EPiC Series in Computing

Volume 70, 2020, Pages 109–118

Proceedings of the 12th International Conference on Bioinformatics and Computational Biology

Graphical Processing Unit - Supported RNA Secondary Structure Comparison

Mutlu Mete and Abdullah N. Arslan

Department of Computer Science and Information Systems
Texas A & M University - Commerce, Commerce, TX 75428, USA
{Mutlu.Mete, Abdullah.Arslan}@tamuc.edu

Abstract

This study is part of our perpetual effort to develop improved RNA secondary structure analysis tools and databases. In this work we present a new Graphical Processing Unit (GPU)-based RNA structural analysis framework that supports fast multiple RNA secondary structure comparison for very large databases. A search-based secondary structure comparison algorithm deployed in RNASSAC website helps bioinformaticians find common RNA substructures from the underlying database. The algorithm performs two levels of binary searches on the database. Its time requirement is affected by the database size. Experiments on the RNASSAC website show that the algorithm takes seconds for a database of 4,666 RNAs. For example, it takes about 4.4 sec for comparing 25 RNAs from this database. In another case, when many non-overlapping common substructures are desired, a heuristic approach requires as long as 85 sec in comparing 40 RNAs from the same database. The comparisons by this sequential algorithm takes at least 50% more time when RNAs are compared from the database of several millions of RNAs. The most recently curated databases already have millions of RNA secondary structures. The improvement in run-time performance of comparison algorithms is necessary. This study present a GPU-based RNA substructure comparison algorithm with which running time for multiple RNA secondary structures remains feasible for large databases. Our new parallel algorithm is 12 times faster than the CPU version (sequential) comparison algorithm of the RNASSAC website. The response time significantly reduces towards development of a realtime RNA comparison web service for bioinformatics community.

1 Introduction

Ribonucleic acid (RNA) substructures are described by context free grammars and stochastic context free grammars. RNA secondary structures can be represented by sequences in various formats (e.g., dot-bracket, bpseq). In this paper, we present an improved algorithm for multiple RNA structure comparison. As the number of compared RNAs increases, the significance of found common substructures also increases. Yet, the computing time also increases with the increasing number of compared RNAs.

Our algorithm benefits from a recently proposed RNA sequence representation in which 2D structural information is embedded in sequences with desirable features [1]. This new

Q. Ding, O. Eulenstein and H. Al-Mubaid (eds.), BICOB 2020 (EPiC Series in Computing, vol. 70), pp. 109–118

representation uses relative addressing between the pairs. As given in [1], it has many important advantages with which RNA secondary substructure search is reduced to a substring search in a single RNA or in a database of RNAs stored appropriately in a suffix array. *RNA Search Submit Annotate Compare* (RNASSAC) website (Figure 1) hosts a database of structure sequences for 4,666 RNAs obtained from RNASTRAND [2]. This website employs several tools for RNA secondary structure analysis. It includes a search-based algorithm for multiple RNA structure comparison problem proposed in [3]. This comparison algorithm takes several seconds for

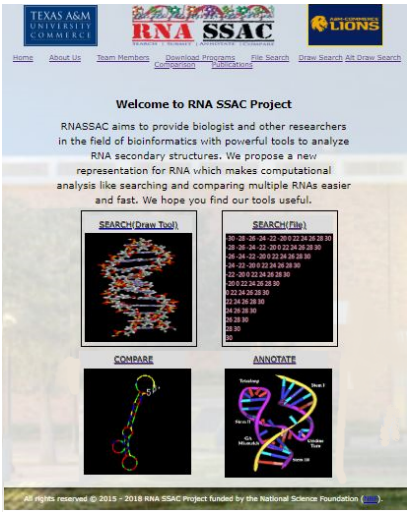

Figure 1: RNASSAC Website, linked from http://faculty.tamuc.edu/aarslan

comparing multiple RNAs in the database of 4,666 RNAs. A similar search-based heuristic algorithm for finding many non-overlapping common structures takes a few minutes on the same database [3]. When comparing many RNAs from today's much larger databases, the execution time takes 50% or higher more time. The required large computing effort and long response times are still tolerated by users because the structure analysis on a larger database yields biologically more significant results carrying more valuable information. However, for RNA structure analysis tools to be more practical and more feasible (in terms of response times) for very large databases, the execution times need to be reduced. By using efficient GPU-supported algorithms we decrease the time-requirement of analysis steps on suffix arrays that become large with increased database sizes. By following statistics from [4], for instance, we can easily estimate that the total number of RNAs is at least thousands of times larger than 4,666. In our perpetual effort to develop and improve RNA secondary structure analysis tools for large databases, we present our new multiple RNA secondary structure comparison algorithm and initial experimental results in this work.

