2010 IEEE International Conference on Bioinformatics and Biomedicine Workshops

On Accelerating Iterative Algorithms with CUDA: A Case Study on Conditional Random Fields Training Algorithm for Biological Sequence Alignment

Zhihui Du[1+] ,Zhaoming Yin[2] , Wenjie Liu[1] and David Bader[3]
[1]*Tsinghua National Laboratory for Information Science and Technology*
Department of Computer Science and Technology, Tsinghua University, 100084, Beijing, China
+Corresponding Author's Email: duzh@tsinghua.edu.cn
[2]*School of Software and Microelectronics, Peking University, 100871, China.*
Email： zhaoming_leon@pku.edu.cn
[3]*College of Computing, Georgia Institute of Technology, Atlanta, GA, 30332, USA.*

Abstract **The accuracy of Conditional Random Fields (CRF) is achieved at the cost of huge amount of computation to train model. In this paper we designed the parallelized algorithm for the Gradient Ascent based CRF training methods for biological sequence alignment. Our contribution is mainly on two aspects: 1) We flexibly parallelized the different iterative computation patterns, and the according optimization methods are presented. 2) As for the Gibbs Sampling based training method, we designed a way to automatically predict the iteration round, so that the parallel algorithm could be run in a more efficient manner. In the experiment, these parallel algorithms achieved valuable accelerations comparing to the serial version.**

Keywords Conditional Random Fields; Biological Sequence Alignment; GPGPU

I. INTRODUCTION

With the rapid growth of biological databases, simply adding new training resources will reveal their limitation, and better algorithms with more complicated model which can include more features are needed. And Conditional Random Fields (CRF) introduced by Lafferty et al [1], is one of them. This method has already been successfully employed in many fields such as Nature Language Processing, Information Retrieval, and Bioinformatics [2, 3, 4, 5]. CRF is a kind of discriminative model, the training algorithms for this kind of model are mainly based on the gradient of the conditional likelihood function, or on a related idea [14].

Currently, the parallelization methods of Conditional Random Fields are mainly the coarse-grained method, such as the FlexCRF [8] and ContraAlign [9]. They are generally about partitioning sub-tasks (such as a single training sample) to different computation nodes. Since the operations of the sub-tasks also consist of loops and iterations, they still have a great potential for the fine-grained acceleration, and the GPU programming is one of the possible way to achieve the fine-grained acceleration.

We provide the design, implementation, and experimental study, of the parallel CRF iterative training algorithm on GPU card. More specifically, the algorithm is aimed at biological sequence alignment. And we implement the parallel algorithm for both Collins Perceptron based algorithm [27] and Gibbs Sampling based algorithm [14], because of their different iterative patterns.

The rest of this paper is organized as follows: in section II, we introduce the basic idea of Conditional Random Fields, Biological Sequence Alignment and GPU CUDA programming language. In section III, we describe the design of the parallelized iterative CRF training algorithm. In section IV, we proposed some of the problems and our optimization ideas. The experiments are presented in section V, Conclusions and future works are discussed in section VI.

II. BACKGROUND AND RELATED WORK

Conditional Random Fields (CRF) introduced by Lafferty et al is a kind of Discriminative Model [12], different from the generative models such as Hidden Markov Model [13], it has many advantages such as: supporting of multiple feature selection, and the relaxation of strong independent assumption. In addition, as a kind of undirected graphical model, it conquers the label bias problem [1] which brings inaccuracy to other directed graph models such as Maximal Entropy Hidden Markov Model [26]. Biological Sequence Alignment (BSA) is the task of comparing DNA or RNA sequences and align them with some objective functions [28]. There is a pair-wise CRF based method by Chun Do et al to do BSA [9].

Liu et al. [15] explore the power of GPU using the OpenGL graphic language. This is the first GPU implementation of biological sequence alignment based algorithms. Munekawa et al. [16] and Cschatz [17] propose the implementation of Smith-Waterman on GPU using CUDA. They discuss in detail of how to arrange the threads and how to make the memory access faster.

