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

You Li
Department of Computer Science
Hong Kong Baptist University
Kowloon Tong, Hong Kong
youli@comp.hkbu.edu.hk

Xiaowen Chu
Department of Computer Science
Hong Kong Baptist University
Kowloon Tong, Hong Kong
chxw@comp.hkbu.edu.hk

***Abstract*—Database searching is a main method for protein identification in shotgun proteomics, and many research efforts are dedicated to improving its effectiveness. However, the efficiency of database searching is facing a serious challenge, due to the ever fast growth of protein and peptide databases resulted from genome translations, enzymatic digestions, and post-translational modifications (PTMs). 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. It becomes very important to study how to speed up database search engines by GPUs for protein identification. In this paper, we mainly utilize GPUs to accelerate the scoring module, which is the most time-consuming component. Specifically, we study two popular scoring method: spectral dot product based method, which is used by X!Tandem, and kernel spectral dot product, which is used by pFind.**

Keywords-protein identification, spectral dot product, GPU computing

I. INTRODUCTION

High-throughput tandem mass spectrometry (referred to as MS/MS hereafter) based protein identification is a powerful method in proteomics. It enables large scale analysis of the protein sequence and PTMs with high sensitivity, accuracy and throughput [1-5]. Among all the MS/MS data analysis methods, protein database search based approaches have been widely used, such as Mascot [6], SEQUEST [7], pFind [8, 9, 10], X!Tandem [11], OMSSA [12], and Phenyx [13].

While most of research effort targets to improve the effectiveness by 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 [14]. Secondly, increasing importance of considering semi- or non-specific digestion leads to 10 or 100 times more digested peptides respectively than specific digestion. Thirdly, post-translational modifications (PTMs) generate exponentially more modified peptides. Till now, over 900 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. At the same time, the generation speed of the mass spectrometer increases steadily.

Since all the algorithms in database search engine calculate the similarity between the experiment MS/MS and the theoretical candidate MS/MS, generated from the protein (peptide) database, one of the direct results from the above increase is the large scale number of scoring calculation, which is the most computing 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.

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). We take NVIDIA GTX580 as an example to show a typical GPU architecture. GTX 580 has 64 Streaming Multiprocessors (SMs), and each SM has 8 Scalar Processors (SPs), resulting in a total of 512 processor cores. The SMs have a Single-Instruction Multiple-Thread (SIMT) mode: at any given clock cycle, each SP executes the same instruction, but operates on different data.

Considering the independence of each score in protein identification database search engine, it is reasonable to parallelize the scoring module in SIMT architecture on GPU or GPU cluster. To the best of our knowledge, no research has ever attempted to utilize GPU in this field. Thus, in this paper, we choose two popular scoring methods: spectral dot product (*SDP*), used by X!Tandem and SEQUEST, and kernel spectral dot product (*KSDP*) adopted by pFind. We 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 GPU cluster to speed up the database search protein identification. Our second contribution is carefully utilizing the GPU memory architecture and computation power to get a high efficiency *SDP* algorithm. Our third contribution is supplying a high performance *SDP*-based scoring module, which could be adopted by all the search engine easily. As a result, *SDP* gets a speedup of thirty to one hundred.

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

II. BACKGROUND AND RELATED WORK

We firstly introduce the background knowledge for MS/MS based protein identification, then present the existing speedup method.

A. Tandem Mass spectrum based protein identification

The database search process of tandem mass spectrum based protein sequence identification is shown in Fig.1, which includes three steps.

(1) Experimental MS/MS generation: digest mixed proteins into mixed peptides by enzyme; input the peptides to the liquid chromatography–tandem mass spectrometry (LC-MS/MS), where they will be ionized and further fragmented, e.g., by collision-induced dissociation (CID); output the MS/MS data, containing the measured *m/z* and intensity of the fragments, represented by the ***peaks***, as shown in Fig 2. 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.3.

