A FM-index transformation to enable large k-steps

Rubén Langarita*, Adrià Armejach*†, Miquel Moretó*†
*Barcelona Supercomputing Center (BSC), Barcelona, Spain
†Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
E-mail: {ruben.langarita, adria.armejach, miquel.moreto}@bsc.es

Keywords*—*FM-index, sequence alignment, genomics.

I. Extended Abstract

The high demand for fast and low-cost genomic sequencing has pushed onward the rapid development of next-generation sequencing (NGS) technologies. These systems are able to produce huge amounts of short reads (in the order of giga base-pairs) per day of operation. Usually the first step in NGS corresponds to sequence alignment, where sequence reads must be aligned or compared to a genomic reference to identify regions of similarity. Most popular alignment methods are based on two types of index structures: suffix trees and hash tables [1]. When dealing with large reference genomes, great efforts were devoted to reduce memory requirements for sequence alignment. As a result, a set of alignment algorithms based on the FM-index (Full-text index in Minute space) structure have been developed.

The following section introduces the widely used FM-index technique. Then, we introduce our proposal, that uses a different data structure organization to enable search steps of longer symbols. Finally, we present our findings when evaluating our proposal on an Intel Knights Landing (KNL) machine.

A. Background - FM-index

FM-index is well suited for fast exact matches of short reads to large reference genomes while keeping a small memory footprint. The objective is to find short sequences of around 200 bases in a large reference genome. Normal problem sizes involve the human genome, which is around 3 gigabases. Therefore, to keep the memory footprint of these search algorithms manageable, FM-index uses the Burrows-Wheeler transform (BWT); a method for rearranging a character string that is useful for data compression.

The first step is to make all possible permutations of the reference genome, moving one letter from the start to the end each time (Figure 1). Then, we have to sort all the permutations alphabetically and extract the last column (Figure 2), usually also called BWT, which will be used to build the index.

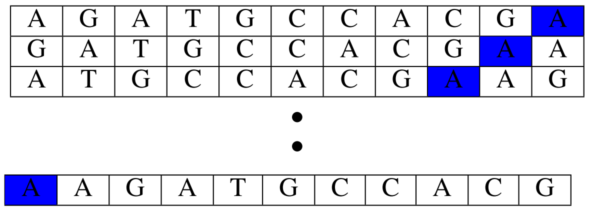

Fig. 1. BWT step 1: Permutations.

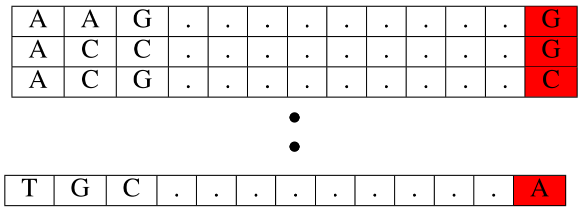
A A G . . . . G
A C C . . . . G
A C G C
T G C . . . . A

Fig. 2. BWT step 2: Sort alphabetically and extract last column.

FM-index consists of two structures, called *C* and *Occ*. The structure *C* indicates where each symbol (letter of the alphabet) starts in the sorted permutation matrix. The Occ structure has a row for each entry of the BWT column, and one column for each symbol. Each entry of Occ contains the number of occurrences of each symbol until that point in the BWT column (Figure 3).

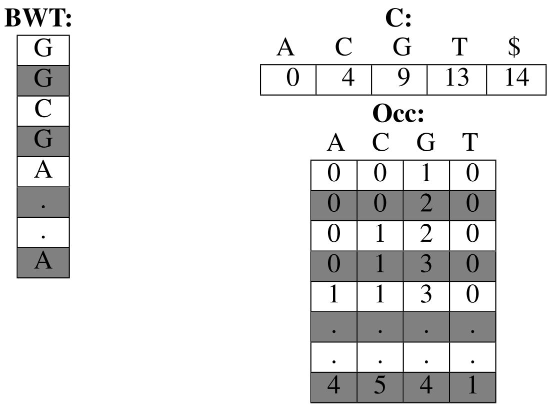

Fig. 3. *C* and *Occ* structures built from BWT.

FM-index uses two pointers (start pointer and end pointer) that indicate in each moment where are all the possible candidates in the sorted matrix. FM-index starts searching from the end of the search string. For the initialization, we have to read C and setup the pointers according to the first symbol read. Then, each iteration has to perform the following operations for each pointer: i) read the next symbol, ii) index C with that symbol, iii) index Occ with the symbol and the previous pointer and iv) add the entries of C and Occ and store the result in the pointer. This process repeats until all the sequence we are searching is read.

B. Changes - An Occ representation with constant size

A way to improve this algorithm is to increase the number of bases per symbol (K). For example, with K equals 2, the alphabet would be: $\{$AA, AC, AG, AT, CA, CC ... TT$\}$; instead of $\{$A, C, G, T$\}$. Increasing K also means the size of the structures grows exponentially, since these are dependent on the number of symbols in the alphabet, which is 4^K. In order to maintain the size more or less constant while increasing K, we propose a novel method to store the entries of Occ.

Each entry of the BWT increases by one a single counter of the Occ table per row (see Figure 3). For example, with a K of 15 bases per symbol, only one column out of 4^{15} will change for each entry of BWT. Because of that, we propose to store the indexes when the counters change. We will call this new structure Changes. If the reference genome is of 3 gigabases, Changes would occupy 12 GB of memory no matter the K. The size of C does increase with K, as in the original FM-index proposal. For K equal 15, C would occupy 4 GB of memory, i.e., 4^{15} entries x 4 bytes/entry. C also indicates the offsets where each column in the Changes array starts (Figure 4).