1.1 Multicore Computing

Computational research has benefited from scientific software toolkits utilizing central processing units (CPU) as well as graphics processing units (GPU). In last two decades, affordable CPU-GPU systems, introduced by industry leaders NVIDIA, AMD, and Intel, have been vastly integrated many biomedical research studies. GPU-supported technology has become computational norm providing powerful computation with the ease of flexible programming on multi-core systems. General purpose GPU (GPGPU) technology involves one or a group GPUs to perform

computational problems traditionally developed for single thread executions on CPU. Furthermore, GPGPU programming provides the most economical way for parallel computing power. However, this advancement comes with a price: extra time for software developers to parallelize underlying CPU-based algorithms. The algorithm being parallelized should be modified considering hardware specification of the target GPU to have the best speedup results. To address this issue, NVIDIA introduced Compute Unified Device Architecture (CUDA) in 2007 allowing software developers to harness the computational capabilities of powerful graphical processing, initially designed for video and image-processing tasks [5].

Recent studies in biomedical fields have proved that time-consuming computations can be successfully accelerated up to 55 times [6, 7, 8, 9, 10]. RNA related bioinformatics problems also had their share from GPU-based accelerated research. Li et al. [11] reported that the multicore version of the cache efficient single core Nussinov RNA folding algorithm runs 1582 times as fast as the naive single core algorithm. Another RNA folding study using the Four-Russians method reported [12] that their method is faster than Nussinov by up to a factor of 20 and the Frid-Gusfield method by a factor of 3. In a recent study [13], authors reported a tile-based approximate pattern matching implementation on a NVIDIA GeForce Titan graphics card runs up to 2.9 times faster than an implementation on Intel Xeon Phi 5110P.

This paper is part of our effort to develop a new GPU-based RNA structural analysis framework that supports fast multiple RNA secondary structure comparison for very large databases. Specifically, we parallelize the search-based RNA secondary comparison algorithm. A novel GPU-based algorithm is developed and tested on known RNA structures. The run-time efficiency of new software is recorded as 12 fold faster than its CPU counterpart.

The outline of this paper is the following: In Section 2, the primary motivation of this study is presented: GPU-supported improved structure comparison algorithms for the fast growing databases for RNA secondary structures. We give basic definitions and notation we use in Section 3. We also briefly describe the sequential multiple RNA secondary structure comparison algorithm [3] in the same section. We present our GPU-based algorithm for comparison in Section 4. We also report our preliminary test results in this section. We summarize our results and include our remarks in Section 5.

2 RNA Secondary Structure Databases & GPU-Support

Biomedical data are growing at accelerated pace. Cell biology databases take their share, too. There are various websites for RNA structure analysis. For example, Gutell's laboratory at University of Texas [14] has an RNA database and tools for RNA analysis for ribosomal, intron, and other RNAs. Another resource, RNA Fragment Search Engine [15], is owned and maintained by Polish Academy of Sciences. This system has a database of RNA fragments that can be searched online. Finally, the RNA Central portal [4] has one of the largest databases on RNA secondary structure diagrams. Currently, it is claimed that more than seven million structure diagrams (with general consensus) for over 2,000 RNA families are hosted from 35 different sources including Rfam, PDBe, and NCBI. It is expected that the RNA structure database sizes will continue to increase further by including predicted structures with consensus using structure prediction tools, such as [16, 17]. The resulting structure databases will make the RNA structure analysis near perfect at the cost of increases in the database sizes.