(2) Theoretical MS/MS generation: digest the protein sequence in the database into peptides by simulating the process of LC-MS/MS to obtain the theoretical MS/MS of the peptide. However, the physical and chemical factors in the experimental MS/MS generation process, are very complicated and hard to learn. Thus, the intensity information does not appear in the theoretical spectrum, as shown in Fig. 4. The data may also be further preprocessed before the identification.

(3) Similarity scoring: for each experimental MS/MS, search all the theoretical MS/MS within a preset mass tolerance and get candidates; calculate the similarity between the experimental MS/MS and candidates, and keep the top one(s) as the result; merge the identified MS/MS, which stands for the peptide, to the final protein sequence.

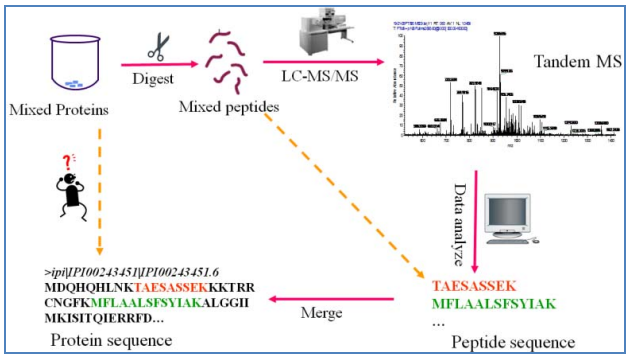

Figure 1. Protein identification flow

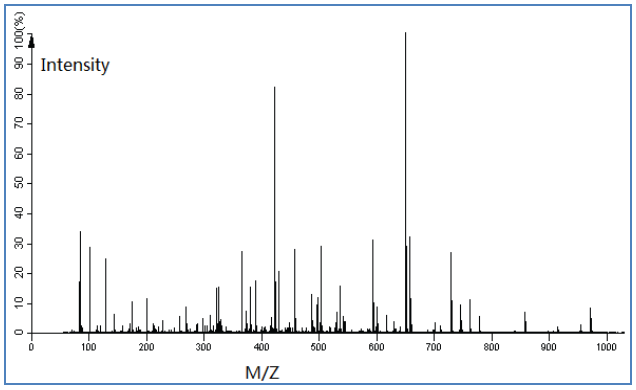

Figure 2. An example of MS/MS Spectrum

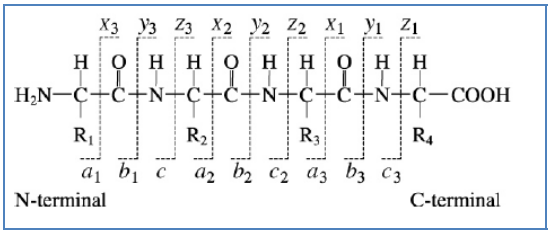

Figure 3. Fragment ions from peptide bonds cleavage by CID

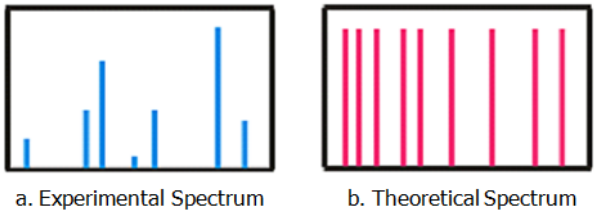

Figure 4. Difference between experimental and theoretical spectrum.

B. Speeding up methods

As the computational demand is increasing significantly, the research on the efficiency keeps developing, including the following respects:

Tag: The peptide sequence tag [15] is one of the most significant method of protein identification, and followed by GutenTag [16], MultiTag [17], InsPecT [18], and Spectral Dictionary[19].

Indexing techniques: Li & Chi systematically introduced the effect of indexing techniques and design a inverted index strategy for protein identification [14]. Edwards & Lippert considered the problem of redundant peptides and peptide-spectrum matching and used suffix tree index [20], Tang et al. adopted peptide and b/y ions indexes [21], Dutta & Chen utilized the nearest neighbor search to improve peptide-spectrum matching [22].

