Bioinformatics, 2016, 1–3
doi: 10.1093/bioinformatics/btw419
Advance Access Publication Date: 13 July 2016
Applications Note

OXFORD

Applications Note

sBWT: memory efficient implementation of the hardware-acceleration-friendly Schindler transform for the fast biological sequence mapping

Chia-Hua Chang[1,2], Min-Te Chou[1], Yi-Chung Wu[4], Ting-Wei Hong[1], Yun-Lung Li[1], Chia-Hsiang Yang[4,*] and Jui-Hung Hung[1,2,3,*]

[1]Institute of Bioinformatics and Systems Biology, [2]Institute of Biomedical Engineering, [3]Department of Biological Science and Technology, National Chiao Tung University, Hsin-Chu, Taiwan and [4]Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan

*To whom correspondence should be addressed.
Associate Editor: Inanc Birol

Received on March 9, 2016; revised on June 8, 2016; accepted on June 25, 2016

Abstract

ABSTRACT **Motivation:** The Full-text index in Minute space (FM-index) derived from the Burrows–Wheeler transform (BWT) is broadly used for fast string matching in large genomes or a huge set of sequencing reads. Several graphic processing unit (GPU) accelerated aligners based on the FM-index have been proposed recently; however, the construction of the index is still handled by central processing unit (CPU), only parallelized in data level (e.g. by performing block-wise suffix sorting in GPU), or not scalable for large genomes.
Results: To fulfill the need for a more practical, hardware-parallelizable indexing and matching approach, we herein propose sBWT based on a BWT variant (i.e. Schindler transform) that can be built with highly simplified hardware-acceleration-friendly algorithms and still suffices accurate and fast string matching in repetitive references. In our tests, the implementation achieves significant speedups in indexing and searching compared with other BWT-based tools and can be applied to a variety of domains.
Availability and implementation: sBWT is implemented in C++ with CPU-only and GPU-accelerated versions. sBWT is open-source software and is available at http://jhhung.github.io/sBWT/
Supplementary information: Supplementary data are available at *Bioinformatics* online.
Contact: chyee@ntu.edu.tw or jhhung@nctu.edu.tw (also juihunghung@gmail.com)

1 Introduction

Index structures, such as the Full-text index in Minute space (FM-index) derived from the Burrows–Wheeler transform (BWT) (Ferragina and Manzini, 2000) and enhanced suffix arrays (Abouelhoda, *et al.*, 2002), have been widely used in NGS applications for mapping short reads to the references (Li and Homer, 2010; Vyverman, *et al.*, 2012). Recently the biological applications of the FM-index are expanding; the FM-index has been used to compress NGS data (Cox *et al.*, 2012) and *de novo* assembly (Simpson and Durbin, 2012). These applications require dynamic construction of the indexes based on reads. For instance, in the construction of the overlap graph for doing *de novo* assembly as demonstrated by Simpson and Durbin (Simpson and Durbin, 2012). The suboptimal performance of

indexing, especially for huge read sets or assembling contigs/chromosomes has become a cause for concern.

Several algorithms have been proposed to construct the FM-index from a huge collection of reads, such as the dynamic FM-index (Makinen and Navarro, 2008), BCR (Bauer, *et al.*, 2013), CX1 (Liu, 2014) and ropeBWT2 (Li, 2014); however, these algorithms are unable to handle long fragments efficiently (up to kilobases), which appear in the later rounds of the contig assembly or are generated from long read sequencing techniques (Huddleston, *et al.*, 2014). Many efforts have been made to improve the efficacy of the construction of the index structure (Adjeroh *et al.*, 2008).

One BWT variant, Schindler transform (a.k.a. sort transform, ST), uses only order-k contexts (i.e. the order of suffixes are determined only by their first k-mers) and is still invertible with some extra complexity and performance penalty (Nong and Zhang, 2006; Schindler and Wien, 2001). ST is faster and much easier to parallelize with hardware than BWT. Several implementations of ST have been proposed but the indexes built are not applicable to searching (Culpepper, *et al.*, 2012; Nong and Zhang, 2007). Recently, Torres *et al.* (2012) has demonstrated the use of ST for short read alignment; however, the proposed approach required to keep huge data structure from the fully sorted BWT, which is not scalable and does not take full advantage of the ST.