The parallel CRF based training method also implement some of the ideas in Hidden Markov Model based BSA, such as Viterbi algorithm. ClawHMMER [18, 19] is an HMM-based sequence alignment application on GPUs. We parallelized the HMM based BSA using CUDA [20], and proposed a tile based way to cope with long sequences more efficiently. We also used the Viterbi algorithm in this paper.

Currently the parallelization of CRF is mainly on coarse-grained method using MPI, such as FlexCRF [8] and ContraAlign [9], their work do not conflict with our fine-grained method. Since the training of CRF occupies most of the workload in the BSA, we mainly concern on the training of CRF for BSA.

III. TRAINING ALGORITHMS

A. BSA and CRF training

The sequence alignment is, for example, there are two sequences, template sequence: AACT, target sequence: AAACT, and the alignment is: The problem for sequence alignment is $\begin{matrix}\text{AA- CT}\\\text{AAACT}\end{matrix}$ how to select the proper objective function to guide the alignment process. For example, the second column of the template sequence is A, if at this time, it faces the thirds column of target sequence which is also A, the factors that may cause them to be matched are: one possible factor is the amino acid itself, say A match A, under such circumstance, the chance is high, and there might be other factors that influent the match result, let's say the following characters such the third column of template, which is C, and the fourth T, because of the existence of these characters, they reduced the possibility of A matching A at this time. We call all these factors "features".

In biological sequence alignment realm, there are basically two elements that forms feature. One is observations, which is the occurrence of sequence characters, for example, the third column of template is C, and this is the observation. Another is states, which is the "match", "delete" and "insert" result for a specific column. With the combination of these two basic elements, we could construct many features, for example, the following are some of the potential features:

Feature 1: the current state and the next state, since there are 3 possible states for each column, and each column could form a feature vector of length 9.

Feature 2: the current observation and the current state, for each column there are 20 kinds of amino acids (or 4 kinds of DNA or RNA) it could form a feature vector of length 20*3=60 or 3*4=12.

Feature 3: the combination of current observation and the next observation, with the current state. For each column it could form a feature vector of length 4*4*3=48.

For example: for the column 1 of the previous alignment example, the feature vector length is (9+12+48=69), and the 1st, 10th, 22nd position is set to 1, because the feature values are: match-match for feature 1, A-match for feature 2, and A-A-match for feature 3.

CRF is the mathematical tool to integrate all these features, it can be described as:

$$P(y\mid x,\lambda)=\frac{1}{Z(x)}\exp\sum_j \lambda_j F_j(y,x) \qquad (1)$$

In which, x stands for the observations and y stands for states, $Z(x)$ is a normalizer, it can be expressed as:

$$Z(x)=\sum_y \exp\sum_j \lambda_j F_j(y,x) \qquad (2)$$

In the formulas above, $F(y, x)$ stands for feature functions, y is the input of state, and x is the input of observation. We could define the form of feature function using binary function as follows:

$$f(x,y)=\begin{cases}1 & \textit{if the ith residue is A}\\ 0 & \textit{others}\end{cases} \qquad (3)$$

The problem discussed in this paper is on how to use CUDA to design algorithm to efficiently train the CRF model for sequence alignment. For how to use CRF model to align sequences please see [9]. The log likelihood function for $P(y|x,\lambda)$ is:

$$l(x,y,\lambda)=\sum_x[\sum_{j=1}^{d}\lambda_j F_j(y,x)-\log Z(x)] \qquad (4)$$

According to the Maximum likelyhood rule, we make partial deriviation on the $P(y|x,\lambda)$ to compute the according gradient for each weight, in this way we could update the weights in the gradient direction to reach the optimal point. We neglect the process of mathematical induction and get the following formula:

$$w_j=w_j+\alpha\times(F_j(x,y)-E_{\hat{y}\sim P(\hat{y}|x;w)}F_j(x,\hat{y})) \qquad (5)$$

In this formula α is a constant which represent the learning rate (velocity), $F_j(x, y)$ is the practical feature value of the trainning data (template sequence), and is the expectation of estimated feature value, it is hard to compute [23], therefore, we need some simplification to compute it, Collins Perceptron and Gibbs Sampling are the ones to solve this problem.

