Concurrency Final Project

Himanshu Reddy
Department of Computer Science
UT Austin

December 4, 2023

1 Abstract

I implemented a Gibbs sampling motif finding algorithm for DNA sequences. I implemented both a sequential and CUDA-based parallel version to compare the runtimes as different problem parameters scaled. I then tested the runtimes while scaling four different parameters. The results show that, overall, the parallel algorithm runs faster than the sequential algorithm across all input dimensions.

2 Introduction

Motif finding is a classical problem in biological sequence analysis. In its most basic form, we are trying to find a common substring of length K in N different strings. This common shared substring is called a motif. Finding motifs is of great importance in bioinformatics, as they can elucidate common functional elements and their locations across various biological sequences. This project focuses specifically on motifs in DNA sequences.

The problem statement seems simple enough. However, the basic motif finding scheme presented above is insufficient for practical use. DNA sequencing is inherently noisy, so some nucleotides in the input data may not be accurate. Additionally, we may not expect a motif to be present in all of the input sequences, but rather most of them. These considerations make the search space enormous, and many motif-finding algorithms use heuristics to find good (but perhaps sub-optimal) motifs in order to achieve much faster runtimes. The Gibbs sampling algorithm is one such motif finding algorithm.

While the Gibbs sampling algorithm is a popular motif finding algorithm, there does not seem to be much existing literature regarding speedups gained from parallelizing the algorithm. In fact, I was only able to find one study that attempted to parallelize the Gibbs motif finding algorithm using CUDA. However, the paper was presented in 2009 (so it used much older hardware) and made modifications to the algorithm, changing its semantics. Additionally, the paper reported only three runtime data points, making it hard to understand the general performance of the algorithm.

This project parallelizes the Gibbs motif finding algorithm using CUDA while preserving the algorithm's semantics. As far as I know, this is the only project that does so. This project also does this on modern GPU hardware, enabling a more up-to-date understanding of the scalability of this algorithm. Finally, this project reports comprehensive scaling results so that we can better understand the true scalability of a CUDA-enabled Gibbs sampling motif finding algorithm.

3 Hardware Details

I ran my experiments on the "pedagogical-5" machine. The details of the hardware are reported below. The server ran the Linux Ubuntu OS, with a version of 20.04.6 LTS.

Model	Memory (GB)	CUDA Cores
Quadro RTX 6000	24	4,608

Table 1: GPU Details (note that there are two identical GPUs with the above specifications on this machine, but the algorithm only used one). The GPU name was obtained with the ”nvidia-smi” command.

Model	Logical Cores	Clock Speed
Intel(R) Xeon(R) Gold 6226R	64	2.90GHz

Table 2: CPU Details

4 Project Goals

As stated above, the goal of this project is to comprehensively benchmark the parallel motif finding algorithm so that we can understand its relative speedup compared to the sequential algorithm. To do this, I measured performance while varying the four different parameters below:

- Number of Iterations (base value = 100): How many times the algorithm ”samples” new motifs before returning the best solution.
- Length of Motif (base value = 10): The length of the shared motif (as the number of DNA nucleotides) we are searching for in the input sequences.
- Number of Sequences (base value = 100): The number of input DNA sequences.
- Length of Sequences (base value = 2000): The length of each input DNA sequence (as the number of DNA nucleotides).

The base values represent the default values for the parameters of each run. I ran four experiments, each varying one of the above parameters. Each experiment had five iterations, and the value of the varied parameter would be the base value * 2^i, where i = {0, 1, 2, 3 4}. Structuring my experiments in this way allowed me to see the effects of varying these four crucial parameters in isolation, allowing for a more nuanced understanding of the scalability of this parallel algorithm.

5 Major Design Decisions / Issues

The design for the sequential algorithm was straightforward: it followed the description of the Gibbs sampling motif finding algorithm in Bioinformatics Algorithms: An Active Learning Approach. It was implemented as a series of steps, where each step was handled by one or more functions. The parallel algorithm also followed the same semantics and structure, but I replaced the parallelizable steps with CUDA kernels. I used the Hamming distance as the distance metric for the score function, but this could easily be modified for other distance metrics (e.g. entropy).

One major design issue I encountered was thinking through how to effectively parallelize this algorithm for CUDA devices. After thinking through the design space and implementing the algorithm, I found that it was hard to perfectly coalesce memory and avoid bank conflicts. However, the parallel implementation still ran significantly faster overall.

