Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/072L28

Accelerating GOR Algorithm Using CUDA

Xinbiao Gan[1,2], *Cong liu*[1,2], *Zhiying Wang*[2], *Li Shen*[2], *Qi Zhu*[2], *Jie Liu*[1], *Lihua Chi*[1], *Yihui Yan*[1] *and Bin Yu*[3]

[1]School of Computer, National University of Defense Technology University, Changsha 410073, China
[2]State Key Laboratory of High Performance Computing, National University of Defense Technology, Changsha 410073, China
[3]CIty college, Wuhan University of Science and Techology, Wuhan 430083, China

Received: 18 Oct. 2012, Revised: 15 Jan. 2013, Accepted: 18 Jan. 2013
Published online: 1 Jun. 2013

Abstract: Protein secondary structure prediction is very important for its molecular structure. GOR algorithm is one of the most successful computational methods and has been widely used as an efficient analysis tool to predict secondary structure from protein sequence. However, the running time is unbearable with sharp growth in protein database. Fortunately, CUDA (Compute Unified Device Architecture) provides a promising approach to accelerate secondary structure prediction. Therefore, we propose a fine-grained parallel implementation to parallelize GOR-IV package for accelerating protein secondary structure prediction, in which each amino acid would be assigned to one single CUDA thread, hence protein secondary structure prediction would be parallelized by many CUDA threads simultaneously, and constant cache is resorted to cache parameter table. Experimental results show a speedup factor is more than 173X over original GOR-IV version.

Keywords: Protein secondary structure prediction, GOR, fine-grained, onstant cache, CUDA

1 Introduction

Protein structure prediction plays a very important role in determining functions in biological systems, and protein tertiary structure prediction is one of ultimate goals for protein science. However, methods including homology modeling[1]. protein fold recognition [2], and ab initio modeling [3] for Protein tertiary structure prediction is complex and unfeasible. Consequently, Instead of predicting 3D structure directly, it is much easier to predict fundamental elements of protein secondary structure including a-helices, b-sheets, coils, and turns. All these elements can be easily observed in protein 3D structure and can serve as an input for protein tertiary structure prediction successfully.

Fortunately, several methods such as GOR method[4, 5], Hydrophobic-Polar model[6] and AI methods [7,8] have been proposed for predicting 2D structure from amino acid sequence. Particularly, GOR algorithm is one of the earliest and most successful methods for secondary structure prediction from protein sequence. It is based on the information theory combined with the Bayesian statistics. Although protein 2D structure prediction based on GOR is efficient but execution time is intolerable with steep growth in protein database. ThereforeXia presented a fine-grained parallel hardware implementation based on FPGA accelerator and attained speedup factor of 430X[9], while customization of FPGA accelerator is complex, expensive and unfeasible. Accordingly, we introduce many-threaded GPU to parallelize GOR, which is feasible and efficient for parallelizing protein Secondary Structure Prediction.

2 Motivation

2.1 GOR algorithm

GOR program is one of the first major methods proposed for protein secondary structure prediction from sequence. The original version (GOR-I) was released in 1978 by Garnier, Osguthorpe and Robson[4,5].

The basic idea of GOR method is the use of information theory and Bayesian statistics method to relate amino acid sequence for protein secondary structure prediction. It takes into account not only the propensities of individual amino acids to form particular secondary structures, but also conditional propensities of amino acid to form a secondary structure when its

* Corresponding author e-mail: xinbiaogan@163.com

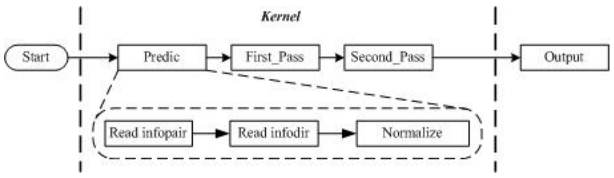

Fig. 1 Dataflow of kernel for GOR

immediate neighbors have already formed that structure [4,5,9].

In the past twenty years, GOR algorithm has been improved by larger structure databases and more detailed statistics. The GOR-IV analyzes sequences to predict *alpha-Helix*(**H**), Extended *beta-sheet*(**E**), or *Coil*(**C**) secondary structure at each position based on 17-amino acid sequence windows to consider information of local segment, Eight nearest neighboring residues on each side are considered for a given residue and a database of 267 sequences with known secondary structure to calculate information function[9].

In practice, GOR runs with a single protein sequence as input, the kernel of the algorithm executes in three stages, as shown in figure 1. The First stage (*Predic*), it predicts the 2D structure for each input amino acid. The latter two stages (*first_pass* and *second_pass*), which perform scanning procedure to correct the secondary structure generated by the first stage.

