K-mer matching using the Automata processor in *de novo* DNA assembly

Michael Conry
Advisor: Dr. Alan Lemmon
Florida State University

Abstract *De novo* assembly of post-sequencing DNA reads is a complex computational task requiring the comparison of a massive amount of sequenced reads against reference sequences. To date, there have been attempts at constructing heuristic algorithms to cut down on the intense computational demands when using a traditional CPU as well as implementation of the algorithms on other architectures such as GPUs.

This study analyzes the application of a novel non-Von Neumann memory hardware, Micron's Automata Processor(AP), which is theoretically capable of dramatically reducing computation time through hardware based massively parallelized pattern matching[1]. We convert 20-mer DNA sequences into abstract mathematical models known as a non-deterministic finite automata and use the Automata Simulator to match them to an input stream. By estimating the drastic decrease in computation time required and demonstrating the benefits of the exhaustive nature of the pattern matching technique we attempt to show the advantages of using this device in place of a traditional CPU.

Introduction

One of the most computationally time consuming tasks in bioinformatics is the *de novo* assembly of sequenced short reads into longer consensus sequences. Longer consensus sequences provide more information about the genes in question and phylogenetic software will often give highly inaccurate results if the sequences are too short. Unfortunately, building these consensus sequences involves matching billions of bases together and storing the results, often exhausting both processing power and memory. There have been many promising software assemblers such as Bowtie[2], which uses the Burrows-Wheeler transformation to align reads to a reference geneome. However, in this study we are focusing on a novel hardware solution instead.

The Automata Processor(AP) is an alternative to traditional Von Neumann architecture CPUs and boasts groundbreaking parallelized pattern matching speeds. It achieves these speeds through parallelized hardware implementation of an abstract mathematical model known as Non-Deterministic Automata(NFA). Automata are state machines that are governed by their transition function. They are often represented by circles (states) and arrows (transitions) between them. When an Automaton is fed an input symbol it moves from the state it was at to a new (or back to the same) state based on the transition function.

Figure 2 is a representation of an NFA, which is a special type of state machine that can allow more than one active state at a time. NFAs can be easily made to represent text patterns (or most other data) to query a larger text. Because of the ability to be at multiple states at a time, NFAs implemented on the AP can check large streams of data for matching patterns in parallel faster than high performance computing clusters and can solve previously unsolved NP-hard problems[3]. When using the Automata Processor the computation time is a result of the complexity of the NFA, the rate of activation of reporting states, and the length of the input stream. Each input symbol (in our case a nucleotide base) is processed at 7.5 nanoseconds.

To solve the assembler bottleneck, we aim to use the AP hardware as a preprocessor to match k-mers (k length sequences of DNA) to sequenced reads. By performing this step on the AP hardware we can drastically reduce the most computationally complex step of the post sequencing process. Herein, we show the results of using the AP Simulator to test and design automata and to estimate computation time for different sizes of DNA sequences.

