2009 IEEE International Symposium on Parallel and Distributed Processing with Applications

A Parallel Gibbs Sampling Algorithm for Motif Finding on GPU

Linbin Yu
Department of Computer Science
University of Science and Technology of China
Hefei, P.R.China
Email: lbyu@mail.ustc.edu.cn

Yun Xu
Department of Computer Science
University of Science and Technology of China
Hefei, P.R.China
Email: xuyun@ustc.edu.cn

Abstract—**Motif is overrepresented pattern in biological sequence and Motif finding is an important problem in bioinformatics. Due to high computational complexity of motif finding, more and more computational capabilities are required as the rapid growth of available biological data, such as gene transcription data. Among many motif finding algorithms, Gibbs sampling is an effective method for long motif finding. In this paper we present an improved Gibbs sampling method on graphics processing units (GPU) to accelerate motif finding. Experimental data support that, compared to traditional programs on CPU, our program running on GPU provides an effective and low-cost solution for motif finding problem, especially for long motif finding.**

Keywords- Motif; Gibbs Sampling;CUDA;GPGPU;

I. Introduction

DNA motif is defined as a short (5 to 20 base-pairs) pattern of nucleic acid sequence, which has some biological significance such as transcription regulation. Given a set of nucleic acid sequence, the main object of motif finding is to detect conserved short pattern overrepresented in these sequences. Motif finding is an important and fundamental problem in computational genomics, and various algorithms for motif finding have been published, which can be classified into two categories: deterministic algorithms and non-deterministic algorithms. Deterministic algorithms such as YMF [1], MITRA [2] and Weeder [3] rely on exhaustive searching, and always guarantee the global optimality of results. Due to high computational complexity, the deterministic algorithms have difficulty to process long motifs. Non-deterministic algorithms such as Gibbs Sampling [4], Expectation Maximization (EM) [5], and Random Projection [6] are more effective for finding long motifs, although they can not guarantee the global optimal solution, and may converge to a locally optimal one.

With the explosive growth of genomic sequence data, the requirement of computing capability in motif finding also grows rapidly. Besides traditional PC clusters and supercomputers, some kinds of specific hardware have been developed to accelerate these computations. Typical hardware-accelerated devices including GPU and FPGA have been applied in many areas successfully. In bioinformatics, W Liu et al. designed many sequence analysis algorithms including Bio-sequence database scanning [7] and GPU-ClustalW [8] on GPU relies on graphics API. Svetlin Manavski and Giorgio Valle implemented a Smith-Waterman alignment program using the novel GPU platform named CUDA [9]. The first algorithm on GPU for motif finding is GPU-MEME [17], which relies on OpenGL. To our knowledge, there is no motif finding algorithm based on CUDA platform.

In this paper, we introduce an improved Gibbs sampling algorithm using Nvidia CUDA, which utilizes modern graphics card to accelerate computation of motif finding. The rest of paper is organized as follows. Section 2 provides necessary background about GPU and CUDA. In section 3 we introduce an improved Gibbs sampling algorithm on CUDA. Section 4 focused on performance and optimization. Finally, experimental results and conclusion are presented in section 5 and 6 respectively.

II. Overview of GPU and CUDA

General Purpose Computing on Graphics Processing Units (GPGPU) is a hot trend in high performance computing research. A modern GPU contains large number of multiprocessors, each of which can execute the same program on separate data set respectively. Nowadays the peak performance of consumer graphics cards such as Nvidia GTX200 serials and AMD RV770 serials are ten or more times faster than comparable consumer CPUs. Furthermore, performance of GPU increased from more than two times per year, which is also much faster than Moore's law. Besides drawing graphics, GPU has been granted more duties of processing massive and scientific computation. To facilitate the development on GPU for general purposes, the two main graphics card providers, NVidia and AMD have developed their new GPGPU platforms CUDA [10] and CTM [11] respectively.

In CUDA programming model, "kernels" are defined as specific functions which are invoked from CPU side (the "host") and executed on GPU (the "device") by many threads. Every thread executes the kernel once. Several threads are organized into "blocks", and in the same block threads can cooperate together though shared memory and synchronous instructions. Furthermore, blocks are organized into "grid", and one grid responses for one CUDA kernel. The distinct difference between CUDA and other parallel language (such as MPI) is, the organization of threads is mainly adaptable to the user's needs, rather than the hardware resource. The architecture of CUDA compatible device is displayed in Figture.1. CUDA device is built around a set of Streaming Multiprocessors, which consists of 8 Scalar Processor cores and on-chip shared memory. Concurrent threads are created and managed by multiprocessor, and executed in hardware without any scheduling overhead. When a CUDA kernel is invoked by host CPU, the thread blocks in the thread grid are distributed to all available multiprocessors and are processed in parallel.