6 Implementation Tricks / Hacks

The implementation ”tricks” I used were not too complex, and they simply amounted to making use of GPU resources efficiently. This included minimizing memory transfers between the host/device, reusing memory allocations in repeated computations, and structuring the kernel code to effectively take advantage of the GPU architecture (e.g. by accessing adjacent memory locations in an iteration).

7 Major Limitations / Unexpected Roadblocks / Future Improvements

As far as I know, there aren't any major limitations for my motif finding algorithm, as it is a faithful implementation of the Gibbs sampling algorithm. The Gibbs sampling algorithm itself has some limitations; notably, it's not guaranteed to converge to the optimal motif, and may get stuck in local minima which prevents the algorithm from making progress. One limitation on the input data is that the algorithm only accepts DNA sequences; however, it should be relatively straightforward to adapt it to protein sequences as well.

One unexpected roadblock was how long it took to implement the sequential algorithm. This was my first time implementing this algorithm at all, and there were several little details that took some time to work out and fix before getting the sequential algorithm running. Another unexpected roadblock was the difficulty of adapting the algorithm to the GPU architecture, which is described above.

A future improvement would be to attempt to reorganize the algorithm to better take advantage of the GPU architecture. I think it might be possible to structure the algorithm in a way that better coalesces memory and reduces bank conflicts, but it would likely be quite complex and take much more work than what has already gone into this project (and the improvements may be marginal). Another future improvement would be to create some wrapper code to run this algorithm many times, using multiple CUDA streams, on the same input with different random seeds and measure GPU occupancy — this is something that would be done on real-world datasets to find the best possible motif, as the Gibbs sampling algorithm may return different results on each run. Finally, another future improvement would be to measure the effect of speeding up different portions of the algorithm individually in order to see what parts of the algorithm derive the most benefit from parallelization.

8 Evaluation

Figure 1, Figure 2, Figure 3, and Figure 4 show the runtimes for each algorithm as well as the associated speedup. Note that the runtimes are in milliseconds and the speedup is calculated as sequential runtime / parallel runtime. As we increase the input size, the parallel algorithm has a clear performance benefit in all cases except for the case where we increase the number of sequences. In that experiment, shown by Figure 3, the speedup is increasing but does not go above one. To test if there was a speedup for larger input sizes, I ran another experiment with 10,000 sequences; the results are shown in Table 3, and they show that there is a speedup as the number of sequences in the input increases. These results show that the parallel algorithm is more performant than the sequential algorithm for large parameter sizes.

The results of these experiments show that the parallel algorithm achieves a speedup across all input dimensions. Therefore, we can conclude that GPU-accelerated computing is very helpful for the motif finding task. Parallelizing other bioinformatics algorithms is a promising avenue for the field, as many bioinformatics tasks have to cope with massive datasets and have a structure that allows for effective parallelization.

Number of Sequences	Sequential Runtime	Parallel Runtime	Speedup
10,000	931	622	1.50

Table 3: Algorithm runtimes and speedup with 10,000 sequences. Note that the other parameters were the base values listed above.

