Bioinformatics, 31(24), 2015, 4003–4005
doi: 10.1093/bioinformatics/btv506
Advance Access Publication Date: 30 August 2015
Applications Note

Sequence analysis

BigBWA: approaching the Burrows–Wheeler aligner to Big Data technologies

José M. Abuín[1,*], Juan C. Pichel[1], Tomás F. Pena[1] and Jorge Amigo[2]

[1]CITIUS, Universidade de Santiago de Compostela, Spain and [2]Genomics Medicine Group (GMX), Universidade de Santiago de Compostela, Spain

*To whom correspondence should be addressed.

Associate Editor: Inanc Birol

Received on April 6, 2015; revised on June 8, 2015; accepted on August 21, 2015

Abstract

Summary: BigBWA is a new tool that uses the Big Data technology Hadoop to boost the performance of the Burrows–Wheeler aligner (BWA). Important reductions in the execution times were observed when using this tool. In addition, BigBWA is fault tolerant and it does not require any modification of the original BWA source code.

Availability and implementation: BigBWA is available at the project GitHub repository: https://github.com/citiususc/BigBWA

Contact: josemanuel.abuin@usc.es

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Burrows–Wheeler aligner (BWA) is a very popular software for mapping sequence reads to a large reference genome. It consists of three algorithms: BWA-backtrack (Li and Durbin, 2009), BWA-SW (Li and Durbin, 2010) and BWA-MEM (Li, 2013). The first algorithm is designed for short Illumina sequence reads up to 100 bp, whereas the others are focused on longer reads. BWA-MEM, which is the latest, is preferred over BWA-SW for 70 bp or longer reads as it is faster and more accurate. In addition, BWA-MEM has shown better performance than other several state-of-art read aligners for mapping 100 bp or longer reads.

Sequence alignment is a very time-consuming process. This problem becomes even more noticeable as millions and billions of reads need to be aligned. For instance, new sequencing technologies, such as Illumina HiSeqX Ten, generate up to 6 billion reads per run, requiring more than 4 days to be processed by BWA on a single 16-core machine. Therefore, NGS professionals demand scalable solutions to boost the performance of the aligners in order to obtain the results in reasonable time.

In this article, we introduce BigBWA, a new tool that takes advantage of Hadoop as Big Data technology to increase the performance of BWA. The main advantages of our tool are the following. First, the alignment process is performed in parallel using a tested and scalable technology, which reduces the execution times dramatically. Second, BigBWA is fault tolerant, exploiting the fault tolerance capabilities of the underlying Big Data technology on which it is based. And finally, no modifications to BWA are required to use BigBWA. As a consequence, any release of BWA (future or legacy) will be compatible with BigBWA.

2 Approach

BigBWA uses Hadoop as Big Data technology. Hadoop is the most successful open-source implementation of the MapReduce programming model introduced by Google (Dean and Ghemawat, 2008). Hadoop applications are typically developed in Java, but BWA is implemented in C. To overcome this issue BigBWA takes advantage of the Java Native Interface (JNI) (Liang, 1999), which allows the incorporation of native code written in programming languages such as C and C++, as well as code written in Java. Two independent software layers were created in BigBWA. The first one corresponds to the BWA software package, whereas the other is, strictly speaking, our tool. This design avoids any modification of the BWA source code, which assures the compatibility of BigBWA with any BWA version.

The complete BigBWA workflow consists of four steps: convert the *fastq* input files to a Hadoop compatible format, copy the input data to the Hadoop cluster (HDFS), perform the alignment, and copy the output back from HDFS to the local filesystem. For more details, refer to the Supplementary Material.

Regarding the alignment process, BigBWA divides the computation into Map and Reduce phases. In the Map phase, BigBWA splits the reads into subsets, mapping each subset to a mapper process. Each mapper is responsible for applying the considered BWA algorithm using as input the reads assigned by BigBWA. Mappers are processed concurrently, speeding up the alignment process. In case any of the mappers fails, BigBWA would automatically launch another identical mapper process to replace the faulty one. At the end, BigBWA generates one output file per mapper. In the reducer phase those files are merged into one unique solution. Note that users could choose to skip the reduction phase.