B. Collins Perceptron Training Algorithm

Collins Perceptron suppose that all of the probability mass are placed on a single state $\hat{y}$ which is mostly probable. It is: $\hat{y}=\arg\max_y p(y\mid x;w)$. The information included in this formula is: At the very beginning, use current weights vector w to compute a state(class) $\hat{y}$, then use this $\hat{y}$ to compute the feature value, this feature value is appriximately the same as the , then use this value to update the weight vector w, repeat this step until the w converges.

The formula of updating the w is as follows:

$$\begin{aligned}w_j&=w_j+\alpha F_j(x,y)\\ w_j&=w_j-\alpha F_j(x,\hat{y})\end{aligned} \qquad (6)$$

The problem is, how to compute the $\hat{y}$? There are two way to solve this problem, local based method and global based method:

Local based method suppose that there are no relattionship between states With the global based method, we will train the model by computing the states $\hat{y}$ as a whole, using a dynamic programming algorithm, typically using viterbi algorithm. For example:

observations: $\begin{matrix}\text{AA- CT}\\\text{AAACT}\end{matrix}$, states: {match,delete, insert}

In the example above, the state sequence is: match->match->insert->match->match.

If we use local based method, in column 3, we construct features by assuming states of {match, delete, insert} one by one, if we need the value of the combination of states to construct the feature vector. (For example, the current state and the previous state), we will use the original state in the training data, let's say the previous state of column 3 is match.

If we use global based method, we would not use the states in the training data, but use a dynamic programming matrix to train every possible state combinations (for example the current assumed state and every possible previous states).

C. Gibbs Sampling Training Algorithm

A method known as Gibbs sampling can be used to find the needed samples of $\hat{y}$. The updating of Gibbs sampling based method is the same as Collins Perceptron method and Gibbs sampling method is very similar to the local based Collins perceptron method, the difference between them are basically two points: 1) Gibbs sampling using randomly generated states as training data, and local based method using data in the training sets. 2) Gibbs sampling method should compute the states one by one, and global based Collins method could compute the states at the same time. Take the previous sample for example: Firstly, we assign a random state sequence to it, which might be delete->match->insert->match->match, then according to the most likely state for column 1, say it is match, then we update this state sequence to match->match->insert->match->match, then do this again in the second column, repeat this step until all the states are updated.

D. Time Complexity Analysis

Suppose that, the training sequence length is ***L***, and feature number is ***F***, and the iteration round number is ***R***, and the time complexity for local based Collins method is ***L*F*R***, for global based Collins method it is L^2*F*R. And for Gibbs Sampling based method, it is, ***L*F*R***.

IV. Parallel Algorithm and Optimization Methods

A. Parallel Collins Perceptron Algorithm

Pesudo Code 1: DoCRFTrain (seq_temp, seq_tar)

```
InitWeights();
While contrlValue < Thresh:
  Parallel_for: Column_i in columns of template:
     for: feature $F_i$ in features of $Column_n$:
        for: state $Y_k$ in three states of dependent Block:
           do:
              calculate $F_i(X, Y_k, j)$
              calculate $\hat{y}$
              UpdateWeights();
Done;
```