To reduce the excessive memory footprint that is usually not applicable to graphic processing unit (GPU) or hardware accelerated environments, we herein propose novel, simple and highly parallelizable algorithms coupled with ST and still competent in string matching for general purposes. Two implementations of our algorithms (i.e. sBWT) are provided and were fully tested and compared with existing tools. We also applied our implementations to improve the most time-consuming step in *de novo* assembly and thereby exemplified the broad utility of sBWT. sBWT can be incorporated in many existing algorithms and pipelines to benefit a wider range of applications.

2 Methods and results

We build the k-ordered FM-index based on ST for the long and repetitive reference from its k-ordered suffix array (k-ordered SA) with only three simple steps: *split sort*, *k-ordered bucket sort*, and *merging and compression* (see *Supplementary Materials* for more details). In the *split sort* step, as described in previous literature (Karkkainen, 2007) but much simplified, extract a sufficiently large set (e.g. $n = 1000$) of suffixes from the reference sequence T by random sampling, and then, in the *k-orded bucket sort* step, sort the set of suffixes in the lexicographical order with the multikey quicksort and/or the bucket sort. Then use every mth (e.g. $m = 100$) element of the sorted set as splitters to group all suffixes of T into $n/m + 1$ suffix blocks. Suffixes that are bigger than the ith splitter and smaller than the $i + 1$th splitter lexicographically fall in the $i + 1$th suffix block. Note that the relative order of these blocks has to be maintained in the succeeding step. To save memory, only one block is created at a time and the indexes of all the suffixes of the block are stored in secondary storage. Multiple blocks can be processed in parallel to decrease running time when sufficient memory is provided. The values of n and m should be determined according to the maximal memory available.

In the *merging and compression* step, the resulting $n/m + 1$ transformed strings are concatenated in the same order of the suffix blocks and are compressed to form the k-ordered FM-index (i.e. the ST string) including other auxiliary tables (i.e. the C, Occ tables, and a specially designed location table, see below and supplementary method for more details). Since biological sequences consist of a relatively small alphabet, the transformed string (T') can be further encoded to save space. Another edge of compressing the BWT string is that the backtracking algorithm (used for searching, see below) can recover multiple nucleotides in reading one single byte and reduce the time calculating positions in searching with the help of some additional lookup tables.

To search with the k-ordered FM-index, we basically follow the similar strategy as Torres *et al.* proposed; however, to save the memory needed for Torres's approach, we construct a lookup table called location table to store the mapping between the index (denoted as i) of T' and the start position of the corresponding suffix (i.e. k-ordered SA[i]) like the text offset table used in fully sorted FM-index (see *Supplementary Materials*); the key distinction is at that the table is sampled according to the k-ordered SA[i], instead of i, in an interval of v. In other words, only the suffixes whose start positions in the text are at the multiples of v are stored in the location table. This table can be constructed easily along with the construction of the T' (see *Supplementary Materials*). With the proposed location table, we can find all the locations within k recursions of enumerating the alphabet (i.e. $\Sigma = \{A, T, C, \text{and } G\}$; see Algorithm 1, 2, 3 and *Supplementary Materials*). Indeed, the maximum length of each query sequence is restricted to the parameters (i.e. k and v) upon construction, which could be inconvenience; however, since modern short read aligners fragmentize the query into several short 'seeds' and tailor the alignments of each seed against the reference together afterward (Langmead and Salzberg, 2012; Li and Durbin, 2010), we believe it is not difficult to choose the right combination of parameters to fit the general purpose of fast string searching by the proposed k-ordered FM-index (By default, k is set as 256, and v is 64; maximum query length = 192; see *Supplementary Materials* for more details).

We implemented the indexing and searching algorithms of the proposed k-ordered FM-index for both central processing unit (CPU)-only (sBWT) and GPU-accelerated (sBWT-CUDA) architectures. The implementations were fully tested and compared with other available tools in the use of general-purposed read mapping and creating the index of a huge set of reads for *de novo* assembly. We compared the performance of indexing and searching in simulated genomes using sBWT and sBWT-CUDA with that of two conventional FM-index based aligners (i.e. Bowtie and BWA; We picked Bowtie instead of Bowtie2 for its better performance of short read mapping) and four GPU-accelerated aligners (i.e. CUSHAW (Liu *et al.*, 2012), SOAP3-dp (Luo *et al.*, 2013), BarraCUDA (Klus *et al.*, 2012) and nvBWT [nvlabs.github.io/nvbio/]). The results showed great time efficiency of sBWT (Fig. 1A–C). The simplicity of the k-ordered FM index also led to a highly parallelizable GPU-acceleration friendly implementation, sBWT-CUDA, which significantly improves the performance of both indexing and searching.

