Prefix Hashing-Based Genome Sub-Sequence Matching

Mihir Awatramani, Mengduo Ma, Joseph Zambreno, Phillip Jones
Department of Electrical and Computer Engineering
Iowa State University
Ames, IA, USA
{mihir, marinama, zambreno, phjones}@iastate.edu

***Abstract*—Advances in Next Generation Sequencing (NGS) have enabled the generation of large amounts of DNA sub-sequences (i.e. reads). Sequence alignment tools are used to optimally align these reads to form a genome sequence by matching them against a known complete sequence (reference) of the same organism. In the first step, namely *exact matching*, alignment algorithms exactly match reads to the reference. Dissimilar regions are then substituted by either a different base-pair or a gap to predict a mutation or insertion/deletion, which provides information about the properties of the genome. This information can then be applied to applications like genome re-sequencing and DNA methylation. In this paper, we focus only on the exact matching part of the algorithm. We use a prefix based hash table of the reference genome as the underlying data structure and use multiple Graphics Processing Units (GPU) to process the individual reads in parallel. Distributing our algorithm across 23 NVIDIA GTX 480 cards resulted in a matching throughput of 15.5 thousand reads per second (283,173 reads in 18.2 seconds).**

***Keywords*-GPU; Sub-Sequence Matching**

I. Introduction

State of the art high-throughput sequencers produce large amounts DNA sequence fragments (reads) as a part of the sequencing process. The latest generation Illumina-Solexa system can generate over 50 million read sequences of length 30-50 bp in a single run in less than 3 days [1]. The Roche-454 systems have been reported to generate 400,000 sequences of length 250-500 bp in 7.5 hours [2]. Considerable research has been performed in the bioinformatics community to develop sequence alignment tools like BLAST [3], BLAT [4], SOAP [5], SeqMap [1], Bowtie [6], BWT-SW [7] and BWA [8]. Sequence alignment tools first attempt to find locations of exact matchs between each read and a reference genome (i.e. a known genome from the same species). Then, dissimilar portions of the sequence are modeled as areas of mutation to find the most optimum alignment. Our approach focuses on the first stage of a sequence alignment algorithm, exact matching.

The exact matching step of the algorithm can be viewed as a string matching problem which can be defined as: for strings of length m (pattern p) and length n (text t) over a finite alphabet Σ such that $m < n$; find locations in t, such that all characters in p exactly match characters in t. In this work, the text is the human genome with $|t| = 3.2$ billion; and patterns are read sequences with $|p| = 100$. Matching of each individual read is a mutually exclusive task and can be done in parallel. We use the high throughput processing power of GPUs to cover the large number of reads. The platform we used for our experiments was the NVIDIA GTX 480, which has 1.5 GB of RAM. As $|t|$ is very large, we used a hash table based data structure to prune the search space. The next section gives details of the hash table, experimental setup and the matching algortihm; while the final section summarizes our results and conclusions.

II. Algorithm

A. Generating the Data Structure

A first step in any alignment algorithm is to create a data structure of either the reference, the reads or both. The three most commonly used data structures are hash-tables, prefix trees and FM-index [9]. As most of the previous work [1,3-8] is based on short reads (30 to 60 bp), it was not clear how the data structures used in such approaches would perform with reads of length 100 bp. Results published in [10] show that SOAP and Bowtie scale well when the length of reads exceeds 100 bps. SOAP uses a hash-table indexing approach, while the latter uses FM-index.

Both hash table indexing and FM-index data structures have their pros and cons. While the FM-index data structure has a smaller memory requirement (2-8 GB for the human genome [11]), the runtime of a matching algorithm that uses a FM-index based data structure is linearly proportional to query length. On the other hand, the search time of a hash table is independent of query length, but suffers from the problem of size growing linearly with the number of samples (unique 100 bp long sub-sequence for our case).

Analysis of the reference genome, shows there are 2.8 billion such unique sub-sequences. As we need 4 bytes to store a reference genome index, the size of the hash table would be 11.2 GB. The RAM size of the GTX GPU 480 is 1.5 GB, and swapping chunks of the table between the disk drive and GPU RAM would significantly degrade performance. By spliting the hash table across 23 GPU cards, each storing 500 MB of the hash table, we are able to avoid swapping chunks of the hash table between GPU RAM and the disk drive. The remaining 1 GB of GPU memory is used to store the reference and reads.

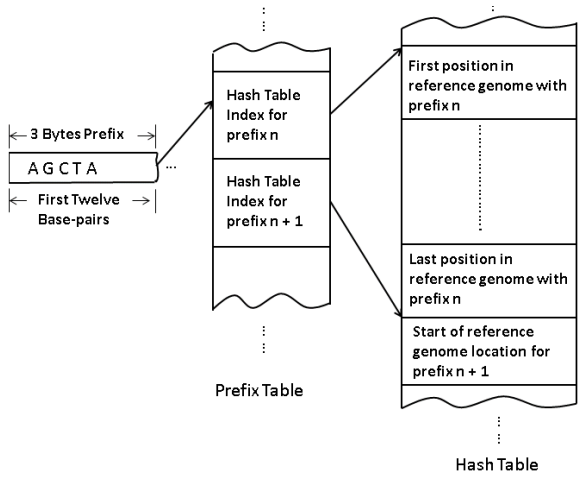

Fig. 1. Each unique 12 bp prefix corresponds to an entry in the prefix table. Each entry in the prefix table points to a start index in the hash table for that prefix.

