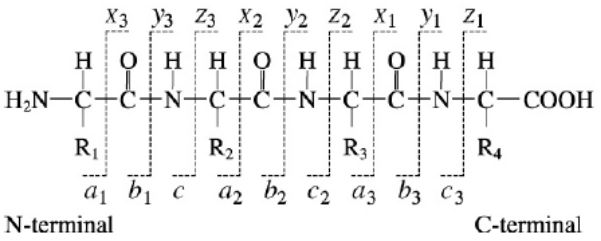

Fig. 2. Fragment ions from peptide bonds cleavage by CID

The scoring method computes the similarity between theoretical and experimental spectra, which could both be expressed as *N*-dimensional vectors, where *N* is the number of m/z values used. We use vector $\boldsymbol{c} = [c_1, c_2, ..., c_N]$ stand for the experimental spectrum and vector $\boldsymbol{t} = [t_1, t_2, ..., t_N]$ the theoretical one. c_i and t_i are binary values {0, 1} (or the intensity).

A very basic scoring method is spectral dot product (SDP). The SDP-based cosine value of the angle between spectral vectors was adopted as a similarity measure (Wan *et al.*, 2002; Tabb *et al.*, 2003). In current peptide-scoring algorithms, the SDP is often adopted directly or indirectly and plays an important role. The vector representation and the dot product were adopted explicitly in the Sonar. In SEQUEST, the cross-correlation of two spectra is actually the SDP, and the score Xcorr is the SDP minus the mean of a series of τ -displaced SDPs intended to reduce the stochastically high SDP. The shared peak count is the special case of the SDP where c_i and t_i are binary values.

While SDP is conceptually simple and effective in many cases, it ignores the correlative information among the dimensions of the spectral vector. One improving method is using kernel function to map the spectral vector space non-linearly into a high-dimensional space in which all the combinations of correlated fragments have their corresponding dimensions, which is KSDP's idea.

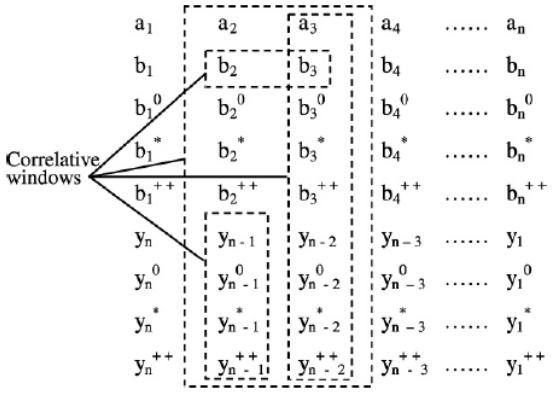

Fig. 3. Correlative matrix and correlative windows

KSDP is a kernel based SDP scoring method, which significantly increase the effectiveness of SDP. The kernel trick is to compute directly the dot product in the correlative space with a proper kernel without an explicit mapping from the spectral space to the correlative space. Considering different kind of fragment ions, all the fragments are arranged in correlative matrix, as shown in Fig.4. All predicted fragments are assumed to possess unique m/z values so that all non-zero dimensions in the theoretical spectral vector, $\boldsymbol{t}$, can be extracted and rearranged into the matrix $\boldsymbol{T}=(t_{pq})_{m*n}$, where m is the number of fragment types and n+1 is the residue number of peptide precursor. For example, $t_{2,3}$ corresponds to the fragment b_3 in Fig.4. The experimental spectral vector $\boldsymbol{c}$ could be organized in the same way.

2.2.Speeding up methods

There are several researches on improving the design of classical database search engines, for example, Edwards & Lippert considered the problem of redundant peptides and peptide-spectrum matching (Edwards and Lippert, 2002), Tang et al. adopted peptide and b/y ions indexes (Tang et al., 2005), Dutta & Chen utilized the nearest neighbor search to improve peptide-spectrum matching (Dutta and Chen, 2007), and Roos et al. made use of hardware cache to speed up identification (Roos et al., 2007).