2.1 Motivation for GPUs for RNA Search & Comparison

RNASSAC website has a database stored in a suffix array which is used by its search and comparison tools. Each RNA is a string in *relative addressing based* (*rab*) format [1]. Let R be the number of RNAs in the database, the RNASSAC website builds a suffix array that contains n sorted suffixes. Each row contains a pair of indices for the corresponding suffix: one for an RNA ID, and another one for the starting position for a suffix in this RNA. Suffix array creation is designed to be a batch operation done only in long periods, such as monthly updated. In this framework, substructure search is a substring search operation. Let $|u|$ denote the length of any given string u. Searching a substructure represented by string s takes time $O(|s| \log n + c)$, where c is the number of occurrences of s in the database. One of the most important parts of the RNASSAC project is a comparison tool that finds the longest substructure included in all RNAs in the list. The list contains RNAs from the database, and optionally one additional RNA not included in the database. We call this problem *FindOne*, which is search-based comparison algorithm and discussed with great details in [18, 3]. *FindOne* runs in time $O((N+\ell)\ell \log n)$, where ℓ is the length of the shortest of the N RNA strings compared. Problem *FindOne* aims to find a largest substructure (longest common substring) in compared RNA substructure. Another related problem, called *FindAll*, aims to find all common non-overlapping substructures. A heuristic algorithm for *FindAll* proposed in [3] uses the algorithm for *FindOne* as a subroutine, and requires ℓ times more time than that of the algorithm for *FindOne*.

The complexity upper bounds reported for search and comparison algorithms in [1, 3] are tight. A suffix array stores rows for all suffixes of given strings. Therefore for a database that contains R RNAs, the suffix array that *RNASSAC* builds contains n rows such that n is approximately ℓR, where ℓ is the average length of an RNA. The time complexity of search and comparisons algorithms increase logarithmically in n. Based on these factors, we consider the time requirement changes for larger databases. *RNASSAC* has a suffix array built from a set of 4,666 RNAs. Substructure search takes milliseconds. The times reported on comparisons are in seconds. For example, on the same database, the algorithm for *FindOne* takes 4.4 sec for comparing 25 RNAs, and the algorithm for *FindAll* takes 85 sec in comparing 40 RNAs. We calculate by the complexity results that for a seven million-RNA database, the time requirement for these algorithms will be 50% more. That is for these two instances, the running times will be approximately 6.6 sec and 127.5 sec. As complexity results indicate, comparisons of substructures for many similar RNAs from the same family are expected to take even longer times. We propose developing new parallel algorithms for comparison algorithms by using GPUs. In this work we present a new such algorithm for *FindOne*, which will improve the response time for *FindAll* that uses *FinOne* algorithm as a subroutine. These improvements will improve the response time for both comparison problems.

3 Notations and Definitions

RNA sequences are strings over the alphabet $\{A, C, G, U\}$. We use RNA sequence and RNA string, RNA subsequence and substrings interchangeably. In an RNA molecule, some nucleotides may be paired. This base-paired structure is called the *secondary structure* of RNA. A secondary structure contains substructure-types, such as a hairpin-loop, helix, and pseudoknot. An RNA nucleotide sequence in the secondary structure should be read from terminal $5'$ to $3'$. Similar substructures in RNA molecules point to similar functions. A *dot-parenthesis* [2] representation of a structure includes pairing information in a sequence format. It uses the alphabet

of {(, . ,)}. Matching pairs of parentheses indicate bonds. Everywhere else '.' represents any of the nucleotides.

We define RNA structure sequence as a string that uniquely identifies an RNA secondary structure. Sequences in *bpseq* [2] and *rab* format [1] have this property. The *relative addressing based* (*rab*) format was introduced in [3]. Figure 2 demonstrates an example structure and its representation in both *bpseq* and *rab* formats. The *rab* representation has two options: structure-only and structure-nucleotide representation. In structure-only representation nucleotide information is not used. In the structure-only option, for example, the *rab* representation of the structure of Figure 2 is the sequence +6, +4, 0, 0, 0, −4, −6, 0, 0, +6, +4, 0, 0, 0, −4, −6. In the actual internal representation for each position, there is an integer value obtained from the offset (distance) from the pairing position and the nucleotide type. That means *rab* format stores sequences of integers of fixed size, such as four bytes. One big advantage of the *rab* format is that the same (consecutive) substructure sequence appears as the same unique string no matter where it appears in the host RNA. In Figure 3, sample comparison results for three RNAs are shown. Only one of these RNAs is displayed. The longest common secondary substructure for all these RNAs is highlighted in this figure. The positions of this common secondary structure are listed for all RNAs in the figure.

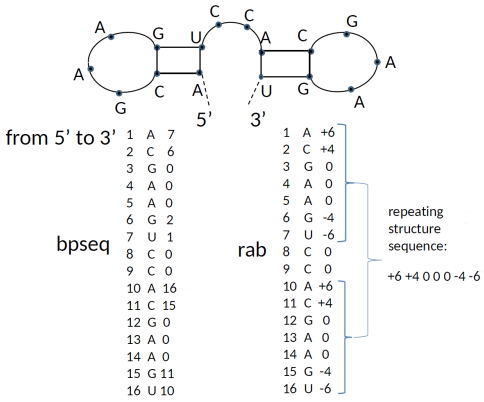

Figure 2: Illustration of one significant advantage of *rab* over *bpseq*.