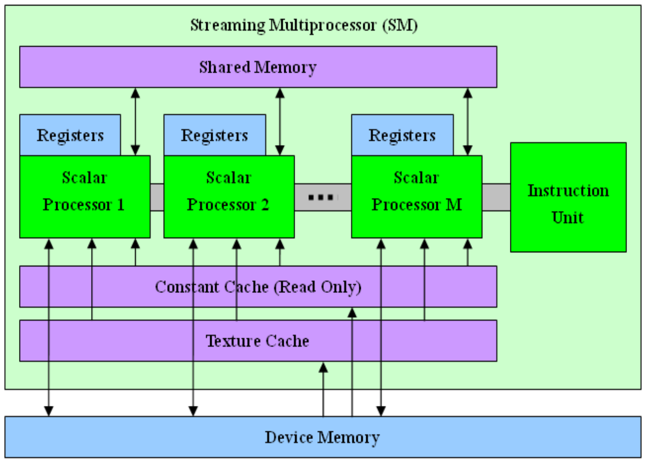

Figure 1. Architecture of CUDA compatible device

III. METHOD

A. Basic Gibbs Sampling Algorithm

Gibbs sampling is a kind of Markov Chain Monte-Carlo procedure which was widely used in finance, and was first applied in bioinformatics for multiple sequence alignment in 1993 [4]. In the following years, many extensions such as AlignACE [12], MotifSampler [13] and GibbsST[14] are published.

Given a set of n sequences S_1, S_2, ..., S_n with uniform length l, and an integer W, the main object of motif finding is to find a overrepresented pattern (motif) with width W, which can be portrayed as a Position Specific Scoring Matrix (PSSM):

$$\begin{pmatrix} q_1^A & q_2^A & \cdots & q_l^A \\ q_1^C & q_2^C & \cdots & q_l^C \\ q_1^G & q_2^G & \cdots & q_l^G \\ q_1^T & q_2^T & \cdots & q_l^T \end{pmatrix}$$

In this matrix, each q_i^x represents the probabilities of finding nucleotide x at position i of the motif. Similarly, the background probabilities $\boldsymbol{B} = \{b_A, b_C, b_G, b_T\}$, which represents the probability distribution of background nucleotide in these sequences. The starting positions $\boldsymbol{R} = \{r_1, r_2 \dots r_n\}$ lists n starting points of n "sampled" segments within each sequences. These segments constitute an alignment which is used to calculate the PSSM.

The Gibbs sampling algorithm attempts to maximize the similarity between these segments. It starts with random initial positions within each sequence, and proceeds through some iterative procedures by two steps.

- Updating Step. In updating step, chose one sequence $S_{updating}$ in random or in specified order, then calculate the PSSM by background probabilities $\boldsymbol{B}$ and each starting positions in $\boldsymbol{R}$ excluding the entry from sequence $S_{updating}$.
- Sampling Step. In sampling step, two probability are calculated. The first one is the probability of each segment generated by the present pattern probabilities P_m. The second one is the probability of each segment generated by the background probabilities P_b. The ratio $P_i = P_m / P_b$ represents the comparative similarity between selected segments and the current PSSM. Then we select a new starting position randomly according to P_i, or simply choose the position with maximum P_i.

Repeat these iterative procedures until convergence, or exceeding a specific iteration times set by user, and the expected motif is given by the final PSSM. The motif sequence can be calculated by PSSM easily, by choosing the character with maximum value in each column.

To quantitative the quality of motif, final PSSM can be evaluated by the average relative entropy score, which tells how much the motif differing from the background nucleotide distribution. The average relative entropy can be calculated by the following expression.

$$\frac{1}{W}\sum_{j=1}^{W}\sum_{x\in\{A,C,G,T\}} q_j^x \log_2 \frac{q_j^x}{b_x} \qquad (1)$$

This score is maximal if the found motif is perfectly conserved and differs considerably from the background nucleotide probabilities.

B. Parallelization on CUDA platform

As a kind of Markov Chain procedure, Gibbs sampling is hard to parallelize because the current status are correlate with the previous one. To maximize the utilization of the computing resources of GPU, a simple strategy is adopted in our improved Gibbs sampling algorithm. We select a sequence in updating step in order, and select the most possible segment instead of random one in the sampling step. The forfeit of randomicity in the sampling step decreases the possibility of finding eligible motifs, but it can be compensated by more running times because the execution time also decreases severely when the deterministic strategies are used.

In our implementation, a modified Gibbs sampling process (noted as "G-process") is executed by a single thread. Each thread starts with different initial status, which is generated randomly by random number generator (RNG) and a different random seed. There is no RNG in Random numbers generated by RNG are fixed sequences in essential, so the final result of single G-process is also fixed, only depending on the first random seed. In particular, each thread maintains a different number as random seed, to generate initial starting positions and the following random numbers. A fast version of Park–Miller RNG [15] is implemented in our program because CUDA doesn't provide RNG on GPU device.