There are also various researches, based on tag, to improve the efficiency of protein identification. One of the most significant method is the peptide sequence tag (Mann and Wilm, 1994), and followed by GutenTag (Tabb et al., 2003), MultiTag (Sunyaev et al., 2003), InsPecT (Tanner et al., 2005), and Spectral Dictionary (Kim et al., 2008). In fact, extracting peptide tag or tags from the tandem mass spectrum is a very complicated process, due to the spectra resolution and accuracy, charge states, peptides sequence length. Consequently this method is still not as commonly adopted as traditional database search engines.

Obviously paralleling database search engines could achieve a high efficiency. pFind, X!Tandem, Sequest, and Mascot all have parallel version. In fact, all the above work could further increase the efficiency by GPUs. For the single PC based search engine, GPUs could work as a small cluster. For parallel version, GPUs could sharply increase the computing power of each node.

2.3. The GPU architecture

We take NVIDIA GTX280 as an example to show the GPU architecture. GTX 280 has 30 Streaming Multiprocessors (SMs), and each SM has 8 Scalar Processors (SPs), resulting a total of 240 processor cores. The SMs have a Single-Instruction Multiple-Thread (SIMT) architecture: At any given clock cycle, each SP

executes the same instruction, but operates on different data. Each SM has four different types of on-chip memory, namely registers, shared memory, constant cache, and texture cache, as shown in Fig.1. Constant cache and texture cache are both read-only memories shared by all SPs, but with very limited size. Off-chip memories such as local memory and global memory have relatively long access latency, usually 400 to 600 clock cycles [10]. The properties of the different types of have been summarized in [10, 12]. In general, the scarce shared memory should be carefully utilized to amortize the global memory latency cost.

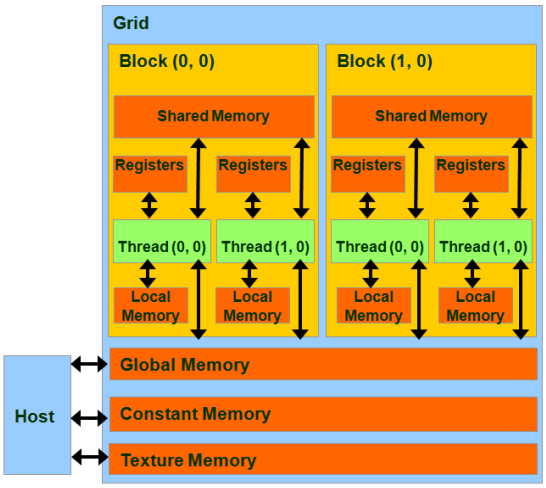

Fig. 4. Hardware architecture of the GPU

In CUDA model, the GPU is regarded as a coprocessor capable of executing a great number of threads in parallel. A single source program includes host codes running on CPU and also kernel codes running on the GPU. Compute-intensive and data-parallel kernel codes run on the GPU. The threads are organized into thread blocks, and each block of threads are executed concurrently on one SM. Threads in a thread block can share data through the shared memory and can perform barrier synchronization. But there is no synchronization mechanism for different thread blocks besides terminating the kernel. Another important concept in CUDA is *warp*, which is formed by 32 parallel threads and is the scheduling unit of each SM. When a warp stalls, the SM can schedule another warp to execute. A warp executes one instruction at a time, so full efficiency can only be achieved when all 32 threads in the warp have the same execution path. There are two consequences: first, if the threads in a warp have different execution paths due to conditional branch, the warp will serially execute each branch which increases the total time of instructions executed for this warp; second, if the number of threads in a block is not a multiple of warp size, the remaining instruction cycles will be wasted. Besides, when accessing the memory, *half-warp* executes as a group, which has 16 threads. If the half-warp threads access the coalesced data, the access operation will perform within one instruction cycle. Otherwise, the access operation will occupy up to 16 instruction cycles.

3. Design of parallel scoring module

Profiling analysis shows that scoring module takes more than 90% of total identification time with both pFind and X!Tandem. Thus, in this paper, we mainly parallel scoring module by GPUs, and choose two widely used scoring methods from two popular search engines.