This is not the case for other existing structure sequence representations, such as *bpseq*. The example in Figure 2 illustrates this very useful property of *rab* representation. By using this very property, the substructure search reduces to the substring search problem. The RNASSAC project stores RNA structure sequences in a suffix array, and substructure search can be done efficiently by performing binary search on this suffix array. Our GPU-based RNA structure comparison algorithm uses this property, too.

Definition 1 (FindOne: RNA-Longest Common Substructure) Given a collection of N RNA structure strings $A = \{r_1, \ldots, r_N\}$ where each r_i represents an RNA secondary structure sequence from the underlying database, find a longest substring (substructure) s that is a common substring of all strings r_i in A.

The comparison algorithm [3] of *RNASSAC* website takes the shortest structure sequence from the input list of RNAs and considers every substring of this RNA structure sequence, and determines if it is contained in structure sequences in all RNAs of the input list. This is done by using two nested binary searches: the outer one for the maximal length, and the inner one for determining if the substring considered is common to all RNAs in the input list. The inner binary search performs the search on the suffix array for a taken substring from the shortest substring of length given from the outer binary search. This search, if successful, returns a

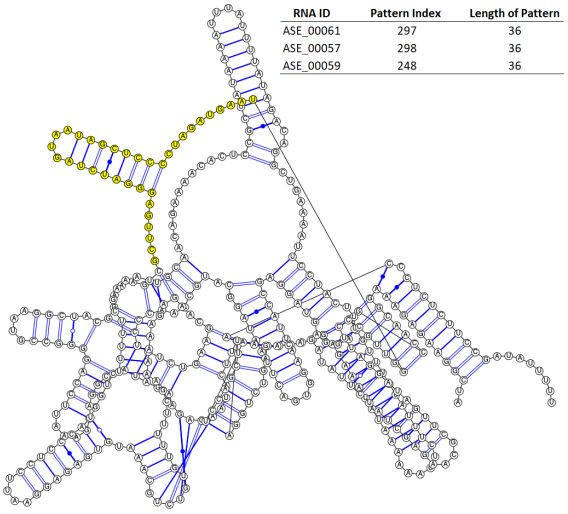

Figure 3: Sample results for three compared RNAs

segment from the suffix array. If this segment contains a list of RNA IDs in list A, then the length-range is increased by the outer binary search. If the attempts for the current length in the inner binary search is not successful, then the length-range is decreased by the outer binary search for maximal common substring length. These two nested searches yield many substring searches on the suffix array. Our GPU algorithm parallelizes these stream of searches.

4 GPU-Based Multiple RNA Structure Comparison

CUDA is the hardware and software architecture that enables NVIDIA GPUs to execute programs written with C, C++, Fortran, and other languages [5]. A GPU program calls parallel kernels, which are called by a CPU (host) originated program. GPUs use the single instruction multiple thread (SIMT) programming model in which the GPU accomplishes a computational task using thousands of light weight threads. The CUDA compiler organizes these threads in grids of thread blocks. Each thread within a thread block executes an instance of the same kernel, and has a thread ID within its thread block, program counter, registers, per-thread private memory, inputs, and output results. A thread block is a set of concurrently executing threads that can cooperate among themselves through barrier synchronization and shared memory. Each thread block has a block ID within its grid so that the programmer can tweak execution of all of thread in that block.

4.1 RNA Structure Comparison Algorithm in GPU

In GPU-based implementation of the *FindOne* problem, we adopt master-slave programming model in which we write a program for the master computer that invokes the GPU-targeted procedures. The key challenge in deriving high performance for the algorithm is efficiently to minimize the memory traffic between the streaming multiprocessor and the global memory of the GPU. This requires redesigning of the algorithms and implementation approaches, which are the two main aims of this project. We developed the comparison framework with Anaconda Python [19].