In the enormous possibilities of initial starting positions, few of them could lead to eligible results. If the probability of finding an eligible result from a random initial status is *p*, the probability of finding it in *n* independent G-processes raises to $1-(1-p)^n$, which is equal to *np* approximately when *p* is considerable small.

The full algorithm on CUDA is presented as following. Step 2 to step 8 are executed on GPU using CUDA.

```
1. Program initiation
2. For each thread on GPU
3.     Select a random seed
4.     Generate initial status
5.     Select one sequence in order, and then calculate
       the PSSM (Updating step)
6.     Select a segment with maximum possibility as
       new starting point (Sampling step)
7.     Go to step 5 unless iteration convergence, or
       iteration times exceeds the specific number
8.     Calculate and return the average relative entropy
       score of final PSSM
9. Find the thread with maximum entropy score, and
   return its PSSM and motif sequence
```

IV. Optimization

A. Improve PSSM score calculation

PSSM and average relative entropy score will be recalculated in every updating step of G-process, but actually it just need to update from the previous status, instead of recalculating.

Note $f(q_j^x) = q_j^x \log_2 \frac{q_j^x}{b_x}$. The last excluded sequence S_t should be added into PSSM, so we can calculate the correcting value of S_t at position *j* as

$$C_t[j] = f\left(\frac{q_j^{S_t[j]}(n-1)+1}{n-1}\right) - f\left(q_j^{S_t[j]}\right) \tag{2}$$

And the correcting value of current exclusive sequence S_{t+1} at position *j* is

$$D_{t+1}[j] = f\left(\frac{q_j^{S_{t+1}[j]}(n-1)-1}{n-1}\right) - f\left(q_j^{S_{t+1}[j]}\right) \tag{3}$$

Then the score in *n+1* step can be calculated using these two correcting values:

$$Score_{t+1} = Score_t + \sum_{j=1}^{W} C_t[j] + \sum_{j=1}^{W} D_{t+1}[j] \tag{4}$$

We can calculate and store *2n* values for $\boldsymbol{C_t}$ and $\boldsymbol{D_{ts+1}}$. After the first calculation of PSSM, only several additions are required for updating. This improvement can save more than 90% time expending in updating step. A similar improvement is also adopted by expectation calculation in GPU-MEME [17].

B. Reorganization of threads

In a multiprocessor, eight Scalar Processors operate in SIMD style under the control of a single instruction sequencer. The threads are organized into groups of 32 threads (named "warp") and only threads on the same path can be executed simultaneously. Divergent threads in a warp are handled with extra time so minimizing SIMD divergence will greatly raise the performance.

In this program, the most divergences are the different iteration times. Execution times of threads with different initial status are very different, and usually 30% time is wasted in divergence. To explore the best way of avoiding divergence, we tested a data set with 20 sequences, and recorded all the iteration steps of 10240 threads in Figure. 2. The maximal iteration time of all threads in this test case is set to 100.

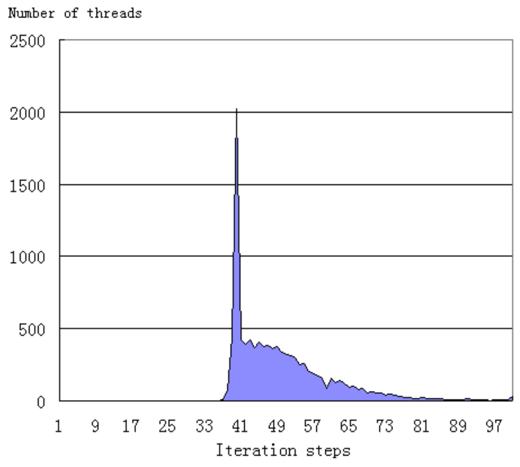

Figure 2. The thread distribution on iteration steps

The phenomenon of more than 25% threads stopping before the 40th step can be explained as following. The date set contains 20 sequences, and many PSSM are converged after updating 20 times (one updating per sequence in most situation). After converged, it still need additional 19 iteration step to validate the convergence, because the updating sequence are back to the first one after a cycle.

Thread reorganization is utilized to reduce the time wasting. Compared to the uniform iteration step limitation, several checkpoints are set to reorganize threads by removing the terminated threads timely. The first checkpoint of problem with *n* sequences should be set to *2n*. When the first thread grid with *2n* steps is finished, all terminated threads will be removed and the second CUDA kernel (also means the second thread grid) with additional iteration, say *n/2* times, will be executed, until there are not enough threads to fill in a thread block. After applied this improvement, about 25% time consuming are saved in this test case.