3.1. Spectral dot product in X!Tandem

The tandem mass SDP between the experimental and theoretical spectra is defined as

$$SDP = <c, t> = \sum_{i=1}^{N} c_i t_i \qquad (1)$$

The algorithm of SDP is simple, as shown in Algorithm 1. Line 1~3, for each spectrum, find all the peptides whose precursor mass are in the spectrum's precursor mass window, assuming there are m peptides; line 4~5 compute the SDP score between the experimental and theoretical spectrum. The computation complexity is $O(|C|\lg(|T|)Nm)$. Adopting GPUs, we can assign each peptide to one thread, scoring with its matched peptide, as shown is Algorithm 2. Obviously, the computation complexity decreases to $O(\lg(|T|)Nm)$.

Algorithm 1: CPU-based SDP

```
// C: the set of experimental spectrum
// c:  experimental spectrum
// T: the set of experimental spectrum
// t: experimental spectrum
1. for each c in C
2.     for each t in T
3.       if c.mass > t.mass-tol && c.mass < t.mass+tol
4.           for i from 1 to N
5.               SDP_Score += c_i t_i
6.           end of for
7.       end of if
8.     end of for
9. end of for
```

Algorithm 2: GPU-based SDP

```
// C_i : the i_th spectrum in C
1.    i = threadId;
2.    for each t in T
3.      if C_i.mass > t.mass-tol && C_i.mass < t.mass+tol
4.          for j from 0 to N
5.              SDP_Score += C_ij t_i
6.          end of for
7.      end of if
8.    end of for
```

Another widely used scoring method XCorr could also

adopt the above parallel algorithm. XCorr is an important scoring part in SEQUEST, which a widely used protein identification software, but with a notorious searching speed. The process of XCorr is as equation 2 and 3. Obviously, calculating XCorr is like calculating SDP for 150 times.

$$R_\tau = \sum_{i=1}^{N} c_i t_{(i+\tau)} \quad (2)$$

$$XCorr = R_{(\tau=0)} - \frac{1}{149} \sum_{-75<k<75} R_{(\tau=k)} \quad (3)$$

3.2. KSDP in pFind

The process of KSDP is show as Equation 4 and Algorithm 5. Line 1~3 find the corresponding peptides for each spectrum; line 4~13 calculate the kernel function by traversing the whole correlative matrix. The computation complexity is O(|*C*|lg(|*T*|)*mn*). Using GPUs, we can also assign each spectrum to one thread, scoring with all its corresponding peptides, as shown in Algorithm 6. And the computation complexity is O(lg(|*T*|)*mn*).

$$K_{pep}(c,t) = \sum_{i=1}^{m}\sum_{j=1}^{n}[\sum_{k=j-l_1}^{j+l_2} (w_{|k-j|}(c_{ik}t_{ik})^{1/d})]^d \quad (4)$$

Algorithm 5: CPU-based KSDP

//*l*: the size of correlative window
//l_1: $\lfloor (l-1)/2 \rfloor$, l_2: $\lceil (l-1)/2 \rceil$
//*win*: temp kernel value
//*d*: a parameter controls the degree of correlation

```
1.  for each c in C
2.    for each t in T
3.      if c.mass > t.mass-tol && c.mass < t.mass+tol
4.        K_ct = 0;
5.        for (i=1; i≤m; ++i)
6.          win_{i,1} = 0;
7.          for(j=1;j≤l2; ++j)
8.            win_{i,1} += (c_ij, t_ij)^{1/d};
9.          end of for
10.         K_ct += win_{i,1}^d;
11.         for (j=2; j≤n; ++j)
12.           win_{i,j}=win_{i,j-1}+(c_{i,j+l2},t_{i,j+l2})^{1/d}-(c_{i, j-l1-1}, t_{i, j-l1-1})^{1/d};
13.           K_ct += win_{i,j}^d;
14.         end of for
15.       end of for
16.     end of if
17.   end of for
18. end of for
```

Algorithm 6: GPU-based KSDP

//

```
1.  i = threadID;
2.    for each t in T
3.      if C_i.mass > t.mass-tol && C_i.mass < t.mass+tol
4.        K_ct = 0;
5.        for (i=1; i≤m; ++i)
6.          win_{i,1} = 0;
7.          for(j=1;j≤l2; ++j)
8.            win_{i,1} += (c_ij, t_ij)^{1/d};
9.          end of for
10.         K_ct += win_{i,1}^d;
11.         for (j=2; j≤n; ++j)
12.           win_{i,j}=win_{i,j-1}+(c_{i,j+l2},t_{i,j+l2})^{1/d}-(c_{i, j-l1-1}, t_{i, j-l1-1})^{1/d};
13.           K_ct += win_{i,j}^d;
14.         end of for
15.       end of for
16.     end of if
17.   end of for
18. end of for
```