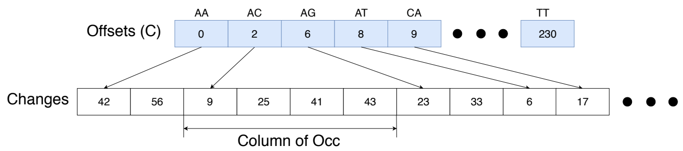

Fig. 4. Proposal structures.

In our proposal, instead of indexing Occ, we would have to perform a search over the column of the symbol we are searching. Using Figure 4 as an example, suppose that we want to find the symbol "AC", and the previous pointers were 31 for start and 42 for end. First, we obtain the boundaries of the column by indexing C. That is 2 and 6. Then, we have to count the number of elements lesser than the pointers of the previous iteration in the column, that is 2 for start (9 and 25) and 3 for end (9, 25 and 41). Finally, we have to add the value obtained indexing C and the counted elements. That would be $2 + 2 = 4$ for start and $2 + 3 = 5$ for end. This process is repeated for the next symbol in the search string with start and end pointers 4 and 5, respectively. Once the entire search string is processed the start and end pointers identify the matches.

With K equals 15, we find that some symbols are repeated significantly, while others do not appear in the whole reference. This leads to columns that are large, while others are empty. The following histogram (Figure 5 left) shows the distribution of column sizes, i.e., repetition of patterns with K equals 15, for a reference human genome. The x axis is the column size and the y axis the number of occurrences of columns with that size. Figure 5 right shows the number of columns accessed during time execution multiplied by the column size, which to illustrates the time spent processing the different column types. This shows the impact of the large columns in terms of execution time if we perform sequential search. As can be seen the distribution is clearly non normal, as certain patterns are much more recurrent in nature than others. In order to improve search speed over the larger columns we perform binary search, as by definition our structure has sorted columns.

C. Evaluation

Recent work by J.M. Herruzo et. al [2] proposed an optimization of the FM-index algorithm based on sampling and the use of bit vectors. They employ K equals 2 and sample 64 entries of the Occ by employing a 64 bit vector. We name this proposal from now on k2bv64. To evaluate this proposal they employ an Intel Knights Landing (KNL) system.

We use this as the baseline for comparison by evaluating our proposal using the same system setup also on a KNL machine. We employ the stacked MCDRAM in flat mode to map the data structures into this memory. In addition, the machine is configured to use 1GB huge pages and Hyper Threading, which means a total of 256 hardware threads are available.

For our proposal we use a k of 15, we will call this version k15inc. In order to measure the performance of the different versions we use a metric called "LF operations per second" (LFOPS). This measures the number of bases found per pointer. With a k of 2, each iteration performes 4 LFs, 2 for start and 2 for end, with a k of 15 would be 30 LFs... The best performe we obtained was 12.643 GLFOPS compare with the best version of the baseline 10.716 GLFOPS. We found that the algorithm performes better with sequences multiple of 15. In Figure 6 we can see the performance for the k2d64bv version and for our proposal with and without sequences multiple of 15.

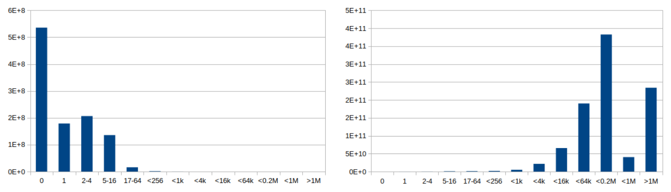

Fig. 5. Number of columns for each size with K = 15. Left figure shows the number of columns of the original structure, while right shows the columns sizes accessed during execution weighted (multiply by the column size).

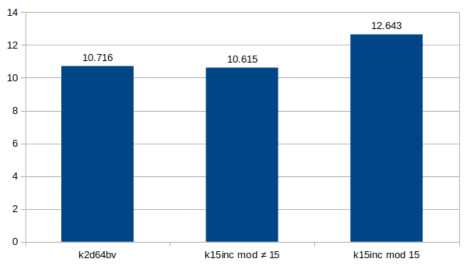

Fig. 6. Performance for different versions.

D. Future work

We are planning to perform further space reduction optimizations over the C structure, as it is currently limiting the use of larger K values. After evaluating the pending optimizations on the KNL machine, we plan to evaluate the proposal in other systems that do not feature MCDRAM. In addition, we are interested to evaluate our proposal using vectorization, i.e., using the recently proposed vector extension (SVE) by Arm.

References

[1] H. Li and N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing," *Briefings in bioinformatics*, vol. 11, no. 5, pp. 473–483, 2010.

[2] J. M. Herruzo, S. G. Navarro, P. Ibánez, V. V. Yufera, J. Alastruey, and O. Plata, “Accelerating sequence alignments based on fm-index using the intel knl processor,” *IEEE/ACM transactions on computational biology and bioinformatics*, 2018.

Rubén Langarita received his BSc degree in Computer Science from University of Zaragoza in 2018. Now, he is doing a Master in Innovation and Research in Informatics at Universitat Politècnica de Catalunya (UPC), while working at Barcelona Supercomputing Center (BSC).