2.2 Motivating warm-up

To identify performance bottleneck of GOR and then accelerate and optimize the kernel component of GOR, we have done a warm-up testing on original GOR-IV package. Our study shows that first stage (*Predic*) takes up 99% of total execution time, as described in figure 2.

Therefore, we would accelerate first stage (*Predic*) using many-threaded CUDA because it is critical to accelerate the GOR program.

3 Architecture

Usually, modern computer systems are composed of CPU and GPU. Figure 3 shows the architecture of collaborative system, in which GPU can be used as an accelerator to consume data transferred from CPU into GPU with PCIe channel on demand.

In above system architecture, GPU architecture consists of a scalable number of streaming multiprocessors (SMs), a read-only constant cache, and a read-only texture cache. Each SM contains eight streaming processor (SP) cores and every three SMs constitute a threading multiprocessor cluster (TPC) in NVidia GTX 280, Additionally, Each SM has16KB

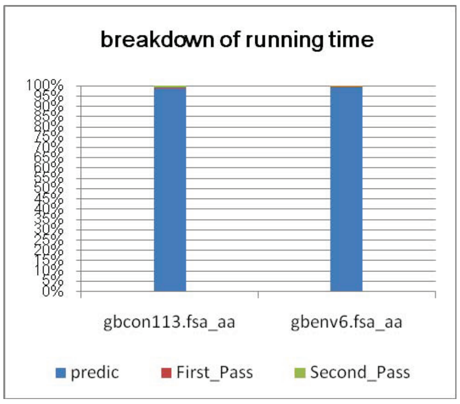

Fig. 2 Warm-up running on GOR

shared memory which is partitioned into 16 banks and common to all 8 SPs inside it.

In CUDA-enabled GPU, instructions are structured in SPMD (Single Program, Multiple Data), and CUDA execution model provides three key abstractions [10], a hierarchy of thread groups, shared memories, and barrier synchronization. Threads have a three level hierarchy. A grid is a set of thread blocks that execute a kernel function. Each grid consists of blocks of threads. Each block is composed of hundreds of threads. Threads within one block can share data using shared memory and can be synchronized at a barrier. All threads within a block are executed concurrently in a form called warp, which is composed of 32 parallel threads, and Instructions are scheduled and managed based on warp in SIMT (Single Instruction, Multiple Threads) architecture. Consequently, thread-level parallelism is prone to exploited for Protein secondary structure prediction according to CUDA execution model as shown in Figure 3.

4 Methodology

4.1 Look-up table in Predic

Insighting into first stage (Predic), the key for Predic is looking-up two tables including infopair and infodir. And the principle for looking-up table is described in figure 4.

In Figure 4, top row represents current predicted protein sequence, the ? pointing to the current residue to predict, horizontal axis and vertical axis dis1 and dis2 represent the relative position of other residues from away current residue to predict in the window. For each center residue, local information for both neighboring 8 residues would be considered. Briefly, predictions were done by using a sliding window of a size of width 17 residues, and details can be referred to references[4,5,11].

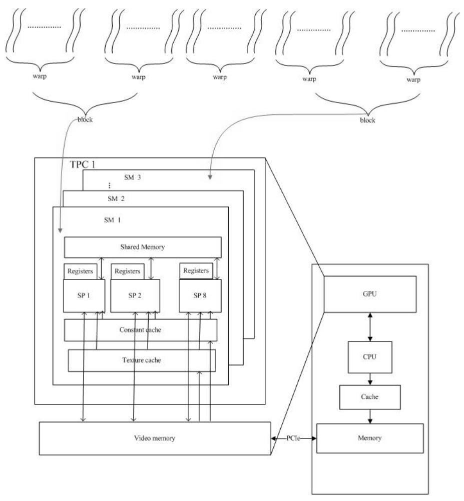

Fig. 3 Collaborative system architecture

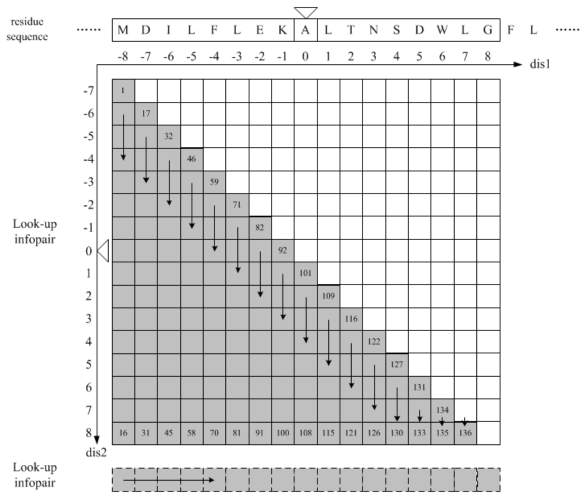

Fig. 4 Principle for look-up table in Predic

4.2 Parallelization

As illustrated in figure 4, only center residue would be calculated in one sliding window, however, every residue in protein sequence should be predicted one after one in the following. Aggressively, we could issue multiple sliding windows, hence, there are multiple center residue would be calculated simultaneously.