To further demonstrate the usefulness of k-ordered FM-index, we used sBWT-CUDA to construct the index for *de novo* assembly from a subset of a real-life deep sequencing library (SRR065390, see *Supplementary Materials* for more details). sBWT-CUDA was compared with two other implementations of the FM-Index based indexing algorithms designed specifically for building indexes on a huge set of reads, ropeBWT2 (Li, 2014) and nvSetBWT, an implementation based on the CX1 algorithm (Liu, 2014) (Fig. 1D). The results suggest that sBWT-CUDA can have a better overall performance during the entire process of *de novo* assembly. More details and the parameters used for all tests and all tools can be found in *Supplementary Materials*.

The proposed k-ordered FM-index, location table and the accompanying algorithms are much simpler to implement than the conventional ones—a simple bucket sort can do all the tricks; no need to deal with repetition at all (e.g. difference covers sampling (DCS);

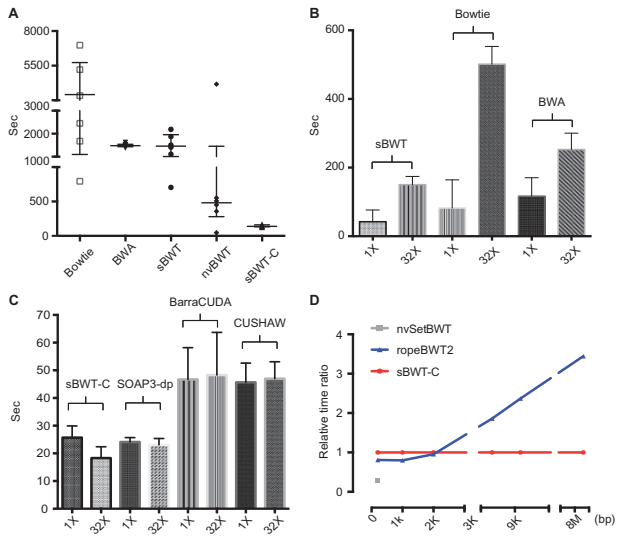

Fig. 1. The overview of the performance comparison with other tools. (sBWT-C: sBWT-CUDA) **(A)** Time needed for indexing genomes with different level of repetition. Five different types of genomes (which are all 80M bp long) were generated and differed in the level of repetition (i.e. 1X, 2X, 4X, 8X, 16X and 32X) and the averages of five repeated measurements are plotted. **(B)** Time needed for searching in the genomes with 1× and 32× genomes by CPU-only tools and **(C)** GPU-accelerated tools. **(D)** The speed of indexing a set of reads relatively to that by sBWT-CUDA. To be comprehensive, we simulated the scenarios of assembling long contigs by creating an array of reads with different length from SRR065390. We generated six different sets of reads, in which reads length are set to 100 (bp), 1k, 2k, 5k, 10k, and finally the length of the entire library. nvSetBWT worked only on 100bp-long reads in our tests. Note that CUSHAW and BarraCUDA do not provide any advancement in indexing, so the comparison with them only focused on searching. Searching by all tools was restricted in exact string matching only. CPU-only implementations (i.e. Bowtie, BWA and sBWT) were tested on a machine with 2.0 G INTEL 24-core CPU and 32G DDR3 memory. GPU-based implementations, namely sBWT- CUDA, SOAP3-dp, nvBWT, CUSHSAW and BarraCUDA, were tested on the same machine with an NVIDIA Tesla C2075 card (1.15G 448-core GPU and 6G GDDR5 memory) installed

Burkhardt and Karkkainen, 2003), see *Supplementary Materials*). The search space of the novel location table is bounded by $O(k|\Sigma|)$, whereas that of the conventional offset table, at worst case, is bounded by $O(N)$. It was somehow unexpected to see the searching of sBWT without any parallelism was faster than other CPU-only tools, but it is possible due to the complexity of their designs for more functionalities such as mapping quality evaluation or information logging.

