Dynamic Programming in RNA Folding Software and Improvements to the Current Regime

Jianglin Liu, Bingxi Li, and Gregory Rehm

University of California, Davis {jiliu, bxli, grehm}@ucdavis.edu

November 20, 2015

Abstract

Finding the secondary structure of RNA is important for understanding how RNA will interact in a cell. Frequently computational algorithms are used to determine structure due to difficulties extracting good *in vivo* data of RNA structures. Many of the algorithms for RNA folding are computationally complex. In this paper we establish benchmarks for two commonly used RNA folding packages, mfold, and RNAFold Vienna, and compare them to an improved Four-Russians folding algorithm. We show ... These results will help software maintainers to understand the benefit of updating their algorithms. The results will also help guide users when choosing RNA folding software when looking for the most computationally optimal package.

1 Introduction

RNA is an essential macromolecule used in protein formation and performs other essential functions within the body[4]. RNA does not stay in single stranded form and instead folds on itself to create the lowest energy conformation possible to ensure thermodynamic stability[6]. When folding, RNA forms a 2D secondary structure[5] with A matching to U and G to C (figure 1). Using this data we can find a 3D tertiary structure[5].

Our paper focuses on benchmarking 2D secondary structure prediction software based off the Nussinov dynamic programming algorithm[3] for RNA folding. This algorithm is $O(n^3)$ time complexity. There have been multiple attempts to parallelize the Nussinov algorithm[!CITE!] which have resulted in large speed increases, however, CPU intensive algorithms only slightly lowered the bound

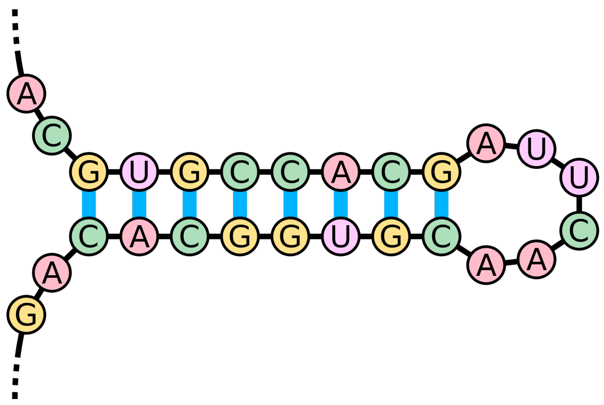

Figure 1: Example of an RNA molecule folding

up until 2011[!CITE!]. In 2011 the Nussinov bound that was improved by the Frid-Gusfield Four Russians method which established you could perform the DP method in $O(\frac{n^3}{log(n)})$ time[1]. Later a parallel method of the Four Russians algorithm presented proof that you can lower this bound to $O(\frac{n^2}{log(n)})$ inside an NVIDIA CUDA environment[2].

There are two major RNA folding software packages, mfold[8, 9], and the Vienna RNA Package[7]. These both utilize the Nussimov method to return results of the RNA secondary structure by finding the lowest possible thermodynamic conformations of the RNA[9, 7]. In order to tell which was faster we performed application level benchmarks[11] to see which of these two applications could more quickly fold arbitrary length RNA of the circular and linear variety. We then performed micro-benchmarks[10] of the folding algorithms utilized in mfold and Vienna for both circular and linear RNA, and then compared that to equivalent benchmarks for the Four Russians and parallel Four Russians algorithms. The first thing we found when comparing mfold and Vienna was ... . When we compared mfold and Vienna to the Four Russians algorithms we found that without a graphics card and parallel processing capabilities a machine can see speed ups of ... in mfold and ... in Vienna for linear RNA. For circular RNA speed increases change to ... in mfold and ... in Vienna. With a graphics card and parallel processing folding clusters can see increases of up to ... for mfold and ... in Vienna. Circular RNA also sees speed bumps of ... in mfold and ... in Vienna.

These results show us that ... for natively differentiating between mfold and Vienna. They also help explain that both pieces of software can experience significant speed increases if they implemented the Frid-Gusfield method. Furthermore the authors of this paper would recommend both software packages to support

GPU hardware to achieve even greater speed gains when inside a parallel capable environment.

2 Methods

Benchmarking is renown as a difficult thing to perform effectively[10, 11]. There are so many things that can be going on in a computer at any one moment that it is completely possible that a benchmark can give inaccurate information due to a conflicting process running on the computer[10]. As a result we used a machine solely dedicated for benchmarking and no other tasks. We also standardized on the following specifications for our runs:

Architecture	**Operating System**	**Compiler**
blah	Linux	GCC xx
”	”	Fortran 77
”	”	python3
”	”	NVIDIA CUDA

We used the following applications with corresponding versions and requirements in our test runs:

Application	**Version**	**Requirements**
mfold	3.6	GCC, Fortran
RNAfold Vienna	xxx	GCC
Frid-Gusfield Four Russians	N/A	python3 ??
Parallel Four Russians	N/A	GCC, NVIDIA CUDA

Our testing architecture was laid out where we would SSH into the benchmarking machine and then execute tests. Test results would then be reported back to the user's central machine where they could be stored in a database for later analysis (figure 2). Our testing required no internet connectivity besides the ssh access required to initiate our testing so all calculations were performed locally. Furthermore there were few I/O read/writes required as we will show so there is very little chance that I/O was a factor on our results.