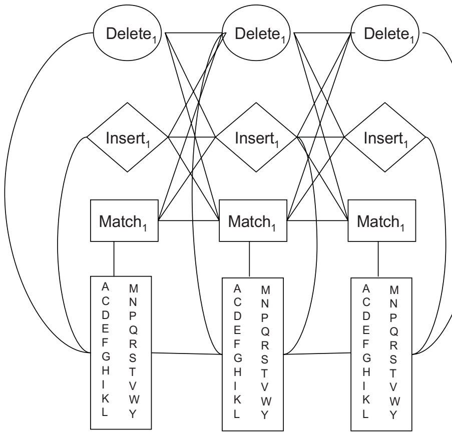

Figure 1 CRF feature selection for biological sequence Alignment

To discuss the parallel algorithm, we start from the local based Collins Perceptron algorithm. Assume that the feature we set is as the Figure 1 shows (this feature selection strategy will be used in all of the following three algorirthms), in the figure, each undirected links stands for the features, for example, the link between "match" and "delete", stands for the feature of the current state "delete" and the previous state "match". And there are link between a given state "match" and the amino acid alphabet box, which stands for the features of the current state "match" with one possible observation in the box.

The local method in itself is the process of iteratively updating the feature weights, since there are no data dependency between the feature weights, and there are no data dependency between different columns, so it is quite fit for the SIMT (Single Instruction, Multiple Threads) computing pattern of CUDA, the algorithm is shown in Peudo Code 1.

For the global based algorithm to train Collins Perceptron algorithm, it is different, it uses viterbi algorithm to get the state vector. And Viterbi Algorithm itself can be parallelized, so the trainning process become the parallelization of viterbi algorithm, we use the basic wave-front algorithm to do the parallelization tasks, and the algorithm could be described using Pseudo Code 2.

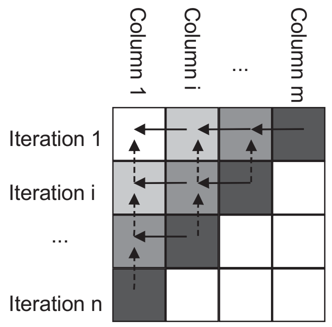

Figure 2, dependency analysis of Gibbs Sampling algorithm, and the way of paralleling different iterations.

B. Parallel Gibbs Sampling Algorithm

Gibbs Sampling algorithm is very similar to the process of local based Collins Perceptron CRF training algorithm. However it differs from the Collins based method in that. For each iteration, the current state should be computed after the computation of the previous state, in this way the **parallel_for** in the Pseudo Code 1, cannot be parallelized in Gibbs Sampling algorithm. Here we introduce the method of paralleling computation of different iterations which is "wave-front" like, see Figure 2.

Peudo Code 2: DoWaveCRFTrain (seq_temp, seq_tar)

```
InitWeights();
While contrlValue < Thresh:
  for: $round_r$ in all rounds:
    parallel_for: $block_{mn}$ in blocks of $round_r$:
      for: feature $F_i$ in features of $Column_n$:
        for: state $S_i$ in three states of $Block_{mn}$:
          for: state $S_j$ in three states of dependent Block:
          //when $S_i$ is Match, the dependent block is
            $Block_{(m-1)(n-1)}$
          // when $S_i$ is Delete, the dependent block is
            $Blcok_{(m-1)n}$
          //when $S_i$ is Insert, the dependent block is
            $Blcok_{m(n-1)}$
                do:
                calculate $F_i(X, Y_k, j)$
calculate $\hat{y}$ Traceback()
UpdateWeights();
```

In the Figure 2, though the data in the same iteration are strictly dependent on each other, but this is not true for data in different iterations (in the Figure, full lines represent the dependent relationship, and the dotted lines represent the independent relationship). Under such dependency condition, data marked with the same color are independent of each other which can be calculated in parallel.

One problem is , this wave-front algorithm is different from the wave-front pattern to parallel viterbi algorithm [20], for we know how many rows in the dynamic programming matrix, but we do not know how many iterations there will be in the Gibbs Sampling based method. So, there must be redundant computations with this wave-front manner if we compute all the iterations permitted at the same time. One way to solve this problem is to "predict" how many iterations there will be, the parallel algorithm is show in Pseudo Code 3.