More aggressively, we would make hundreds of sliding windows in flight, because there are millions of residues would be predicted in sequence and there are no dependences on prediction and computation among residues to be calculated.

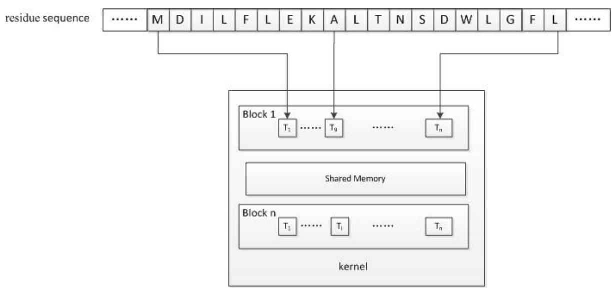

Fig. 5 Parallelizing model for predicting residues

Therefore, each predicting residue could be attached to one CUDA threads, thus, hundreds of predicting residues would be parallelized and calculated simultaneously by many CUDA threads and good-sized threads would group into a thread block, as illustrated in figure 5. Comparing figure 3 and figure 5, it is obvious that parallelizing model we proposed is catering to CUDA execution architecture. Additionally, shared memory is resorted to communication among threads in one thread block, which would avoid frequent global memory or video memory access and improve performance for pixel-level image fusion on CUDA.

4.3 Optimization

CUDA-enabled GPU architecture is memory-bound architecture, so reasonable data layout on CUDA and memory optimization is critical for performance improvement[12, 11].

Usually, source data are transferred from CPU into global memory and CUDA threads would fetch operands frequently from global memory.

During execution, CUDA threads may access data from multiple memory spaces such as registers, local memory, shared memory, constant memory and texture memory except for global memory, as detailed in table 1.

Table 1 Memory Hierarchy for NVidia GTX 280

Memory	Location	Size&Read only	Hit Latency
Shared	on-chip	16KBper SM/No	1 cycle
Constant	on-chip cache	64KB total/Yes	1 cycles
Texture	on-chip cache	up to global/Yes	100 cycles
Global	off-chip	1GB/No	200-300cycles

Consequently, it is unadvisable to store source images into global memory in predicting residues. Practically, Programmers would resort to constant memory or texture memory to cache source data since registers are private by single thread and the number is limited, and shared memory is favor to communicate among threads in one thread block. Hence, constant memory and texture memory are candidates. Due to latency for texture memory is higher than that of constant memory. Hence we employ constant memory to store shared parameter tables. That is because constant memory is low latency, and its space is big enough for shared parameter tables.

5 Experimental Result

5.1 Experiment Setup

In order to validate our proposed methods, we implemented GOR algorithm on CUDA-enabled GPU and CPU respectively, and tested two protein data bases with different scale: *gbenv6.fsa_aa*, *gbconl13.fsa_aa*, downloaded from the NCB I ftp server[14]. And the testing platform is configured as follows.

(1) Intel Core2 Quad 2.33 GHz, 4GB main memory, Microsoft Visual Studio 2005

(2) GeForce GTX280, 1GB video memory or global memory, CUDA toolkit and SDK 2.0 with NVIDIA Driver for Microsoft Windows XP(177.98)

5.2 Performance Comparison

We test proposed techniques using two protein data bases with different scale. Table 2 demonstrates performance comparisons between original GOR-IV version, CUDA version and optimized CUDA version with constant cache.

As shown in Table 2, the running time for protein secondary structure prediction is zooming when protein database is increasing. Accordingly, running time for protein secondary structure prediction would be unbearable with sharp growth in protein database. But execution time is falling sharply when introducing CUDA for parallelizing protein secondary structure prediction. And speedup is increasing when protein database is growing, which is according to CUDA execution model. When testing database is *gbconl13.fsa_aa*, the speedup factor is more than 173X over original GOR-IV version.

Experimental results validate that we propose a fine-grained parallel method to parallelize GOR-IV package for accelerating protein secondary structure prediction. Moreover, we employ constant cache in CUDA to boost performance further.