The pseudocode of proposed algorithm is given in Algorithm 1. It consists of two procedures, *ParRNASSAC* at line 1 and *KERNEL* at line 16. *ParRNASSAC* serves as starting point of the algorithm and will be run on host system, i.e., CPU. It takes three arguments, two from system resources, one from the user. D is a two-dimensional array of RNA structures as coded in [1], SA is the suffix array, which is derived from whole RNA data set. The formation of SA also was detailed in [1]. TA is set of user selected RNA structures, in which the search and comparison algorithm will search for the longest common substring. We first identified IDs of shortest RNA structure, sR in length. This is because TA cannot have a longest common substring longer than length of sR. Based on the sR, $maxL$ is length of $D[sR]$. Length of the shortest RNA can be also found in GPU kernel but would be costly since clock rate of CPU usually –*two times* faster than the GPU's clock rate. Following finding of $maxL$, we call kernel function, $KERNEL$.

$KERNEL$ is a kernel function to be called by the host machine once for each user query. It takes five pass-by-reference parameters. Three of five parameters, D, SA, and TA are also parameters of main *ParRNASSAC* procedure, and already discussed in previous paragraph. The other two of them, sR and $maxL$ were calculated based on D, SA, and TA in *ParRNASSAC*. The sR is ID of so-called source RNA, the shortest RNA in TA, whose substrings will be generated to be sought in other RNAs of TA. It is not explicitly shown in the pseudocode; however, all data points –*five parameters* are transferred from host system to GPU device before running $KERNEL$. The kernel function starts with finding the thread specific IDs at line 17. The number of threads is $maxL$.

Then, we allocate a shared array CS in GPU with $maxL$ items. CS, being a shared memory array, is much faster to access for all threads than any of the memory location in GPU. The size of CS is $maxL$ since we run $maxL$ threads. The each thread tx accesses and updates $CS[tx]$ later in this procedure. The goal of the kernel is to run $maxL$ threads in parallel and let each thread do the following in order a) Search for longer than tx substrings in length in the set of query RNA, that is TA, b) Register the outcome of the search in $CS[tx]$.

The first loop iterates for each entry of suffix array, SA, which is an array structure (programming language structures, such as *struct* in C), with two variables: *ID* and *str*. *ID* of each entry E of SA is same as ID of RNA structure database, D. The second variable, *str* keeps starting location of suffix in the corresponding RNA. We check later if this suffix is part of sR and is longer than or equals to tx. If not, the search is skipped since the current thread, identified by tx, search only longer than tx substrings in the TA. If line 20 evaluates *true* then we create a local variable for the search string $sSTR$ in the memory and fill it with in the subsequent for loop. Once a thread reaches line 25, it iterates for all RNA IDs of TA, except sR, since sR is source RNA. Once G is eligible RNA ID, we check all of its suffixes using SA. Before that we set a boolean flag, *failed* to false, so that we can break the for-loop of line 31 if substring matching is failed. To reduce computing time further, we have one more conditional check at line 30. This is to exit the outer loop where the thread cannot find long enough substring, coded with Gs in pseudocode, to compare with $sSTR$. If a thread finds $sSTR$ in an RNA without setting *failed* true, this constitutes enough condition to increase its counter by one. Therefore, we check and accumulate number of successful searches of tx in $CS[tx]$ at line 40. The last part of $KERNEL$ at line 47 checks for whether or not all searches in TA find the common substring $sSTR$. Since we skip to search in sR, the counter at $CS[tx]$ returns the size of TA less one. If this holds true, we set this counter to true. When all threads complete their relative calculations, $KERNEL$ returns CS, which is also pass-by-reference operation. The procedure *ParRNASSAC* later iterates over RCS and reports longest common substring found in the set of TA. This GPU-based algorithm is developed only for $FindOne$.