Parallel technology: most of the popular peptide and protein search engines have their own parallel versions: SEQUEST adopts parallel virtual machine (PVM) to build its cluster system [23], while Mascot and Phenyx use message passing interface (MPI). X!Tandem has two parallel strategies [24, 25]. Besides, these systems have been integrated into higher-level application frameworks, such as web service or grid [26], even cloud computing. Halligan has migrated X!Tandem and OMSSA to the Amazon cloud computing platform [27].

Hardware: Roos et al. made use of hardware cache to speed up identification [28]; Bogdan I et al. and Dandass YS utilized FPGA [29]; Hussong R used GPU to speed up the feature selection step [30]. Baumgardner A implemented a spectrum library search algorithm on GPU [31].

The recent advances of computing power in GPUs have driven the development of general-purpose computing on GPUs (GPGPU), which has been used for accelerating a wide range of applications [32-35]. To the best of our knowledge, this is the first work of accelerating scoring module in database search based protein identification by GPUs.

III. DESIGN OF PARALLEL SCORING MODULE

The scoring module calculates the similarity between the 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 $\boldsymbol{c} = [c_1, c_2, ..., c_N]$ to stand for the experimental spectrum and vector $\boldsymbol{t} = [t_1, t_2, ..., t_N]$ for the theoretical one, where N is the number of different *m/z*, c_i and t_i could be binary values {0, 1} or intensity, at the i-th *m/z* value in the MS/MS spectrum.

Profiling analysis shows that scoring module takes more than 90% of total identification time in both pFind and X!Tandem. Thus, we target at paralyzing the scoring module by GPUs, and choose two widely used scoring methods *SDP* and *KSDP*.

A. SDP in X!Tandem

Spectral dot product (*SDP*) is a very basic scoring method, as well as the SDP-based cosine value of the angle between the experimental and theoretical spectrum vectors, which is adopted directly or indirectly in SEQUEST, X!Tandem, and Sonar. The tandem mass *SDP* between the experimental and theoretical spectra is defined as

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

Each spectrum has a precursor mass, which could be considered as the mass of the ironed peptide. Obviously, only the theoretical and experimental spectrum, whose precursor mass distance is within a predefined tolerance need to be scored. Given $|C|$ experimental spectra and $|T|$ theoretical spectrum, the *SDP* scoring process is: for each theoretical spectrum (peptide), binary search all the experimental spectra whose precursor mass are in the peptide's precursor mass window and get C', assuming there are S spectra; then compute the *SDP* score between the peptide and the S spectra. The computation complexity is $O(|T|\lg(|C|)+|T||C'|NS)$, as shown in Algorithm 1. Line 1~2, for each peptide, binary search all the spectra whose precursor mass are in the peptide's precursor mass window; line 3~6 compute the *SDP* score between the peptide and the experimental spectrum. We search spectra for each peptide, instead of searching peptides for each spectra, in order to efficiently deal with Post Translational Modifications [14].

Algorithm 1: CPU-based *SDP*

```
// C: the set of experimental spectrum
// c: experimental spectrum
// T: the set of theoretical spectrum, sorted by mass
// t: theoretical spectrum
1. for each t in T
2.     binary search C , got C'
3.     for each c in C'
4.         for i from 1 to N
5.             SDP_Score += c_i t_i
6.         end of for
7.     end of for
8. end of for
```

Adopting GPUs, the first straight forward method is assigning each peptide to one thread, scoring with its matched experimental spectrum, and the computation complexity decreases to $O(\lg(|C|)+|C'|NS)$, as shown is Algorithm 2. In practice, the number of peptides is much more than the number of threads in GPU. Thus, the speedup effect is not directly the same with the effect in computation complexity.