Assembly Workflow using Automata Processor

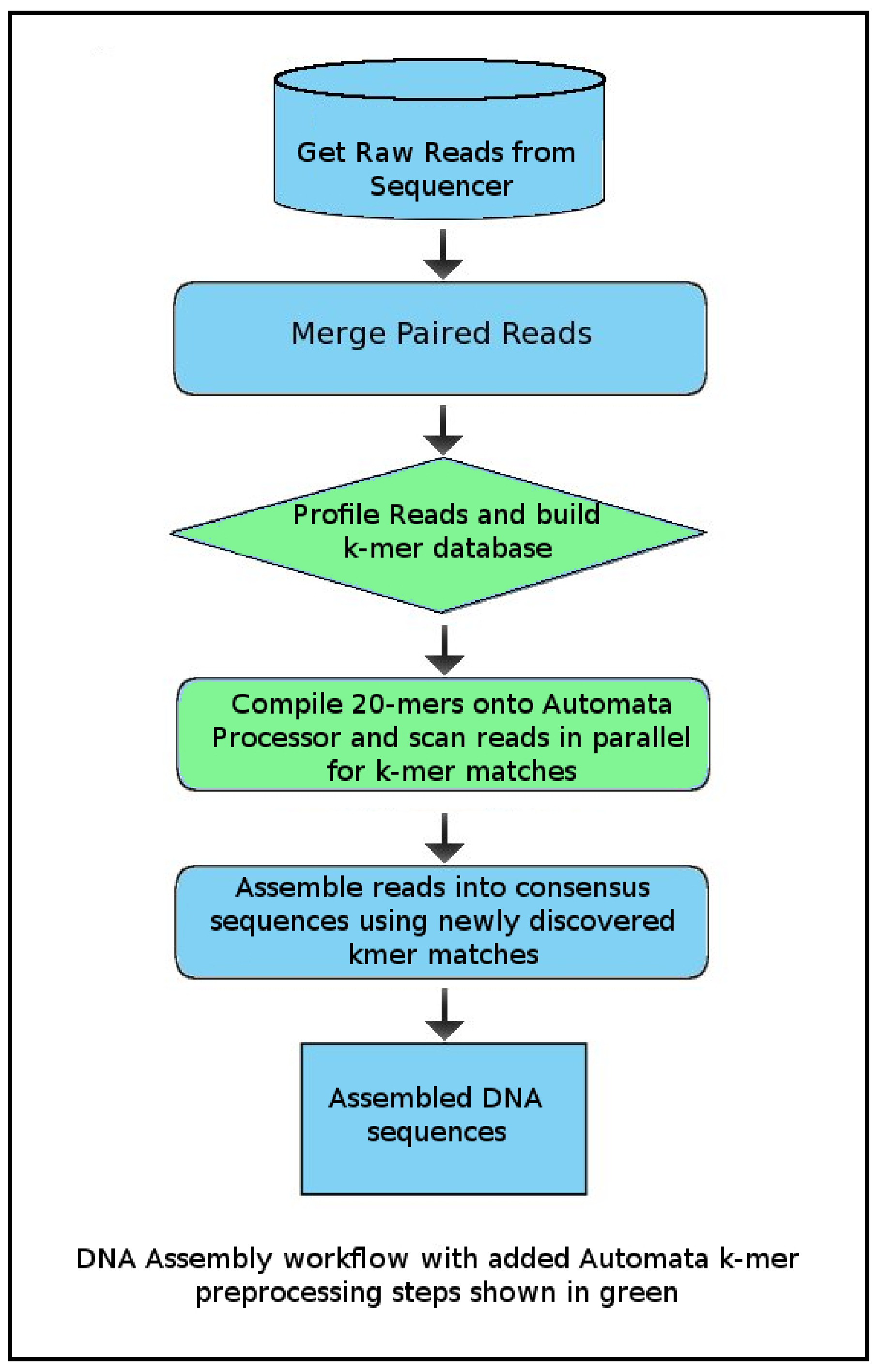

Get Raw Reads from Sequencer
Merge Paired Reads
Profile Reads and build k-mer database
Compile 20-mers onto Automata Processor and scan reads in parallel for k-mer matches
Assemble reads into consensus sequences using newly discovered kmer matches
Assembled DNA sequences
DNA Assembly workflow with added Automata k-mer preprocessing steps shown in green

Fig. 1

This workflow demonstrates how the k-mer matching done on the AP can fit into our DNA assembly pipeline. Currently, the reads are gathered from the sequencer, then merged and assembled using programs run on traditional CPUs. The assembly section is the largest computational bottleneck in the process, but by using the AP we can reduce the runtime from hours to minutes on average.

To achieve this, a k-mer database is built from existing references and the reads themselves. Then, those k-mers (ranging from 20 to 25 bases) are compiled onto the AP hardware as distinct automata as shown in figure 2. The merged reads are provided as an input stream (Gigabases in size) which are scanned by the AP hardware, attempting to find matches to all the k-mers in parallel.

Once those matches are found their locations and k-mer id are reported. Finally, this information is fed back into the traditional assembly software to combine the reads into assembled sequences. This process can work for reference mapped assembly or *de novo* assembly.