Peudo Code 3: DoWaveCRFTrain (seq_temp, seq_tar)

```
InitOriginW();
While contrlValue < Thresh:
  for: K $round_r$ in all rounds:
    parallel_for: $block_{mn}$ in blocks of $round_r$:
      for: feature $F_i$ in features of $Column_n$:
        for: state $Y_k$ in three states of dependent Block:
          do:
            calculate $F_i(X, Y_k, j)$

          calculate $\hat{y}$
          UpdateW();
          judgeWhichRound()
```

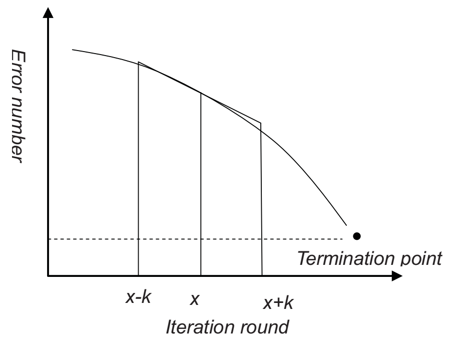

Figure 3 The curve of the learning process .

In this algorithm, we predict the iteration number K with a fixed value to prevent too many redundant computation, for example $K = 10$, with this strategy, there could be at most 9 redundant computations, and the larger K is, the higher parallelization we could achieve with higher probability of doing more redundant computations.

PROBLEMS AND OPTIMIZATION METHODS

A. How to Assign Memory and Threads?

Assigning memory and threads are very important for promoting the performance of CUDA accelerated algorithm. In implement our method, we put the small but often accessed memory in the shared memory and put large but less often accessed memory in global memory. As for the thread scheduling, we use the optimization method in [24], to make the memory access better.

B. How to Predict Iteration Round Number for Gibbs Sampling?

Previously, we proposed method of setting a defined K to parallel the computation of different iterations, in this method the K are hard to select, because for different training samples the K might be different to reach the optimal performance. Suppose the learning curve is as the Figure 3 shows, we could see that if the learning process is converging, and the previous reduced error number is the area of the trapezoid and we could predict the remaining K, therefore we could see K as a variant not a static value, and the method to compute K is as follow:

Method : half the iteration round

1) Compute the slope (we mark it as *sl*) according to the first and last iteration reduced error number (let's say *e1* and *e2*) of the previous round.
2) If remained error number is marked as *re*, and $re-K*(2K-K*sl)/2$ is larger than termination point (which is marked as *term*), then K = mid-point of the expecting rounds, else solve the formula ($re - term$) = $K*(2K-K*sl)/2$ to get the K.

V. EXPERIMENTAL RESULTS

The experiments are performed on the platform which has a dual-processor Intel 2.83GHz CPU with 4 GB memory and an NVIDIA Geforce 9800 GTX GPU with 8 streaming processors and 512MB of global memory. We tested using Windows XP system. And the experiments are run on both debug and release mode. To focus on the algorithmic efficiency in our study, we made two simplifications in our experiments, one is that we use a pseudo count method [29] to train the CRF, and another is that we neglected the discussion of accuracy for our experiments (because we lack the training data set and theory preparation to train the previous knowledge,). We employ the automatic sequence-generating program ROSE [25] to generate different test cases.

A. Test of Collins Perceptron

The test of Collins Perceptron is divided into two parts, the local based method and the global based method. We select groups of sequences which have lengths less than 2000 to test both of the two methods. The experimental results for local based method are shown in Table I.