6 Conclusions

GOR method is one of the earliest and most successful methods in 2D structure prediction. However, predicting protein 2D structure using GOR approach is efficient but time-consuming. Therefore we propose a fine-grained parallel method to parallelize GOR-IV package for accelerating protein secondary structure prediction, in which each amino acid would be assigned to one single CUDA thread, hence protein secondary structure prediction would be parallelized by many CUDA threads simultaneously, Moreover, we employ constant cache in CUDA to boost performance further. Eventually, CUDA version with optimization could attain more than 173X speedup than original GOR-IV version.

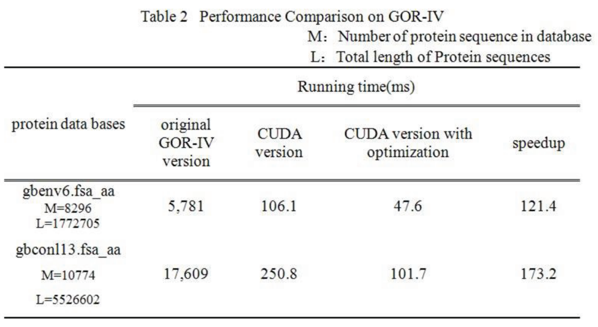
Table 2 Performance Comparison on GOR-IV

M: Number of protein sequence in database

L: Total length of Protein sequences

protein data bases	Running time(ms)			
	original GOR-IV version	CUDA version	CUDA version with optimization	speedup
gbenv6.fsa_aa M=8296 L=1772705	5,781	106.1	47.6	121.4
gbconl13.fsa_aa M=10774 L=5526602	17,609	250.8	101.7	173.2

Acknowledgement

This work is partly supported by 863 Program Foundation of China under Grant No. 2012AA010905, National Natural Science Foundation of China under Grant No. 60803041, No. 61272144, No. 61070037, No. 61272143, No. 60970033 and No. 61025009.

The authors are grateful to the anonymous referee for a careful checking of the details and for helpful comments that improved this paper.

References

[1] Sanchez R, Sali A. Corporative protein structure modeling in genomics. Journal of Compute Physic 1999, 151:388-401.

[2] Jones DT, Taylor WR, Thornton JM. A new approach to protein fold recognition. Nature 1992, 358:86-89.

[3] B.W J, E.B D. An evolutionary approach to folding small alpha-helical proteins that uses sequence information and an empirical guiding fitness function. Proceedings of the National Academy of Sciences of the United States of America 1994, 91:4436-4440

[4] Garnier J, Osguthorpe DJ, Robson B. Analysis and implications of simple methods for predicting the secondary structure of globular proteins. J. Mol. Biol 1978, 120:97-120.

[5] Garnier J, Gibrat JF, Robson B. GOR method for predicting protein secondary structure from amino acid sequence. Methods Enzymol 1996, 266:540-553.

[6] Dill KA. Principles of protein folding: a perspective from simple exact models. Protein Sci 1995, 4:561-602.

[7] Cuff JA, Barton GJ: Application of enhanced multiple sequence alignment profiles to improve protein secondary structure. Proteins: Struct. Funct.Genet 2000, 40:502-511.

[8] King RD, Sternberg MJE. A machine learning approach for the prediction of protein secondary structure. J. Mol. Biol 1990, 216:441-457.

[9] Xia et al. FPGA accelerator for protein secondary structure prediction based on the GOR algorithm BMC Bioinformatics 2011, 12(Suppl 1):S5.

[10] NVIDIA. NVIDIA CUDA C Programming Guide 3.1. NVIDIA Corporation, 2010.

[11] Binod Kumar and N. N. Jani. Prediction of Protein Secondary Structure based on GOR Algorithm Integrating with Multiple Sequences Alignment. International Journal of Advanced Engineering & Applications, 2010, Jan, 107-112.

[12] Ryoo S, Rodrigues C.I, Baghsorkhi S.S, etc. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. // proceeding of ACM SIGPLAN Symposium on Principles and practice of parallel programming, 2008, 73-82.

[13] Xinbiao Gan, Li Shen, Zhiying Wang, etc. Data layout pruning on GPU. Applied mathematics & information science, 2011,5 (2):129S-138S.

[14] ftp://ftp.ncbi.nih.gov/ncbi-asn1/protein_fasta.

Xinbiao Gan received MS degree in computer system architecture from National University of Defense Technology of China in 2008. He is currently pursuing PhD degree in computer system architecture from National University of Defense Technology of China.

His research interests include High performance computing, Computer architecture, GPGPU and Compiler Optimization.