Scalability Potential of BWA DNA Mapping Algorithm on Apache Spark

Zaid Al-Ars **Hamid Mushtaq**
Computer Engineering Lab, Delft University of Technology
2628 CD Delft, The Netherlands
E-mail: z.al-ars@tudelft.nl

Abstract

This paper analyzes the scalability potential of embarrassingly parallel genomics applications using the Apache Spark big data framework and compares their performance with native implementations as well as with Apache Hadoop scalability. The paper uses the BWA DNA mapping algorithm as an example due to its good scalability characteristics and due to the large data files it uses as input. Results show that simultaneous multithreading improves the performance of BWA for all systems, increasing performance by up to 87% for Spark on Power7 with 80 threads as compared to 16 threads (# of physical cores). In addition, Hadoop has slightly better performance of up to 17% for low system utilization, while Spark has up to 27% better performance for high system utilization. Furthermore, Spark is able to sustain high performance when the system is over-utilized, while the performance decreases for Hadoop as well as the native implementation.

1 Introduction

With the fast increase in the sizes of genomics datasets and the growing throughput of DNA sequencing machines, there is an urgent need to develop scalable, high-performance computational solutions to address these challenges. A number of different approaches are being investigated as possible solutions, ranging from highly connected, customized server-based solutions (Kelly15) to Hadoop-based big data infrastructures (Decap15). Predominantly, however, classical computer cluster-based solutions are the most widely used computational approach, either used locally or in the cloud (Stein10).

Each of these solutions has advantages and disadvantages as it relates to the scalability potential on large computer infrastructures. Customized solutions are expensive to design, but have the advantage of being highly optimized for the specific analysis pipeline being performed. Classical cluster-based solutions are more generic making them less costly, but also less effective. Genomics analysis problems, however, have the potential of offering a huge amounts of parallelism by segmenting the large input datasets used in the analysis. This creates the opportunity of using recent big data solutions to address these analysis pipelines.

In this paper, we investigate the scalability potential of using Apache Spark to genomics problems and compare its performance to both the native scalable implementation developed for classical clusters, as well as to Hadoop-based big data solutions (Zaharia12). This analysis is performed using a popular DNA mapping algorithm called the Burrow-Wheeler Aligner (BWA-MEM), which is known for its speed and high scalability potential on classical clusters (Li13).

This paper is organized as follows. Section 2 discusses typical genomics analysis pipelines and the different stages of the analysis. Section 3 presents the BWA-MEM algorithm, evaluates its scalability potential and discusses some of its computational limitations. Section 4 presents and compares the scalability potential of native implementations, Hadoop and Spark on the used test computer systems. Section 5 evaluates the effectiveness of these different solutions for genomics analysis by evaluating the scalability of BWA-MEM. Section 6 ends with the conclusions.

2 Genomics pipelines

In this section, we discuss the basic steps of a so-called *reference-based* DNA analysis pipeline, used for analyzing the mutations in a DNA dataset using a known reference genome.

DNA sequencing

The first step in any genome analysis pipeline starts by acquiring the DNA data using sequencing machines. This is done in a massively parallel fashion with millions of short DNA pieces (called *short reads*) being sequenced at the same time. These reads are stored in large files of sizes ranging from tens to hundreds of gigabytes. One standard file format used today to store these reads is called the FASTQ file format (Jones12).

Read mapping

The second step is used to assemble the short reads into a full genome, by mapping the short reads to a reference genome. This is one of the first computational steps in any genomics analysis pipeline, needed to reconstruct the genome. At the same time, it is one of the most computationally intensive steps, requiring a lot of

CPU time. The output represents a mapping of the possible locations of a specific read in the FASTQ file to a specific reference genome. BWA-MEM is one of the most widely used DNA mapping programs (Li13).

Variant calling

The third step is called variant calling, which uses algorithms to identify the variations of a mutated DNA as compared to a reference DNA. This analysis is becoming standard in the field of genomics diagnostics, where DNA analysis is used to advise clinical decision. The Genome Analysis Toolkit (GATK) and SAMtools are two widely used software tools for variant calling (Pabinger13).

3 BWA mapping algorithm

BWA-MEM is one of the most widely used DNA mapping algorithms that ensures both high throughput and scalability of the large datasets used in genomics. This section discusses the BWA-MEM algorithm which we use as an example for parallel algorithms used in big data application domains.

BWA-MEM, as well as many other DNA mapping algorithms, is based on the observation that two DNA sequences of the same organism are likely to contain short highly matched subsequences. Therefore, they can follow a strategy which consists of two steps: 1) seeding and 2) extension. The seeding step is to first locate the regions within the reference genome where a subsequence of the short read is highly matched. This subsequence is known as a seed, which is an exact match to a subsequence in the reference genome. After seeding, the remaining read is aligned to the reference genome around the seed in the extension step using the Smith-Waterman algorithm (Houtgast15).

In BWA-MEM, before starting the read alignment, an index of the reference genome is created. This is a one time step and hence not on the critical execution path. In our discussion, we assume that an index is already present. The different execution stages of BWA-MEM read alignment algorithm are described below. The first two stages belong to seeding.