V. RESULTS

Our algorithm is tested on Nvidia 9800GT graphics card running at 1.4GHz on Windows. The problem size is set to 20 nucleic acid sequence with uniform length 600, and the length of target motif includes 15, 17 and 19. All sequences are artificial, and are generated randomly by a useful tool rMotifGen [16] with 80% conservation.

All the exact motifs are found in testing cases when the number of G-process is set 8192, which proves the effectiveness of our algorithm on CUDA platform.

We also make a comparison of the execution time on different platforms. The same algorithm is implemented using OpenMP and well optimized on a common PC with Intel Dual-Core E2140 processor. The results are recorded in table I.

TABLE I. COMPARISON OF GPU AND CPU PLATFORM

Motif Width	GPU (ms)	CPU (ms)	Speedup
15	625	6718	10.7
17	687	8312	12.1
19	734	8422	11.5

Gibbs Sampling on GPU is more than ten times faster than common multi-core CPU. Considering the low cost of graphics card (less than 150 USD for Nvidia 9800GT), GPU can provide tremendous high performance/cost ratio for compute-intensive problem like motif finding.

VI. CONCLUSION

In this paper we introduced an improved Gibbs Sampling algorithm on GPU for motif finding. We tested our algorithm on graphics card, and achieved more than 10 times speedup compared to the same algorithm running on CPU. The results of this work show that the graphic cards can be considered as efficient hardware accelerators for motif finding, and other compute-intensive problems in many areas.

Future work includes improving the resistance of local optima, supporting protein motif discovery, and multiple motifs finding. Furthermore, we also plan to implement more bioinformatics algorithms on CUDA platform.

REFERENCES

[1] Sinha S, Tompa M, A statistical method for finding transcription factor binding site. Proceedings of the Eighth International Conference on Intelligent Systems on Molecular Biology, San Diego, CA 2000:344-354.

[2] Eskin E, Pevzner P, Finding composite regulatory patterns in DNA sequences. Bioinformatics 2002, 18(Suppl 1):S354-S3633.

[3] Pavesi G, Mauri G, Pesole G, An algorithm for finding signals of unknown length in DNA sequences. Bioinformatics 2001, 17(Suppl 1):S207-S214.

[4] Lawrence CE, Altschul SF, Boguski MS, Liu JS, Neuwald AF, Wootton JC, Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. Science 1993, 262:208-214.

[5] Lawrence CE, Reilly AA: An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. Proteins 1990, 7:41-51.

[6] Buhler J, Tompa M, Finding motifs using random projections. J Comput Biol 2002, 9:225-242.

[7] W. Liu, B. Schmidt, Voss G, Schroeder A, W. Muller-Wittig, Bio-Sequence Database Scanning On GPU. In Proceeding of the 20th IEEE International Parallel & Distributed Processing Symposium: 2006

[8] W. Liu, B. Schmidt, G. Voss and W. Müller-Wittig, GPU-clustalw: Using Graphics Hardware to Accelerate Multiple Sequence Alignment. in 13th Annual IEEE International Conference on High Performance Computing (HiPC 2006), LNCS 4297, 2006, pp. 363–374.

[9] Svetlin A. Manavski and Giorgio Valle, CUDA Compatible GPU Cards as Efficient Hardware Accelerators for Smith-Waterman Sequence Alignment. BMC Bioinformatics 2008, 9(Suppl 2):S10

[10] NVidia CUDA, http://www.nvidia.com/cuda

[11] AMD CTM, http://ati.amd.com/products/ streamprocessor/

[12] Roth FP, Hughes JD, Estep PW, Church GM, Finding DNA regulatory motifs within unaligned noncoding sequences clustered by whole-genome mRNA quantitation. Nature Biotechnology 1998, 16:939-945.

[13] Thijs G, Marchal K, Moreau Y, A Gibbs sampling method to detect over-represented motifs in upstream regions of coexpressed genes. RECOMB 2001, 5:305-312.

[14] Shida K, GibbsST: a Gibbs sampling method for motif discovery with enhanced resistance to local optima. BMC Bioinformatics 2006, 7:486.

[15] Park, S. K. and K. W., Miller, Random Number Generators: Good Ones are Hard to Find. Comm. ACM 31, 1192-1201 (1988).

[16] R Eric, HC Timothy, rMotifGen: random motif generator for DNA and protein sequences. BMC Bioinformatics, 2007, 8: 292.

[17] C Chen, B Schmidt, L Weiguo, W Müller-Wittig, Using Graphics Hardware to Accelerate Motif Finding in DNA Sequences. Proceedings of the Third IAPR International Conference on Pattern Recognition in Bioinformatics, 2008: 448 – 459