TABLE I. PERFORMANCE COMPARISM OF LOCAL BASED TRAINING METHODS

Sequence-Length		***Execution Time (Second)/Speedup***			
		Debug		Release	
500	*serial*	1.531	1.814	0.718	0.919
	GPU	0.844		0.781	
1000	*serial*	2.671	3.351	1.265	1.528
	GPU	0.797		0.828	
1500	*serial*	4.437	5.681	2.109	2.753
	GPU	0.781		0.766	
2000	*serial*	7.296	9.342	3.625	4.548
	GPU	0.781		0.797	

From the table we could see that our algorithm achieved acceleration comparing to the serial version, and the longer the sequence is, the higher acceleration performance it will be. However, there are two problems indicated by this experiment:

1) The acceleration rate is not high enough as we expected, as our previous analysis, the local based algorithm should fit the SIMT computation mode most, but the truth is not like that, this might be related to the small problem size itself.
2) When the sequence length is small, the acceleration rate is not obvious, to solve this problem, we must unite other local based method tasks as a whole to promote the usage of GPU and the performance.

Table II shows the result of global based Collins Perceptron algorithm, because the running time for viterbi algorithm is long, the experimental results show the average time for each iteration.

TABLE II. PERFORMANCE COMPARISM OF GLOBAL BASED TRAINING METHODS

Sequence-Length		***Execution Time (Second)/Speedup***			
		Debug		Release	
500	*serial*	1.72	8.113	1.03	4.813
	GPU	0.212		0.214	
1000	*serial*	5.27	13.077	3.17	5.591
	GPU	0.403		0.567	
1500	*serial*	13.95	15.587	8.37	10.07
	GPU	0.895		0.831	
2000	*serial*	28.17	22.357	16.7	12.945
	GPU	1.26		1.29	

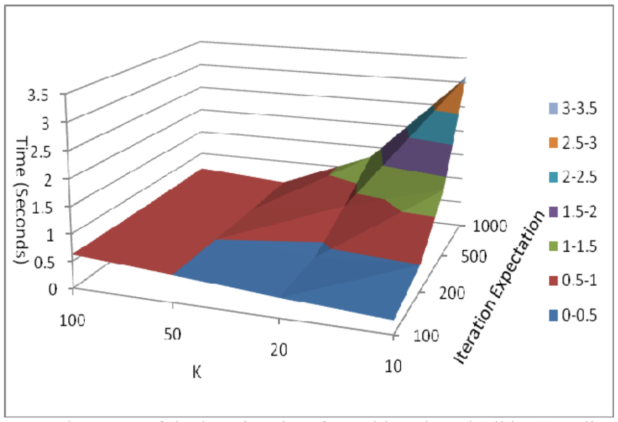

Figure 4 The curve of the learning time for stable K based Gibbs Sampling.

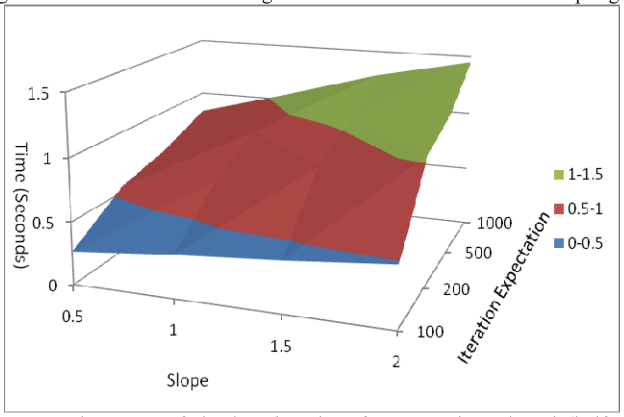

Figure 5 The curve of the learning time for Dynamic K based (half the iteration expectation) Gibbs Sampling.

1) As table II shows, comparing to the local based algorithm, the acceleration rate is higher. This is because the problem size for global based algorithm is larger than the serial version, and under such circumstances, the GPU might be better prepared for the work. In addition, we used the methods of partition different kind of computations as shown in [20], and because the computation of a single kernel is very large, divide it will obviously increase the utilization of GPU.