Automata Design

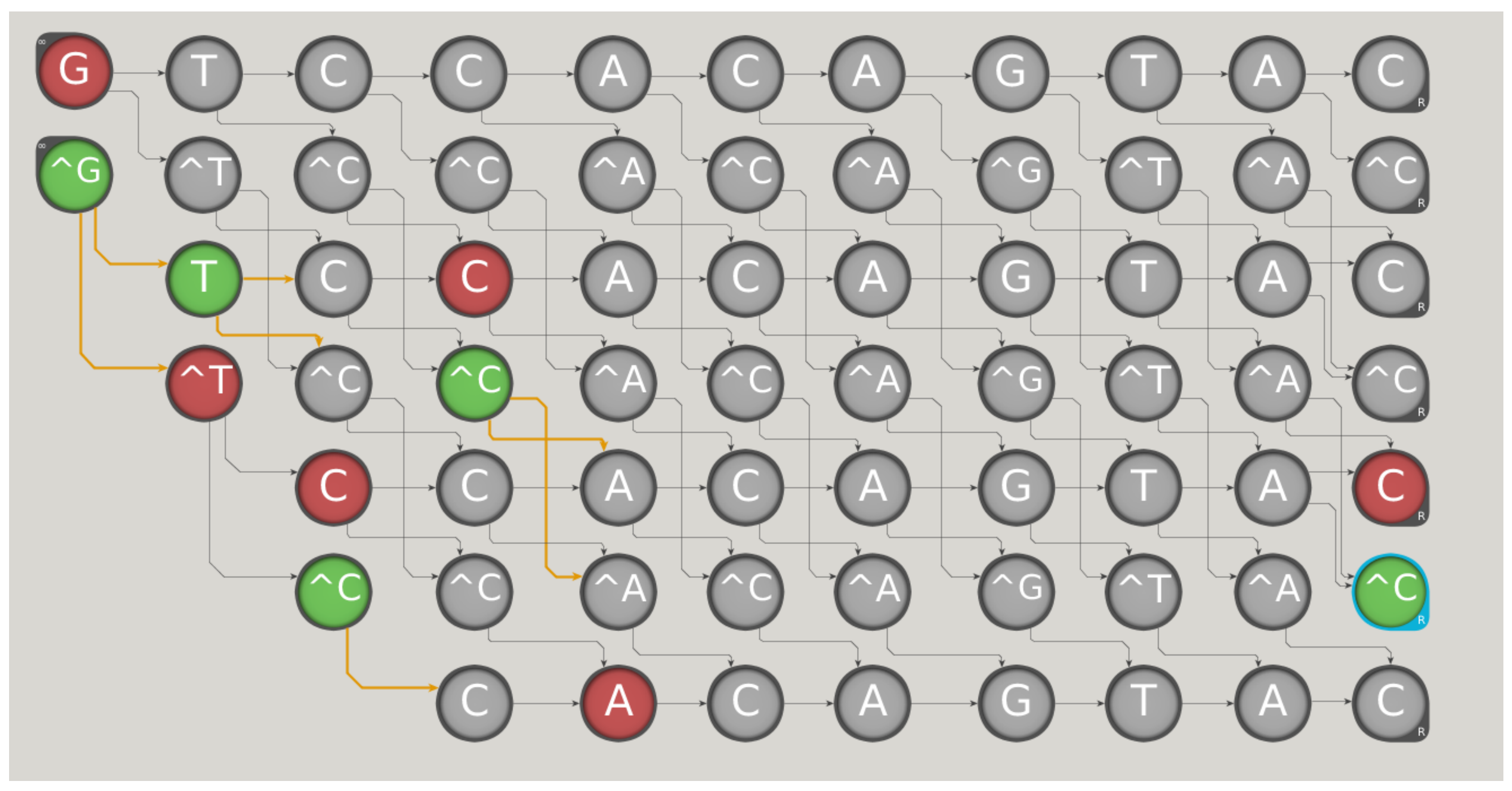

G T C C A C A G T A C
^G ^T ^C ^C ^A ^C ^A ^G ^T ^A ^C
T C C A C A G T A C
^T ^C ^C ^A ^C ^A ^G ^T ^A ^C
C C A C A G T A C
^C ^C ^A ^C ^A ^G ^T ^A ^C
C A C A G T A C