4. Experiment

We have implemented both CPU- and GPU based scoring algorithms using CUDA version 2.3. Our experiments were conducted on a PC with an NVIDIA GTX280 GPU and an Intel(R) Core(TM) i5 CPU. GTX 280 has 30 SIMD multi-processors, and each one contains eight processors and performs at 1.29 GHz. The memory of the GPU is 1GB with the peak bandwidth of 141.7 GB/sec. The CPU has four cores running at 2.67 GHz. The main memory is 8 GB with the peak bandwidth of 5.6 GB/sec. We calculate the time of the application after the file I/O, in order to show the speedup effect more clearly.

We compare the speed of CPU- and GPU based scoring algorithms, varying the number of the spectrum and peptide. To show the number of scoring and the time consumption more clearly, we let each spectrum score with a specific number of peptide.

As shown in Table 3, the speedup of SDP by GPUs is very favorable, from thirty to more than one hundred, mainly resulting from the distribution of each spectrum to each thread. Besides, owing to the simple calculation process of SDP, most of the work could be conducted on on-chip register without reading latency. We can also observe that the increase of the number of spectrum does not increase the time consumption, which means this simple parallel algorithm works well.

As is shown in Table 4, KSDP achieves a speedup of two to eight, which is not very favorable right now. The speedup mainly comes from the simple multi-thread without any optimization concerning memory usage.

Table 3. The speedup effect of SDP, in second.

Pep	*Spec*	*On CPU*	*On GPU*
1024	1024	3.74	0.11
	2048	7.49	0.14
	4096	14.97	0.16
2048	1024	7.51	0.14
	2048	15.02	0.18
	4096	30.01	0.22
4096	1024	15.07	0.18
	2048	30.26	0.23
	4096	60.28	0.31

Table 4. The speedup effect of KSDP, in second.

Pep	*Spec*	*On CPU*	*On GPU*
1024	1024	1.85	0.78
	2048	3.74	0.81
	4096	7.58	0.97
2048	1024	3.86	1.52
	2048	7.78	1.57
	4096	15.56	1.96
4096	1024	8.01	3.01
	2048	16.02	3.09
	4096	32.01	3.98

5. Future work

As talked in section 4, KSDP has not got a very high speed-up effect, since we only use global memory. In the following work, we would:

Firstly, we would put the spectrum on texture, since each spectrum would score with multi peptides, so using texture with cache mechanism would decrease the reading latency.

Secondly, when the set of peptides grows, letting one thread deals with one spectrum is not surely a good idea. We need to make a new strategy: letting one thread dealing with one spectrum and a limited set of peptides, when the peptides grow, we will use multi thread to deal with one spectrum.

Thirdly, we find out that some spectrum with near precursor mass would score with the same set of peptides, which makes it possible to adopt shared memory to further optimize the algorithm.

References

[1] Perkins, D. N.; Pappin, D. J.; Creasy, D. M.; Cottrell, J. S., Probability-based protein identification by searching sequence databases using mass spectrometry data. *Electrophoresis* **1999,** 20, (18), 3551-67.

[2] Eng, J., An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. *Journal of the American Society for Mass Spectrometry* **1994,** 5, (11), 976-989.

[3] Fu, Y.; Yang, Q.; Sun, R.; Li, D.; Zeng, R.; Ling, C. X.; Gao, W., Exploiting the kernel trick to correlate fragment ions for peptide identification via tandem mass spectrometry. *Bioinformatics* **2004,** 20, (12), 1948-54.

[4] Gronert, S.; Li, K. H.; Horiuchi, M., Manipulating the fragmentation patterns of phosphopeptides via gas-phase boron derivatization: determining phosphorylation sites in peptides with multiple serines. *J Am Soc Mass Spectrom* **2005,** 16, (12), 1905-14.