B. Test of Gibbs Sampling Algorithm

As for the Gibbs sampling algorithm, there are two ways of getting the proper “jumping step” *K*--the stable method and the variant method. The experiment is executed on the sequence of length 500, the iteration expectation range from 100 to 1000, and for the case of stable *K* the *K* is ranging from 10 to 100, for the case of dynamic *K*, the slopes are ranging from 0.2 to 2. The figure from 4 to 5 shows the experiment results. From the figures, we could see that, comparing to the variant methods, the stable methods spend more time to train the model on average, when the *K* is less than about 50 the performance will be worse than the dynamic methods. What’s more the performance of dynamic *K* based algorithm is steadier with the variation of iteration expectation comparing to the stable *K* based algorithm. This is a very important result, for in the real application, we cannot assure that the iteration number is just as our expectation.

Finally, table 3 shows the results on the test of the execution time on different length of sequence, we used the method of stable method which set *K=50*. Comparing to the local based parallel Collins Perceptron training algorithm, the parallel

Gibbs sampling algorithm is a little worse, this is because that their work load are the same, but the thread load for Gibbs sampling method is unbalanced, smaller than Collins method.

TABLE III. PERFORMANCE COMPARISM OF GIBBS SAMPLING METHODS

Sequence-Length	***Execution Time (Second)/Speedup***			
	Debug		Release	
500	0.25	6.124	0.25	2.872
1000	0.42	6.36	0.469	2.697
1500	0.66	6.723	0.735	2.869
2000	0.97	7.522	1.06	3.42

VI. CONCLUSION AND FUTURE WORK

In this article, we analyzed the Conditional Random field model and its application on the Biological Sequence alignment, we designed the parallel version of training sequence alignment oriented CRF training algorithm (which also includes many optimization ideas), experiment shows that our method perform well on GPU card with CUDA, still there are more work to be done which are listed as follows: 1) Much work should been done on our algorithm to support arbitrarily large feature sets. 2) We need to integrate our work with the work done by Chun Do et al [9] and their MPI based coarse grained parallel methods.

VII. ACKNOWLEGEMENT

This paper is partly supported by National Natural Science Foundation of China (No. 61073008 and No. 60773148), Beijing Natural Science Foundation (No. 4082016), NSF Grants IIP-0934114 and OCI-0904461, NIH award RC2 HG005542, and the NVIDIA CUDA Center of Excellence at Georgia Tech.

REFERENCES

[1] Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proc. 18th International Conf. on Machine Learning, Morgan Kaufmann, San Francisco, CA (2001) 282–289

[2] McCallum, A.: Efficiently inducing features of conditional random fields. In: Proc. 19th Conference on Uncertainty in Artificial Intelligence. (2003)

[3] Sha, F., Pereira, F.: Shallow parsing with conditional random fields. Technical Report MS-CIS-02-35, University of Pennsylvania (2003)

[4] Sarawagi, Sunita; William W. Cohen (2005). "Semi-Markov conditional random fields for information extraction". in Lawrence K. Saul, Yair Weiss, Léon Bottou (eds.). Advances in Neural Information Processing Systems 17. Cambridge, MA: MIT Press. pp. 1185-1192.

[5] Leaman, R., Gonzalez, G.: BANNER: An executable survey of advances in biomedical named entity recognition. In 'Pacific Symposium on Biocomputing'

[6] Christopher M. Bishop Neural Networks for Pattern Recognition Oxford England Oxford University Press.

[7] Minsky M. L. and Papert S. A. 1969. *Perceptrons*. Cambridge, MA: MIT Press.

[8] J. Shan, Y. Chen, Q. Diao, Y. Zhang. Parallel information extraction on shared memory multi-processor system. In Proc. of International Conference on Parallel Processing, 2006.

[9] Do, C.B., Gross, S.S., and Batzoglou, S. (2006) CONTRAlign: Discriminative Training for Protein Sequence Alignment. In Proceedings of the Tenth Annual International Conference on Computational Molecular Biology (RECOMB 2006).