Similar approaches to BigBWA are SEAL (Pireddu *et al.*, 2011) and pBWA (Peters *et al.*, 2012). SEAL uses Pydoop (Leo and Zanetti, 2010), a Python implementation of the MapReduce programming model that runs on the top of Hadoop. It allows users to write their programs in Python, calling BWA methods by means of a wrapper. As we will show in the next section, using Pydoop introduces an overhead as compared with using JNI. pBWA uses a standard parallel programming paradigm as MPI to parallelize BWA. Unlike BigBWA, pBWA lacks fault tolerant mechanisms. There are another important differences between these tools and BigBWA. First, SEAL and pBWA only work with a particular modified version of BWA, whereas BigBWA works directly with the original BWA implementation. Therefore, no modifications to the BWA source code are required by BigBWA, keeping the compatibility with future and legacy BWA versions. Second, both SEAL and pBWA are based on BWA 0.5 version, which does not include the new BWA-MEM algorithm. Therefore, to the best of our knowledge, BigBWA is the first tool to handle the parallelization of the BWA-MEM algorithm using Big Data technologies.

BWA has its own parallel implementation, but it only supports shared memory machines. For this reason, scalability is limited by the number of threads (cores) available in one computing node. BigBWA, however, can be executed on clusters consisting of hundreds of computing nodes.

Table 1. Main characteristics of the input datasets

Tag	Name	Number of reads	Read length (bp)	Size (GB)
D1	NA12750/ERR000589	12×10^6	51	3.9
D2	HG00096/SRR062634	24.1×10^6	100	13.4
D3	150140/SRR642648	98.8×10^6	100	54.7

All the datasets were extracted from the 1000 Genomes Project (Altshuler *et al.*, 2010).

3 Discussion

Performance: BigBWA was tested using data from the 1000 Genomes Project (Altshuler *et al.*, 2010) (Table 1 for details). Measurements were performed on a five-node AWS cluster with 16 cores per node (Intel Xeon E5-2670 at 2.5 GHz CPUs), running Hadoop 2.6.0. Detailed information about the experimental setup is provided in the Supplementary Material. Performance results for BigBWA and the other evaluated tools only take into consideration

Table 2. Comparison of the performance for the BWA-backtrack algorithm

Dataset	Tool	Execution time (minutes) Number of cores						Speedup Number of cores				
		1	4	8	16	32	64	4	8	16	32	64
D1	SEAL	148.5	55.7 ± 1.6	28.3 ± 1.0	22.2 ± 0.6	11.1 ± 0.1	5.7 ± 0.0	2.7	5.2	6.7	13.4	26.0
	pBWA		42.0 ± 0.7	25.3 ± 1.1	17.7 ± 0.5	9.2 ± 0.1	5.1 ± 0.1	3.5	5.9	8.4	16.1	29.1
	BigBWA		42.4 ± 0.9	23.8 ± 0.7	15.4 ± 0.4	8.5 ± 0.2	4.5 ± 0.1	3.5	6.2	9.6	17.3	33.0
D2	SEAL	556.9	186.5 ± 1.7	92.6 ± 0.8	68.1 ± 1.9	35.4 ± 0.7	18.5 ± 0.3	2.9	6.0	8.2	15.7	30.1
	pBWA		155.0 ± 0.4	94.5 ± 1.6	61.2 ± 1.5	32.7 ± 0.4	17.1 ± 0.3	3.6	5.9	9.0	17.0	32.6
	BigBWA		152.0 ± 0.3	82.3 ± 1.6	57.2 ± 0.8	30.3 ± 0.5	15.3 ± 0.1	3.7	6.8	9.7	18.3	36.4

Highlighted the best tool for a particular number of cores. For fair comparison with the other tools, BigBWA obtains these results using BWA version 0.5.10. Tool versions: pBWA 0.5.9 and SEAL 0.4.0.