Our innovation can also help the construction of the overlap graph (including finding irreducible edges) used in *de novo* assembly. Most excitingly, our implementation can create the index for multiple reads of any length. Other tools that designed specifically for the purpose do not scale as good as the proposed methods when the contig length grows along with the assembly. Future applications such as searching within multiple gigabase genomes can be better achieved by our innovations.

Acknowledgements

We thank the members of the Hung laboratories for helpful discussion and critical testing.

Funding

This work was supported by the
National Science Council (103-2221-E-009-128-MY2 and 104-2311-B-009 -002 -MY3 to J.H.H.) of Taiwan.

Conflict of Interest: none declared.

References

Abouelhoda,M.I. *et al.* (2002) The enhanced suffix array and its applications to genome analysis. *Lect. Notes Comput. Sci.* **2452**, 449–463.

Adjeroh,D. *et al.* (2008) Variants of the burrows-wheeler transform. In, *The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching*. Springer, US, pp. 153–186.

Bauer,M.J. *et al.* (2013) Lightweight algorithms for constructing and inverting the BWT of string collections. *Theor. Comput. Sci.*, **483**, 134–148.

Burkhardt,S. and Karkkainen,J. (2003) Fast lightweight suffix array construction and checking. *Comb. Pattern Match. Proc.*, **2676**, 55–69.

Cox,A.J. *et al.* (2012) Large-scale compression of genomic sequence databases with the Burrows–Wheeler transform. *Bioinformatics*, **28**, 1415–1419.

Culpepper,J.S. *et al.* (2012) Revisiting bounded context block-sorting transformations. *Softw. Pract. Exp..*, **42**, 1037–1054.

Ferragina,P. and Manzini,G. (2000) Opportunistic data structures with applications. *Ann. Ieee Symp. Found.*, 390–398.

Huddleston,J. *et al.* (2014) Reconstructing complex regions of genomes using long-read sequencing technology. *Genome Res.*, **24**, 688–696.

Karkkainen,J. (2007) Fast BWT in small space by blockwise suffix sorting. *Theor. Comput. Sci.*, **387**, 249–257.

Klus,P. *et al.* (2012) BarraCUDA - a fast short read sequence aligner using graphics processing units. *BMC Res. Notes*, **5**, 27.

Langmead,B. and Salzberg,S.L. (2012) Fast gapped-read alignment with Bowtie 2. *Nat. Methods*, **9**, 357–359.

Li,H. (2014) Fast construction of FM-index for long sequence reads. *Bioinformatics*, **30**, 3274–3275.

Li,H. and Durbin,R. (2010) Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics*, **26**, 589–595.

Li,H. and Homer,N. (2010) A survey of sequence alignment algorithms for next-generation sequencing. *Brief. Bioinformatics*, **11**, 473–483.

Liu,C. *et al.* (2014) GPU-Accelerated BWT construction for large collection of short reads. *arXiv*, eprint arXiv:1401.7457.

Liu,Y. *et al.* (2012) CUSHAW: a CUDA compatible short read aligner to large genomes based on the Burrows-Wheeler transform. *Bioinformatics*, **28**, 1830–1837.

Luo,R. *et al.* (2013) SOAP3-dp: fast, accurate and sensitive GPU-based short read aligner. *PloS One*, **8**, e65632.

Makinen,V. and Navarro,G. (2008) Dynamic entropy-compressed sequences and full-text indexes. *Acm T Algorithms*, **4**, 1–38.

Nong,G. and Zhang,S. (2006) Unifying the burrows-wheeler and the Schindler transforms. *Ieee Data Compression Conference*, Snowbird, UT, pp. 464–464.

Nong,G. and Zhang,S. (2007) Efficient algorithms for the inverse sort transform. *Ieee. Trans. Comput.*, **56**, 1564–1574.

Simpson,J.T. and Durbin,R. (2012) Efficient de novo assembly of large genomes using compressed data structures. *Genome Res.*, **22**, 549–556.

Schindler,M.B. and Wien,AT. (2001) *Method and apparatus for sorting data blocks*. Schindler Michael, United States. U.S. Patent 5659733 A.

Torres,J.S. *et al.* (2012) Using GPUs for the Exact Alignment of Short-Read Genetic Sequences by Means of the Burrows-Wheeler Transform. *Ieee Acm Trans. Comput. Biol.*, **9**, 1245–1256.

Vyverman,M. *et al.* (2012) Prospects and limitations of full-text index structures in genome analysis. *Nucleic Acids Res.*, **40**, 6993–7015.