[10] J. M. Nageswaran, et al. A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors,Special issue of Neural Network, Elsevier, vol. 22, no. 5-6, pp. 791-800, July 2009.

[11] Mohammad A. Bhuiyan, Vivek K. Pallipuram and Melissa C. Smith Acceleration of Spiking Neural Networks in Emerging Multi-core and GPU Architectures In HiComb 2010 Atlanta 2010

[12] Kevin P. Murphy "An Introduction to Graphical Models" 2001

[13] L.R Rabiner "A tutorial on hidden Markov models and selected applications in speech recognition". In Proceedings of the IEEE, Vol. 77, No. 2. (06 August 2002), pp. 257-286.

[14] Charles Elkan Log-linear Models and Conditional Random Fields ACM 17th Conference on Information and Knowledge Management, tutorial, 2008

[15] Y. Liu, W. Huang, J. Johnson, and S. Vaidya, GPU Accelerate Smith-Waterman, Proc. Int'l Conf. Computational Science (ICC 06) pp.188-195,2006

[16] Y. Munekawa, F. Ino, and K. Hagihara. Design and Implementation of the Smith-Waterman Algorithm on the CUDA-Compatible GPU. 8th IEEE International Conference on BioInformatics and BioEngineering, pages 1 C6, Oct .200

[17] S.A. Manavski, G. Valle. CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. BMC Bioinformatics. 2008 Mar 26;9 Suppl 2:S10

[18] R. Horn, M. Houston, P. Hanrahan. ClawHMMer: A streaming HMMer –search implementation. Proc. Supercomputing (2005).

[19] I. Buck, T. Foley, D. Horn, J. Sugerman , K. Fatahalian, M. Houston, P. Hanrahan. Brook for GPUs: Stream Computing on Graphics Hardware (2004) ACM Trans. On Graphics.

[20] Zhihui Du, Zhaoming Yin, David. A Bader, A Tile-based Parallel Viterbi Algorithm for Biological Sequence Alignment on GPU with CUDA IEEE International Parallel and Distributed Processing Symposium (IPDPS) —HiComb Workshop, Atlanta USA, 2010

[21] Smith, Temple F.; and Waterman, Michael S. (1981). "Identification of Common Molecular Subsequences". Journal of Molecular Biology 147: 195–197.

[22] Berger et al.: A. Berger, A. Della Pietra, and J. Della Pietra. A maximum entropy approach to natural language processing. Computational Linguistics, pp.39-71, No.1, Vol.22, 1996

[23] Klinger, R., Tomanek, K.: Classical Probabilistic Models and Conditional Random Fields. Algorithm Engineering Report TR07-2-013, Department of Computer Science, Dortmund University of Technology, December 2007.

[24] Shane Ryoo Christopher I.Rodrigues Sara S. Baghsorkhi Sam S. Stone David B. Kirk Wen-mei W. Hwu Optimization Principles and Application Performance Evaluation Of a Multithreaded GPU Using CUDA Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming Salt Lake City, UT, USA 2008

[25] J. Stoye, D. Evers and F. Meyer. "Rose: generating sequence families". In Bioinformatics. 1998;14(2):157-163

[26] A. Mccallum , D. Freitag , Fernando Pereira Maximum Entropy Markov Models for Information Extraction and Segmentation Proceedings of the Seventeenth International Conference on Machine Learning Pages: 591 – 598 2000

[27] M. Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing, pp. 1-8, 2002.

[28] Notredame C. Recent progresses in multiple sequence alignment: a survey Pharmacogenomics. 2002 Jan;3(1):131-44.

[29] Jorja G. Henikoff , Steven Henikoff , Howard Hughes: Using substitution probabilities to improve position-specific scoring matrices Computer Applications in the Biosciences 1996.