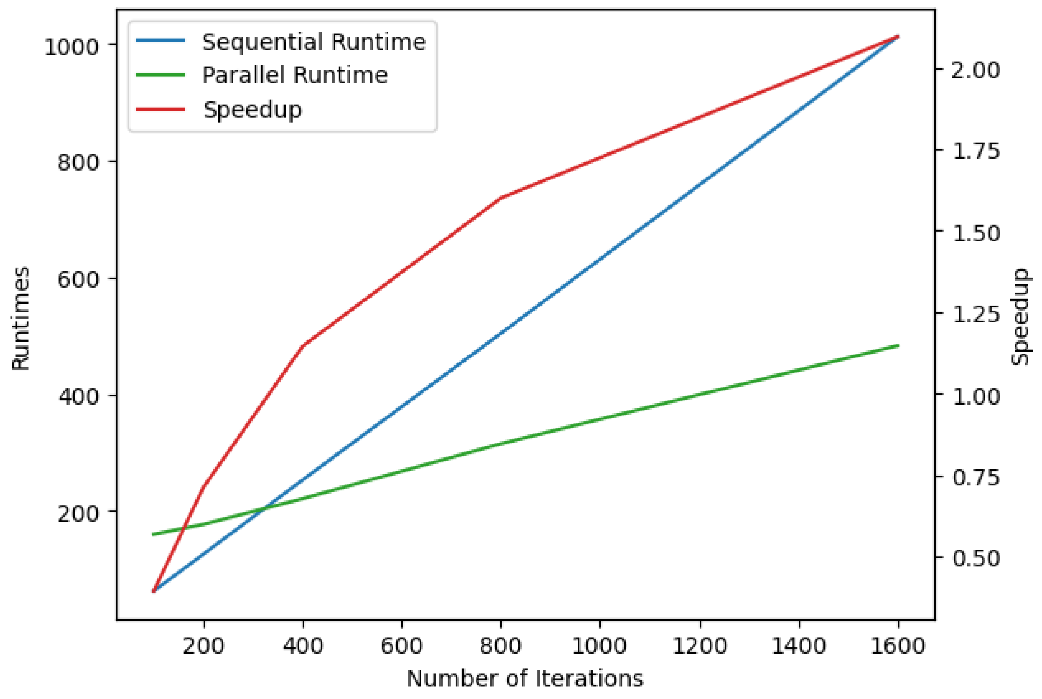

Figure 1: Graph showing the runtimes (rounded to the nearest millisecond) of both algorithms and the associated speedup as the number of iterations (the number of times the Gibbs sampling procedure was performed) is increased.

9 Time Spent on Project

About 50 hours.

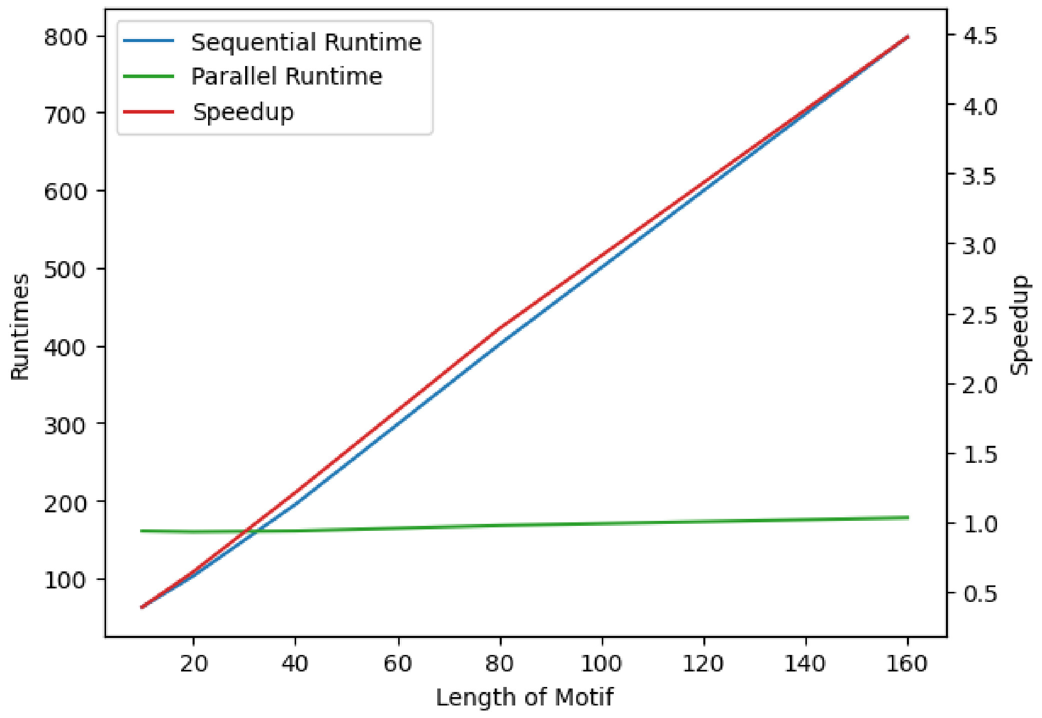

Figure 2: Graph showing the runtimes (rounded to the nearest millisecond) of both algorithms and the associated speedup as the length of the motif we are searching for (measured in base pairs, or characters) is increased.