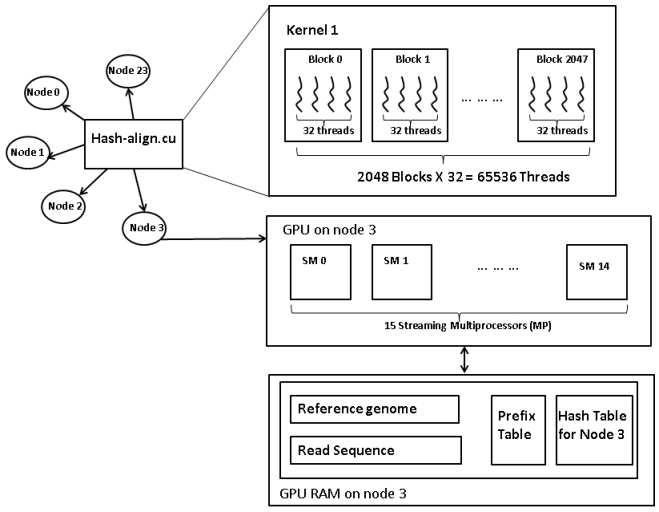

Fig. 2. Setup of 23 GPU nodes each having the reference, reads and a 500 MB chunk of hash table in its RAM. Reads as processed in batches of 65536 using 2046 threads blocks of 32 threads each.

Each GPU node needs to know which genome entries of the hash table are stored in its memory. As hashing algorithms are random, this is not a trivial task. By organizing the 28 billion unique 100 bp reads of the reference genome as a prefix tree and using a prefix of each read as its hash index, sequential locations of the hash table are populated with all the paths from the prefix to the root.

For our design, we used a 12 bp prefix and iterated over the reference genome to collect positions corresponding to each 100 bp sequence with a unique 12 bp prefix. The start index of the hash table for each unique prefix was then stored in a much smaller prefix table. The entries in the ṕrefix-indexed hash table' are sequential and bounded by the range given by the prefix table (Figure 1). The GPU at any node can now find out whether the hash table entry for a read exists in its memory using the prefix table. If the index of a given read does not reside in the GPU's memory it simply moves to the next read.

B. The Matching Algorithm

Matching of each individual read to the reference is an independent task and can be performed in parallel; with each CUDA thread processing its individual reads. The host loads the reference, tables and the reads into GPU memory (Figure 2). After the required data has been copied, the host launches a grid of threads for execution on the GPU. The on-chip hardware resources of the GPU are dynamically partitioned depending on the register requirement of the threads. Hence the number of threads in a grid is constrained by the register requirement of each thread.

We processed the reads over multiple kernels of 65536 threads each (grid of 2048 blocks, each of 32 threads). Each thread corresponds to a unique read from the read set. The algorithm looks at the first 12 base pairs to find an index in the prefix table. The entry at that index stores the starting hash-table location, where all positions in the reference genome for that prefix are stored sequentially (Figure 1). If the hash table location is not in the range that is stored in a given GPU's RAM, the read is skipped. If the index is present in the memory, the remaining 88 bp long sequence is copied to thread local memory and compared with the remaining read sequence. The matches found by all 23 GPU nodes are then combined to represent the matching positions of all the reads from the read set.

III. Results and Conclusion

The average time for matching all the reads on nodes 1-22 was 2.57 seconds. However, node 0 took 18.2 seconds to match the same number of reads. We determined this was due to a large number of sequences with a prefix of 0 (all A in the 12 bp prefix), which causes an imbalance in the work needed to process reads with this prefix. Consequently, our future work will use a FM-index based data structure, and we intend to compare these two approaches for read sequences of different lengths.

References

[1] H. Jiang and W. H. Wong, "Seqmap: mapping massive amounts of oligonucleotides to the genome," *Bioinfomatics*, 2008.

[2] K. L. Patrick, "454 life sciences: Illuminating the future of genome sequencing and personalized medicine," *Yale Journal of Bio. Med.*, 2007.

[3] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, 1990.

[4] W. J. Kent, "BLAT-the BLAST-like alignment tool," *Journal of Genome Res.*, 2002.

[5] R. Li, Y. Li, K. Kristiansen, and J. Wang, "SOAP: short oligonucleotide alignment program," *Bioinfomatics*, 2008.

[6] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biology*, 2009.

[7] T. Lam, W. Sung, S. Tam, C. Wong, and S. Yiu, "Compressed indexing and local alignment of dna," *Bioinfomatics*, 2008.

[8] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows-Wheeler transform," *Bioinfomatics*, 2009.

[9] P. Ferragina and G. Manzini, "An experimental study of an opportunistic index," *SODA*, 2001.

[10] A. Hatem, D. Bozdag, and V. Umit, "Benchmarking short sequence mapping tools," in *Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine*, 2008.

[11] H. Li and N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing." *Bioinformatics*, 2010.