Algorithm 2: GPU-based *SDP*

```
// T_i : the i_th spectrum in T
1.  i = threadId;
2.  binary search C, got C' ;
3.  for each c in C'
4.      for j from 0 to N
5.          SDP_Score += C_ij t_i
6.      end of for
7.  end of for
```

Algorithm 3: GPU-based *SDP*

```
//sdata, g_idata: data address on shared and global memory;
//blockIdx, threadIdx: the index of block in grid, and
//thread in block, with x and y dimension;
//gridDim, blockDim: the dimension of grid and block;
//gridSize, blockSize: the thread No. of grid and block
//i, tid: the No. of data and thread;
//n: the size of data;
// g_oD: the address of the result;
1.  extern __ shared __ int sdata[];
2.  unsigned int i = blockIdx.x*(blockDim.x*2) +
    threadIdx.x;
3.  unsigned int gridSize = blockSize*2*gridDim.x;
4.  sdata[tid] = 0;
5.  while (i<n)
6.  {
7.     sdata[tid] = g_idata[i] + g_idata[i+blockSize];
8.     i += gridSize;
9.  }
10. if (blockSize >= 512)
11.    if (tid < 256) {sdata[tid] += sdata[tid +256];}
12.    __synchthreads();
13. if (blockSize >= 128)
14.    if (tid < 128) { sdata[tid] += sdata[tid +128];}
15.    __synchthreads();
16. if (blockSize >= 64)
17.    if (tid < 64) { sdata[tid] += sdata[tid +64];}
18.    __synchthreads();
19. if (tid <32)
20.   { sdata[tid] += sdata[tid +32];
21.     sdata[tid] += sdata[tid +16];
22.     sdata[tid] += sdata[tid +8];
23.     sdata[tid] += sdata[tid +4];
24.     sdata[tid] += sdata[tid +2];
25.     sdata[tid] += sdata[tid +1];}
26. if(0 == tid)
27.       g_oD = sdata[0];
```

Normally, N, the dimension and the number of ion peak of the spectrum, is around 200, to limit the computing operation. However, as the demand of effectiveness increases, it is worth to adopt more effect ion peak in the spectrum. Thus, we also optimize the algorithm to deal with the large N, focusing on the computing *SDP* of two spectra, which is line 4-5 in algorithm 2. We divide this step into two parts: first, multiply corresponding elements in two spectra; second, add all the multiplication results. The optimization is mainly on the second part, as shown in algorithm 3: since the GPU does not support global synchronization, the algorithm calls the kernel function multiply times, each of which deals with the elements inside one block. line 1-9 load the data from global memory to the shared memory, as well as performing the first step of adding; line 10-18 add two elements, whose distances are from 512 to 64; line 19-25 deal with the last 32 elements; line 26-27 send the result back. The core idea in the algorithm is avoiding reading bank conflict on GPU and unroll the *for* operations.

Another scoring method is *XCorr*, used in SEQUEST. The process of *XCorr* is as equation 2 and 3. Obviously, calculating *XCorr* contains two parts: firstly, calculate *SDP* for 150 times, where 75 is chosen in SEQUEST according to their experience; then compute the average and minus the value where τ equals to zero. The first step could apply the strategy from *SDP* on GPU, and probably get a better speedup effect, resulting from more computing operations with the same reading operations. The second step could be conducted on CPU, since τ only has less than two hundred of values. If τ grows to thousands, we could also use parallel reduction in this step as Algorithm 3.

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

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

B. KSDP in pFind

While *SDP* is conceptually simple and effective in many cases, it ignores the correlative information among the dimensions of the spectral vector. One improved method used by *KSDP* 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.