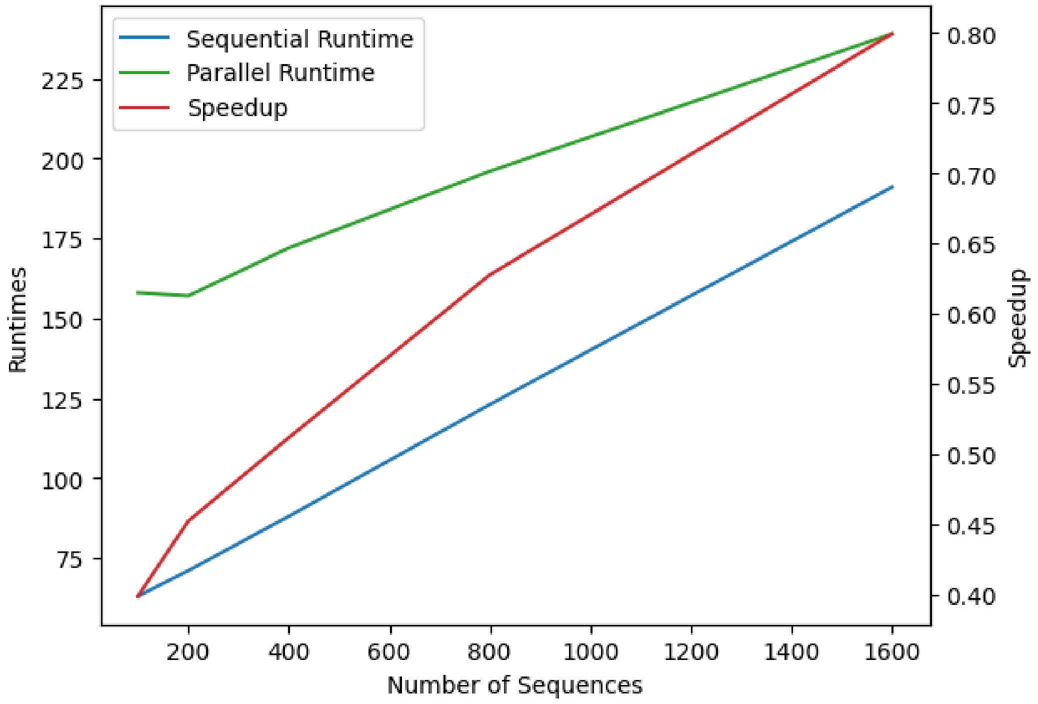

Figure 3: Graph showing the runtimes (rounded to the nearest millisecond) of both algorithms and the associated speedup as the number of input sequences is increased.

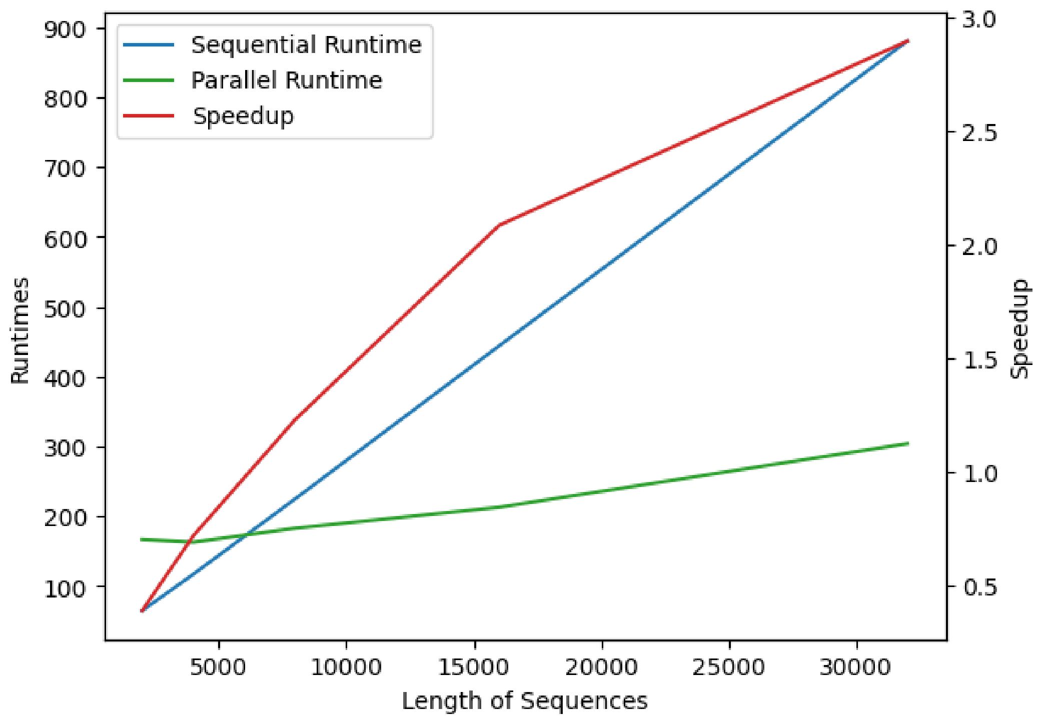

Figure 4: Graph showing the runtimes (rounded to the nearest millisecond) of both algorithms and the associated speedup as the length of the input sequences (measured in base pairs, or characters) are increased.