SMEM generation

BWA-MEM first computes the so-called supermaximal exact matches (SMEMs). An SMEM is a subsequence of the read that is exactly matching in the reference DNA and cannot be further extended in either directions. Moreover, it must not be contained in another match.

Suffix array lookup

The suffix array lookup stage is responsible for locating the actual starting position of the SMEM in the reference genome. An SMEM with its known starting position(s) in the reference genome forms seed(s) in the reference.

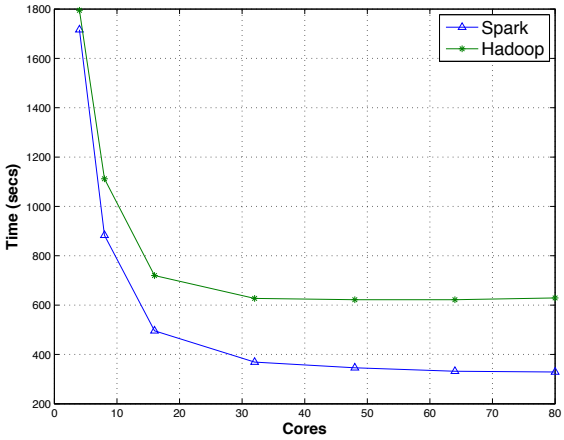

Figure 1: Execution time of WordCount on Power7

Seed extension

Seeds are subsequences of the read that are exactly matching in the reference genome. To align the whole read against the reference genome, these seeds are extended in both directions. This extension is performed using a dynamic programming algorithm based on Smith- Waterman.

Output

The read alignment information is written to a file in the SAM (sequence alignment/map) format (Li09).

4 Baseline performance analysis

The increasing sizes of big data files have called for a continued effort to develop systems capable of managing such data sizes and enabling the needed scalability to process them efficiently. Hadoop MapReduce is the current industry system of choice for big data systems, which uses the in-disk Hadoop distributed file system (HDFS). Apache Spark is emerging as a strong competitor in the field due to its in-memory resilient distributed datasets. This section compares these two systems using the WordCount benchmark to identify a baseline for the BWA-MEM implementation.

We tested the WordCount application on two different kinds of machines. The first one is an IBM PowerLinux 7R2 with two Power7 CPUs and 8 physical cores each. The Power7 cores are capable of executing 4 simultaneous threads. The second machine is an Intel Linux server with two Xeon CPUs and 10 physical cores each. The Xeon cores are capable of executing 2 simultaneous threads.

The results for the WordCount application are shown for the IBM Power7 and Intel Xeon in Figure 1 and 2, respectively. For the Hadoop version, the input files are read from the HDFS file system, while for Spark, the files are read from the local file system. We can see that on both machines, the Spark version is faster than the Hadoop version. Moreover, there is an approximately constant increase in the execution time of