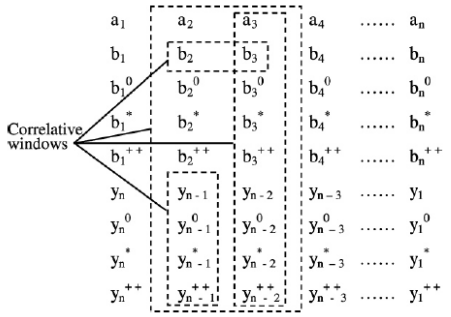

Figure 5. Correlative matrix and correlative windows

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.5. 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.5. The experimental spectral vector $\boldsymbol{c}$ could be organized in the same way.

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

The process of *KSDP* is as follows: firstly, find the candidate experimental spectra for peptide; secondly, for each pair of spectra, compute the result correlative matrix, which tells whether the element in the correlative matrix of experimental spectrum exists in the correlative matrix of theoretical spectra. Then traverse the result matrix and

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

Algorithm 5: CPU-based *KSDP*

```
//l: the size of correlative window
//l1: ⌊(l−1)/2⌋, l2: ⌈(l−1)/2⌉
//win: temp kernel value, revised by a experience weight in
//      practice, which is the W in equation 4 .
//d: a parameter controls the degree of correlation
1.  for each t in T
2.      binary search C , got C'
3.      for each c in C'
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 for
17. end of for
```

Algorithm 6: GPU-based *KSDP*

```
// threadID: the id of the current thread
1.  i = threadID;
2.      binary search C , got C'
3.      for each c in C'
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 for
```

IV. 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 highlight the speedup effect more clearly.

TABLE I. 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 II. 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

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 1, 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 shown in Table 2, *KSDP* achieves a speedup of two to eight, which is not very favorable right now. But with the increase of *Spec*, the speedup also increases. Currently the speedup mainly comes from the simple multi-thread without any optimization concerning memory usage. The latest GPUs such as GTX 680 has much more computational power and higher memory bandwidth than GTX 280, so we believe that GPU is still a more promising solution than CPU even for *KSDP*. We will leave it as a future work to further optimize the performance of *KSDP*.

V. CONCLUSION

In this paper, we have presented a novel GPU based scoring method, designed and implemented *SDP* and *KSPD* on GPU platform, and achieve around 100 times speedup on *SDP* and around 8 times speedup on *KSDP*. As a future work, we plan to further improve the performance of *KSDP*, and also to extend our algorithms to a heterogeneous CPU/GPU cluster to support even larger database search.

ACKNOWLEDGEMENT

This work is supported by an FRG grant FRG2/10-11/099 from Hong Kong Baptist University.

REFERENCES

[1] Wilkins M.R., Gasteiger E., Gooley A.A., Herbert B.R., Molloy M.P., Binz P.A., Ou K., Sanchez J.C., Bairoch A., Williams K.L., High-throughput mass spectrometric discovery of protein post-translational modifications. *J Mol Biol* **1999**, 289(3):645-657.

[2] Mann M., Jensen O.N., Proteomic analysis of post-translational modifications. *Nat Biotechnol.* **2003**, 21(3):255-261.

[3] Witze E.S., Old W.M., Resing K.A., Ahn N.G., Mapping protein post-translational modifications with mass spectrometry. *Nat Methods* **2007**, 4(10):798-806.

[4] Uy R., Wold F., Posttranslational covalent modification of proteins. *Science* **1977**, 198(4320):890-896.

[5] Walsh C.T., Posttranslational Modification of Proteins: Expanding Nature's Inventory. *Englewood (Colorado): Roberts & Company Publishers*, **2005**.

[6] 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.

[7] 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.

[8] 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.

[9] Li D., Fu Y., Sun R., Ling C.X., Wei Y., Zhou H., Zeng R., Yang Q., He S., Gao W., pFind: a novel database-searching software system for automated peptide and protein identification via tandem mass spectrometry. *Bioinformatics* **2005**, 21: 3049.