[5] Gao, Y.; Wang, Y., A method to determine the ionization efficiency change of peptides caused by phosphorylation. *J Am Soc Mass Spectrom* **2007,** 18, (11), 1973-6.

[6] Craig, R.; Beavis, R. C., TANDEM: matching proteins with tandem mass spectra. *Bioinformatics* **2004,** 20, (9), 1466-7.

[7] Geer, L. Y.; Markey, S. P.; Kowalak, J. A.; Wagner, L.; Xu, M.; Maynard, D. M.; Yang, X.; Shi, W.; Bryant, S. H., Open mass spectrometry search algorithm. *J Proteome Res* **2004,** 3, (5), 958-64.

[8] Colinge, J.; Masselot, A.; Giron, M.; Dessingy, T.; Magnin, J., OLAV: towards high-throughput tandem mass spectrometry data identification. *Proteomics* **2003,** 3, (8), 1454-63.

[9] Mann, M.; Wilm, M., Error-tolerant identification of peptides in sequence databases by peptide sequence tags. *Anal Chem* **1994,** 66, (24), 4390-9.

[10] Tabb, D. L.; Saraf, A.; Yates, J. R., 3rd, GutenTag: high-throughput sequence tagging via an empirically derived fragmentation model. *Anal Chem* **2003,** 75, (23), 6415-21.

[11] Sunyaev, S.; Liska, A. J.; Golod, A.; Shevchenko, A., MultiTag: multiple error-tolerant sequence tag search for the sequence-similarity identification of proteins by mass spectrometry. *Anal Chem* **2003,** 75, (6), 1307-15.

[12] Tanner, S.; Shu, H.; Frank, A.; Wang, L. C.; Zandi, E.; Mumby, M.; Pevzner, P. A.; Bafna, V., InsPecT: identification of posttranslationally modified peptides from tandem mass spectra. *Anal Chem* **2005,** 77, (14), 4626-39.

[13] Datta, R.; Bern, M., Spectrum fusion: using multiple mass spectra for de novo Peptide sequencing. *J Comput Biol* **2009,** 16, (8), 1169-82.

[14] Kim, S.; Gupta, N.; Bandeira, N.; Pevzner, P. A., Spectral dictionaries: Integrating de novo peptide sequencing with database search of tandem mass spectra. *Mol Cell Proteomics* **2008**.

[15] Bafna, V.; Edwards, N. In *On de novo interpretation of tandem mass spectra for peptide identification*, RECOMB '03: Proceedings of the seventh annual international conference on Research in computational molecular biology, 2003; ACM Press: 2003; pp 9-18.

[16] Shilov, I. V.; Seymour, S. L.; Patel, A. A.; Loboda, A.; Tang, W. H.; Keating, S. P.; Hunter, C. L.; Nuwaysir, L. M.; Schaeffer, D. A., The Paragon Algorithm, a next generation search engine that uses sequence temperature values and feature probabilities to identify peptides from tandem mass spectra. *Mol Cell Proteomics* **2007,** 6, (9), 1638-55.

[17] Yen, C. Y.; Russell, S.; Mendoza, A. M.; Meyer-Arendt, K.; Sun, S.; Cios, K. J.; Ahn, N. G.; Resing, K. A., Improving sensitivity in shotgun proteomics using a peptide-centric database with reduced complexity: protease cleavage and SCX elution rules from data mining of MS/MS spectra. *Anal Chem* **2006,** 78, (4), 1071-84.

[18] Dutta, D.; Chen, T., Speeding up tandem mass spectrometry database search: metric embeddings and fast near neighbor search. *Bioinformatics* **2007,** 23, (5), 612-8.

[19] Roos, F. F.; Jacob, R.; Grossmann, J.; Fischer, B.; Buhmann, J. M.; Gruissem, W.; Baginsky, S.; Widmayer, P., PepSplice: cache-efficient search algorithms for comprehensive identification of tandem mass spectra. *Bioinformatics* **2007,** 23, (22), 3016-23.

[20] Park, C. Y.; Klammer, A. A.; Kall, L.; MacCoss, M. J.; Noble, W. S., Rapid and accurate peptide identification from tandem mass spectra. *J Proteome Res* **2008,** 7, (7), 3022-7.