Algorithm 1 Parallel RNASSAC

```
1: procedure ParRNASSAC(D, SA, TA)
2:     sR ← 0                                              ▷ the index of shortest RNA
3:     for v ← 1, |TA| do
4:         if D[TA[v]] < |D[sR]| then
5:             sR ← v
6:         end if
7:     end for
8:     maxL ← |D[sR]|
9:     RCS ← Kernel(D, sR, maxL, SA, TA)
10:    for i ← 0, maxL do
11:        if RCS[maxL − v] = true then
12:            return i
13:        end if
14:    end for
15: end procedure

16: procedure Kernel(D, sR, maxL, SA, TA)
17:    tx ← cuda.gpu()                                              ▷ Thread ID
18:    alloc CS                                  ▷ cudaShared array |CS| = maxL
19:    for all E ∈ SA do
20:        if E.ID = sR and D[E.ID] − E.str − 1 ≥ tx then
21:            init sSTR                                        ▷ array of search str
22:            for j ← 0, tx do
23:                sSTR[j] ← D[sR][E.Str + j]                   ▷ slicing operation
24:            end for
25:            for all G ∈ TA do
26:                if G ≠ sR then
27:                    for all Gs ∈ SA do
28:                        failed ← false
29:                        if Gs.ID = G then
30:                            if D[Gs.ID] − Gs.str − 1 ≥ tx then
31:                                for k ← 0, tx do
32:                                    if sSTR[k] ≠ D[Gs.ID][Gs.str + k] then
33:                                        failed ← true        ▷ substring matching is failed
34:                                        break
35:                                    end if
36:                                end for
37:                            end if
38:                        end if
39:                        if failed = false then
40:                            CS[tx] ← CS[tx] + 1   ▷ accumulate number of successful searches
41:                        end if
42:                    end for
43:                end if
44:            end for
45:        end if
46:    end for
47:    if CS[tx] = |TA| − 1 then
48:        CS[tx] ← true
49:    else
50:        CS[tx] ← false
51:    end if
52:    return CS
53: end procedure
```

4.2 Experimental Setup and Results

Experimental software was developed with Python v3.6 on HP Z400 workstation equipped with Intel 2.7 GHz Xeon processors, 24 GB RAM, and NVidia Quadro FX 1800. The primary library used in our software is Numba 1.7 [19]. The RNA dataset is downloaded from the *RNASSAC* website [2] in *rab* format which was converted from 4,666 structure sequences originally obtained from *RNASTRAND* database in *bpseq* format. The average length of RNA structures is 511 characters long. We randomly choose 2 to 10 structures incrementally and search for the longest common substring in chosen structures. This process is repeated 20 times with various size of the substrings. Correctness of GPU version of the algorithms is validated with results obtained from serial (CPU) version of the algorithms. Figure 4 compares running time of this comparison.

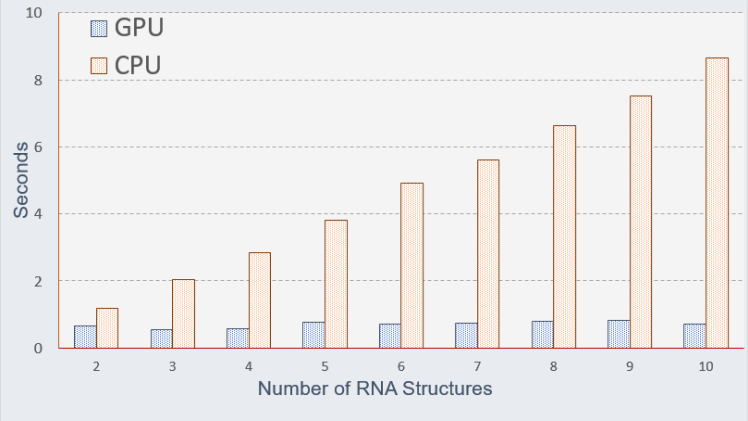

Figure 4: Running time comparison of CPU- and GPU-based implementation of RNASSAC algorithm for different size of RNA structures.

As expected GPU version of the algorithm showed almost stable running time with 0.70 sec average and 0.09 standard deviation. The slope of line regression is 0.02. CPU-based running time is linear with the slope of 0.92, and $R^2 = 0.998$. These results were expected since the running time complexity of RNASSAC is $O((N + \ell)\ell \log n)$ [18], where ℓ is number of RNA structure being compared. The speedup in case of substring search in 10 structures was reported as 12.29. In summary, results show that running time complexity of substring search and comparison significantly benefit from GPU-based parallelization.

5 Conclusion and Future Works

We develop a GPU based algorithm for search-based comparison of multiple RNA substructure sequences for finding the largest common substructures. Our results indicate that our algorithm speeds-up the sequential programs for the same task by 12 times or better. The same speed-up can be observed for the comparison algorithm that finds all non-overlapping common structures since this is achieved by calling our algorithm as a subroutine in the same number of times both in sequential or parallel versions. These are important achievements for making and keeping such comparisons feasible in ever-increasing large RNA secondary structures databases. We believe that our RNA secondary structure comparison tools will be very useful for the bioinformatics community. Our findings in developing GPU-based implementations for multiple RNA structure comparison will be useful in other related data, text mining applications, and in particular, in applications where suffix arrays are used.