2.1 Micro-benchmarks

The easiest type of benchmark to perform is the micro-benchmark. The micro-benchmark is a single piece of code executed many times in serial so that we can get a good profile of its run characteristics. Once these characteristics are observed we can then make inferences about its performance and ways that it

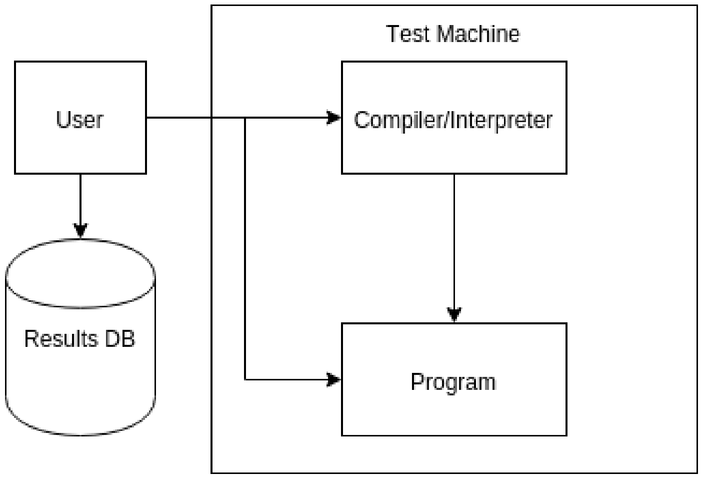

Figure 2: Test Architecture

can be improved. Micro-benchmarks have the downside of losing generality of performance across the entire application[11]. A good example of this is if an IO heavy function made many consecutive calls to the *read* function on the OS while the rest of the application made no calls to *read* whatsoever. So if we tried to generalize this one function to the rest of the application we would misguidedly attempt to optimize disk IO across our entire system.

We avoid this trap in our paper by benchmarking only parts of the code that execute the Nussinov algorithm in mfold and the Vienna package. We then

2.2 Application Benchmarks

One type of benchmark we can perform is an application level benchmark. Applications generally are composed of many different modules an processes and sometime

Micro-benchmarking is the process of running small pieces of code and then testing the amount of time it takes to execute. These tests are performed many times to return some kind of statistical distribution of run times and other relevant information.

3 Results

4 Discussion

References

[1] Frid Y, Gusfield D. *A simple, practical and complete $O(n^3)$-time algorithm for RNA folding using the Four-Russians Speedup*. Algorithms Mol Biol 2010, 5:13

[2] Venkatachalam B, Gusfield D. *Faster algorithms for RNA-folding using the Four-Russians method*. Algorithms for Molecular Biology20149:5.

[3] Nussinov R, Jacobson A. *Fast algorithm for predicting the secondary structure of single-stranded RNA*. Proc. Nati. Acad. Sci. USA Vol. 77, No. 11, pp. 6309-6313, November 1980.

[4] Mathews D, Turner D. *Prediction of RNA secondary structure by free energy minimization*. Current Opinion in Structural Biology 2006, 16:270–278.

[5] McCaskill J.S. *The Equilibrium Partition Function and Base Pair Binding Probabilities for RNA Secondary Structure.* Biopolymers, Vol. 29,1105-1119 (1990)

[6] Herschlag D. *RNA Chaperones and the RNA Folding Problem.* Vol. 270, No. 36, Issue of September 8, pp. 20871–20874, 1995

[7] Hofacker I. L, Fontana W, Stadler P. F, Bonhoeffer L. S, Tacker M, Schuster P. *Fast Folding and Comparison of RNA Secondary Structures*. Monatshefte ftir Chemie 125, 167-188 (1994)

[8] Zuker M. *Computer Prediction of RNA Structure*. Methods in Enzymology, vol. 180.

[9] Zuker M, Steigler P. *Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information.* Nucleic Acids Research vol. 9 Number 11981.

[10] Gregg B. *Systems Performance; Enterprise and the Cloud.* Pearson Education 2014 Upper Saddle River, NJ

[11] Cantrill B. *Eulogy for a benchmark.* http://dtrace.org/blogs/bmc/2009/02/02/eulogy-for-a-benchmark/. 2009.