Table 3. Comparison of the performance for the BWA-MEM algorithm

Dataset	Tool	Execution time (minutes) Number of cores						Speedup Number of cores				
		1	4	8	16	32	64	4	8	16	32	64
D1	BWA-Threads	106.6	27.6 ± 0.1	14.3 ± 0.1	10.9 ± 0.0	—	—	3.9	7.4	9.8	—	—
	BigBWA (hybrid)		29.6 ± 0.2	15.1 ± 0.3	11.8 ± 0.1	6.8 ± 0.3	3.6 ± 0.1	3.6	7.0	9.0	15.7	29.6
	BigBWA		29.1 ± 0.3	15.7 ± 0.1	7.9 ± 0.1	4.5 ± 0.1	3.0 ± 0.1	3.7	6.8	13.4	23.5	35.5
D2	BWA-Threads	258.0	66.0 ± 0.1	33.7 ± 0.1	24.9 ± 0.0	—	—	3.9	7.6	10.4	—	—
	BigBWA (hybrid)		69.6 ± 1.3	36.4 ± 0.6	24.5 ± 0.5	15.3 ± 0.1	8.8 ± 0.1	3.7	7.1	10.5	16.8	29.3
	BigBWA		69.1 ± 1.4	37.5 ± 0.4	20.7 ± 0.5	10.9 ± 0.3	7.2 ± 0.3	3.7	6.9	12.5	23.6	35.8
D3	BWA-Threads	3208.6	816.8 ± 2.5	408.1 ± 1.7	333.3 ± 0.3	—	—	3.9	7.9	9.6	—	—
	BigBWA (hybrid)		828.8 ± 9.5	431.1 ± 9.0	221.9 ± 4.0	183.3 ± 2.2	107.2 ± 0.8	3.9	7.4	14.5	17.5	29.9
	BigBWA		848.8 ± 13.6	444.9 ± 8.2	229.2 ± 5.1	120.1 ± 1.4	87.8 ± 0.2	3.8	7.2	14.0	26.7	36.6

Highlighted the best tool for a particular number of cores. These results were obtained using BWA version 0.7.12.

the alignment process time, which was calculated as the average of 5 runs per data point after one warm-up execution. Table 2 shows a comparison with SEAL and pBWA for the BWA-backtrack algorithm. In this case, BigBWA clearly outperforms these tools, especially when the number of cores used is high. In this way, speedups of 36.4× were reached with respect to the sequential case (using the original BWA tool as reference). It can also be observed that the scalability of SEAL is worse, caused by the overhead introduced by Pydoop with respect to the use of JNI.

Performance of BWA-MEM is shown in Table 3. It was measured using only BWA (threaded version) and BigBWA, because SEAL and pBWA do not support this algorithm. We have also included results for a hybrid version that uses BigBWA in such a way that each mapper processes the inputs using BWA with two threads. Results show that, with a small number of cores, BWA behaves slightly better than BigBWA. Note that BWA is limited to execute on just one cluster node and, therefore, we cannot provide results using more than 16 cores. Considering 16 cores, BigBWA is always the best solution but, due to the memory assigned per map task in our cluster configuration, only 13 concurrent tasks can be executed on one node. In this way, BigBWA always distributes the tasks between two nodes when using 16 cores. In addition, BigBWA shows good behavior in terms of scalability for all the datasets considered, executing up to 36.6× faster than the sequential case. Additional performance results are shown in the Supplementary Material.

Correctness: We verified the correctness of BigBWA by comparing its output file with the one generated by BWA. Differences range from 0.06% to 1% on uniquely mapped reads (mapping quality greater than zero), similarly to the differences shown by the threaded version of BWA with respect to the sequential case.

Funding

This work was partially supported by MINECO (Spain) grants TIN2013-41129-P and TIN2014-54565-JIN.

Conflict of Interest: none declared.

References

Altshuler,D. *et al*. (2010) A map of human genome variation from population-scale sequencing. *Nature*, **467**, 1061–1073.

Dean,J. and Ghemawat,S. (2008) MapReduce: simplified data processing on large clusters. *Commun. ACM*, **51**, 107–113.

Leo,S. and Zanetti,G. (2010) Pydoop: a Python MapReduce and HDFS API for Hadoop. In: *Proceeding of 19th Symposyum on HPDC*, ACM, Chicago (USA), pp. 819–825.

Li,H. (2013) Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv, 1303.3997v2*.

Li,H. and Durbin,R. (2009) Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, **25**, 1754–1760.

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

Liang,S. (1999) *Java Native Interface: Programmer's Guide and Reference*, 1st edn. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Peters,D. *et al*. (2012) Speeding up large-scale next generation sequencing data analysis with pBWA. *J. Appl. Bioinform. Comput. Biol.*, **1**, 1–6.

Pireddu,L. *et al*. (2011) SEAL: a distributed short read mapping and duplicate removal tool. *Bioinformatics*, **27**, 2159–2160.