References

[1] Abdullah N Arslan, Jithendar Anandan, Eric Fry, Rabindra Pandey, and Keith Monschke. A new structure representation for rna and fast rna substructure search. In *IEEE Computational Science and Computational Intelligence*, pages 1226–1231, 2016.

[2] Mirela Andronescu, Vera Bereg, Holger H Hoos, and Anne Condon. Rna strand: the rna secondary structure and statistical analysis database. *BMC bioinformatics*, 9(1):340, 2008.

[3] Abdullah N. Arslan, Jithendar Anandan, Eric Fry, Keith Monschke, Nitin Ganneboina, and Jason Bowerman. Efficient rna structure comparison algorithms. *Journal of Bioinformatics and Computational Biology*, 15(06):1740009, 2017.

[4] Rna central, Nov 2019. Available at https://www.rnacentral.org.

[5] CUDA Nvidia. Nvidia cuda c programming guide. *Nvidia Corporation*, 120(18):8, 2018.

[6] Lars Wienbrandt, Jan Christian Kassens, Matthias Hübenthal, and David Ellinghaus. Fast genome-wide third-order snp interaction tests with information gain on a low-cost heterogeneous parallel fpga-gpu computing architecture. *Procedia Computer Science*, 108:596–605, 2017.

[7] Devrim Akgün, Ünal Sakoğlu, Johnny Esquivel, Bryon Adinoff, and Mutlu Mete. Gpu accelerated dynamic functional connectivity analysis for functional mri data. *Computerized Medical Imaging and Graphics*, 43:53–63, 2015.

[8] Jorge González-Domínguez, Jan Christian Kässens, Lars Wienbrandt, and Bertil Schmidt. Large-scale genome-wide association studies on a gpu cluster using a cuda-accelerated pgas programming model. *The Inter. J. of High Performance Computing Applications*, 29(4):506–510, 2015.

[9] Matija Korpar and Mile Šikić. Gpu-enabled exact alignments on genome scale. *Bioinformatics*, 29(19):2494–2495, 2013.

[10] Shreyank Periyapatna Ramesh. *Density based visualization of big data with Graphical Processing Units*. Texas A&M University-Commerce, 2014.

[11] Junjie Li, Sanjay Ranka, and Sartaj Sahni. Multicore and gpu algorithms for nussinov rna folding. *BMC bioinformatics*, 15(8):S1, 2014.

[12] Balaji Venkatachalam, Dan Gusfield, and Yelena Frid. Faster algorithms for rna-folding using the four-russians method. *Algorithms for Molecular Biology*, 9(1):5, 2014.

[13] Tuan Tu Tran, Yongchao Liu, and Bertil Schmidt. Bit-parallel approximate pattern matching: Kepler gpu versus xeon phi. *Parallel Computing*, 54:128–138, 2016.

[14] Jamie J Cannone, Sankar Subramanian, Murray N Schnare, James R Collett, Lisa M D'Souza, Yushi Du, Brian Feng, Nan Lin, Lakshmi V Madabusi, Kirsten M Müller, et al. The comparative rna web (crw) site: an online database of comparative sequence and structure information for ribosomal, intron, and other rnas. *BMC bioinformatics*, 3(1):2, 2002. Available at http://www.rna.ccbb.utexas.edu/.

[15] Mariusz Popenda, Marta Szachniuk, Marek Blazewicz, Szymon Wasik, Edmund K Burke, Jacek Blazewicz, and Ryszard W Adamiak. Rna frabase 2.0: an advanced web-accessible database with the capacity to search the three-dimensional fragments within rna structures. *BMC bioinformatics*, 11(1):231, 2010. Available at http://rnafrabase.ibch.poznan.pl/.

[16] David H Mathews, Douglas H Turner, and Richard M Watson. Rna secondary structure prediction. *Current protocols in nucleic acid chemistry*, pages 11–2, 2007.

[17] Ivo L Hofacker. Vienna rna secondary structure server. *Nucleic acids research*, 31(13):3429–3431, 2003.

[18] Jithendar Anandan, Eric Fry, Keith Monschke, and Abdullah Arslan. A fast algorithm for finding largest common substructures in multiple rnas. In *9th International Conference on Bioinformatics and Computational Biology*, pages 51–57, Mar 2017.

[19] Numba, Nov 2019. Available at https://numba.pydata.org/.