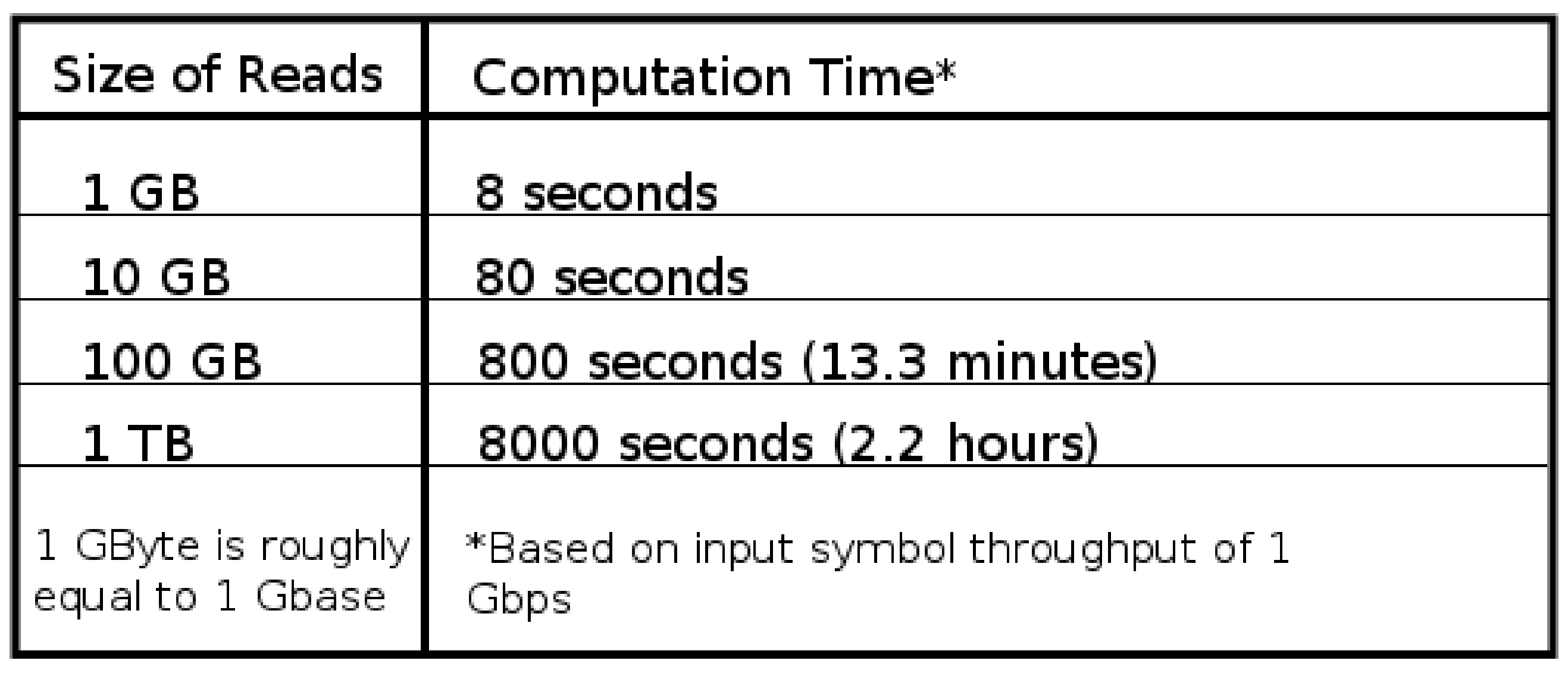
Fig. 2

The above automaton is an example of an NFA consisting of 11 nucleotides which is being fed a small input stream of 1000 DNA bases. Constructed in the AP Simulator, the automaton allows up to 3 mismatches. Each mismatch causes the current state in the automaton to transition down two rows. Because the hardware can match the pattern to the input stream in parallel, it attempts to match each new character as if it is the beginning of the pattern. The green states indicate the active states while the red indicates a dead-end. A match is reported if the automaton reaches the right most states before encountering four mismatches.

The reporting states at the right of the graph have unique ids that are reported when a match is found. This allows one to know how many mismatches were encountered (from 0 to 3), which k-mer was matched, and what position it's found at in the input stream.

In our testing we created much larger 20-mer automata that were matched against a 1.5 million base DNA sequence. The number of k-mers is limited by the number of State Transition Elements (STE) allowed on the hardware (around 1.5 million STEs using the single board hardware configuration). The number of STEs used by our mismatch automaton is defined by $l * (2d + 1) - d^2$ where l is the length of the k-mer and d is the number of mismatches allowable.[3] Thus, a 20-mer allowing 3 mismatches will use 131 STEs giving us space for around 11,450 k-mers.

Estimated Computation Time



Size of Reads	Computation Time*
1 GB	8 seconds
10 GB	80 seconds
100 GB	800 seconds (13.3 minutes)
1 TB	8000 seconds (2.2 hours)
1 GByte is roughly equal to 1 Gbase	*Based on input symbol throughput of 1 Gbps

Table 1

The table above details the estimated time required to exhaustively match the k-mers to the input streams of various sizes. Since the automata all match in parallel, the time required is almost entirely governed by the input stream size and does not depend on the number of k-mers. The hardware's input stream speed is 1 Gbit/second. In addition to this computation cost there is also a reporting cost of 2.5-40 symbol cycles (depending on the size of the information reported) when a reporting state is reached.

Conclusion

Using the AP Simulator, we have shown that the AP hardware will reduce computation costs drastically while still providing an exhaustive search of our data for k-mer matches. However, during our analysis we did not have access to the AP hardware, so there are possible unknown issues that will need to be overcome when the hardware is accessible such as automata compile time and reporting state costs. In addition to the assembly process, we believe the AP hardware will be useful when mapping reads in sequencer probe design.

References

- 1. P. Dlugosch, et al., "An efficient and scalable semiconductor architecture for parallel automata processing", IEEE Transactions on Parallel and Distributed Systems, 2013
- 2. B. Langmead, et al., "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome", Genome Biology, 10:R25, 2009.
- 3. I. Roy, S. Aluru, "Finding Motifs in Biological Sequences Using the Micron Automata Processor", Parallel and Distributed Processing Symposium, 2014 IEEE 28th International, p.415-424, 2014.