[10] Wang L.H., Li D.Q., Fu Y., Wang H.P., Zhang J.F., Yuan Z.F., Sun R.X., Zeng R., He SM, Gao W., pFind 2.0: a software package for peptide and protein identification via tandem mass spectrometry. Rapid Commun. Mass Spectrom. **2007**, 21: 2985.

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

[12] 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.

[13] 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.

[14] Li Y., Chi H., Wang L.H., Wang H.P., Fu Y., Yuan Z.F., Li S.J., Liu YS, Sun RX, Zeng R, He SM. Speeding up Protein Identification by indexing technology. *Rapid Commun. Mass Spectrom*. **2010**, 24: 807.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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**.

[20] Edwards N, Lippert R., Generating peptide candidates from amino-acid sequence databases for protein identification via mass spectrometry. *WABI 2002 Algorithms in Bioinformatics: Second International Workshop*, **2002**, 68.

[21] Tang W.H., Halpern B.R., Shilov I.V., Seymour S.L., Keating S.P., Loboda A., Patel A.A., Schaeffer D.A., Nuwaysir L.M., Discovering known and unanticipated protein modifications using MS/MS database searching. *Anal. Chem*. **2005**, 77: 3931.

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

[23] Sadygov R.G., Eng J., Durr E., Saraf A., McDonald H., MacCoss M.J., Yates J.R. III. J. A, correlation algorithm for the automated quantitative analysis of shotgun proteomics data. *Proteome Res*. **2002**, 1: 211.

[24] Bjornson R.D., Carriero N.J., Colangelo C., Shifman M., Cheung K.H., Miller P.L., Williams K.. X!!Tandem, an improved method for running X!tandem in parallel on collections of commodity computers. *J. Proteome Res.*, **2008**, 7: 293.

[25] Duncan D.T., Craig R., Link A.J.., Parallel Tandem: a program for parallel processing of tandem mass spectra using PVM or MPI and X!Tandem. *J. Proteome Res*., **2005**, 4: 1842.

[26] Dominic B., David Sigfredo A., MPI Framework for Parallel Searching in Large Biological Databases, *J. Parallel Distrib. Comput.* **2006**, 66: 1503.

[27] Halligan B.D., Geiger J.F., Vallejos A.K., Greene A.S., Twigger S.N., Low cost, scalable proteomics data analysis using Amazon's cloud computing services and open source search algorithms, *J. Proteome Res.* **2009**, 8: 3148.

[28] 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.

[29] Bogdan I., Coca D., Rivers J., Beynon R.J., Hardware acceleration of processing of mass spectrometric data for proteomics. *Bioinformatics* **2007**, 23: 724.

[30] Hussong R., Gregorius B., Tholey A., Hildebrandt A., Highly accelerated feature detection in proteomics data sets using modern graphics processing units. *Bioinformatics* **2009**, 25: 1937.

[31] Lydia Ashleigh Baumgardner, Avinash Kumar Shanmugam, Henry Lam, Jimmy K. Eng, and Daniel B. Martin, Fast Parallel Tandem Mass Spectral Library Searching Using GPU Hardware Acceleration, *J. Proteome Res*. **2011**; 10: 2882.

[32] Chu X.-W., Zhao K., Wang M., Massively parallel network coding on GPUs. In *Proceedings of IEEE IPCCC 2008*, December **2008**.

[33] Chu X.-W., Zhao K., Wang M., Practical Random Linear Network Coding on GPUs. In *Proceedings of IFIP Networking 2009*, Archen, Germany, May **2009**.

[34] Li Y., Zhao K., Chu X.-W., Liu J. -M., Speeding up k-means by GPUs, *the 10th. CIT*, **2010**.

[35] Wang L.H., Wang W.P., Chi H., Wu YJ., Li Y., Fu Y., Zhou C., Sun RX., Wang HP., Liu C., Yuan ZF., Xiu LY., He, SM., An efficient parallelization of phosphorylated peptide and protein identification, *RCM*, **2010**, 24:1791-1798.