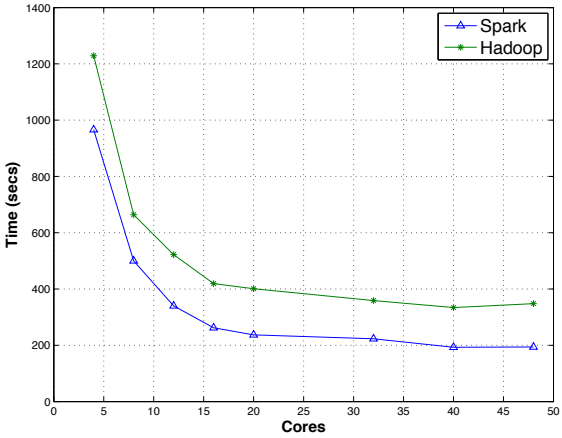

Figure 2: Execution time of WordCount on Xeon

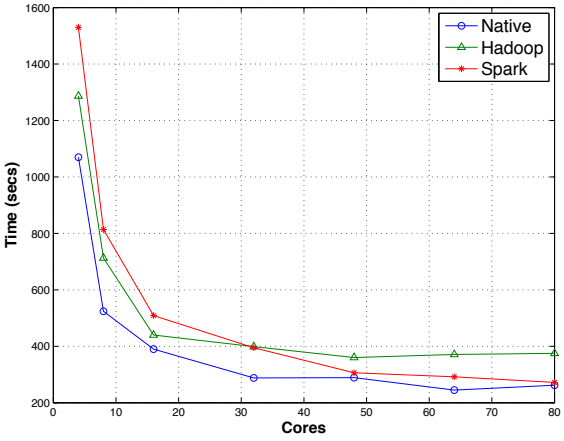

Figure 3: Execution time of BWA-MEM on Power7

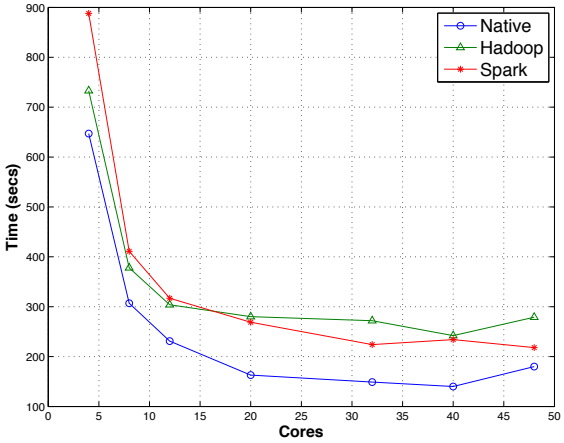

Figure 4: Execution time of BWA-MEM on Xeon

Hadoop as compared to that of Spark for the different thread count on both machines. This indicates a constant added overhead for Hadoop-based programs over Spark for different number of threads. This overhead is partly incurred by Hadoop having to access files from the HDFS system instead of the local file system.

The figures also show that the highest performance gains are achieved using scalability of the physical cores (up to 16 threads for the Power7 and 20 threads for the Xeon). In addition, high performance gains are achieved by running 2 threads per core, with the Power7 achieving better performance gains as compared to the Xeon. Further increase in the thread count on the Power7 only achieves marginal gains. One interesting remark is that on both machines, over-saturating the CPUs by issuing more threads than the machine is capable of causes Hadoop to loose performance slightly, while Spark is still capable of a (marginal) increase in performance.

5 Experimental results

In this section, we investigate the scalability potential of BWA-MEM on big data systems as an example of genomics data processing pipelines. We compare its native implementation, developed for classical clusters, to the performance of an Apache Spark as well as of a Hadoop-based big data implementation. The Hadoop version uses the Halvade scalable system with a MapReduce implementation (Decap15). In this evaluation, we use the same IBM PowerLinux 7R2 and Intel Xeon servers we used for the WordCount example above.

The results for the BWA mapping are shown for IBM Power7 and Intel Xeon in Figure 3 and 4, respectively. The results are shown for 3 different version of BWA: 1. native, 2. Hadoop, and 3. Spark. Both the Hadoop and Spark versions divide the input dataset of short reads into a number of smaller files referred to as *chunks*. For example, for these experiments, we had 32 input chunks. Halvade and Spark were run with 4 instances, which means that 4 simultaneous BWA tasks were run in parallel on different data chunks. The number of threads used in each experiment was varied by controlling the number of threads issued by each BWA instance (using the `-t` command line argument). For example, to use 12 threads, we used 4 instances with each instance having 3 threads, while to use 32 threads, we used 4 instances with 8 threads each.

It also has to be mentioned here that there are few differences in how files are read in Hadoop and the Spark versions. In the Hadoop version, the input zipped files are extracted on the fly by the Hadoop MapReduce framework and delivered to the mappers line by line, where each mapper is executing one instance of BWA. Each instance then processes the input data line by line. That data is read right away by the BWA instances using the stdin buffer. On the other hand, each instance in the Spark version reads a zipped input file and then decompresses it first. Afterwards, the decompressed file is forwarded as an input to a BWA instance. However, in both cases, the output is written to the local file system. It is also important to mention here that in the Power7 case, we wrote the output into a RAM

disk for all three BWA versions. This is because we wanted to know how good BWA scales with simultaneous threads without letting file I/O overshadowing the execution time.

On both the Power7 and Xeon machines, we can see that simultaneous multithreading improves the performance of BWA. This is because the BWA algorithm usually suffers from a large number of memory stalls, as a result of random accesses to the memory that renders the cache ineffective causing a high cache miss rate. These cache misses can be reduced by simultaneous multithreading, since some threads can run while others are stalled. The Power7 system is able to make significant performance gains using its capability to issue 4 simultaneous threads. Spark is able to increase in performance by up to 87% with 80 threads as compared to 16 threads (# of physical cores).

One interesting result in the figures is that while the Hadoop version is faster by up to 17% using lower number of threads, the Spark version gets faster by up to 27% at higher number of threads. This behavior can be explained by the way the input chunk files are handled. As mentioned before, the Hadoop version uses on-the-fly decompression of the zipped input chunk files, while the Spark version first decompresses a zipped input chunk file before using it. This causes an increased overhead that makes Spark run slower for lower number of threads. As the number of threads increases, Spark improves in performance and overtakes Hadoop in execution time.

Finally, the figures show that over saturating the thread capacity of the cores (i.e., issuing more threads than number of virtual cores available) causes the performance of the native version and the Hadoop version to degrade, while Spark is able to continue to improve in performance. This behavior is similar to the one observed in the WordCount example. Therefore, this could be caused by the internal implementation of the Spark and Hadoop systems themselves.

6 Conclusions

This paper analyzed the scalability potential of the widely used BWA-MEM DNA mapping algorithm. The algorithm is embarrassingly parallel and can be used as an example to identify the scalability potential of embarrassingly parallel genomics applications using the Spark and Hadoop big data frameworks. The paper compared the performance of 3 BWA implementations: 1. a native cluster-based version, 2. a Hadoop version, and 3. a Spark versions. Results show that simultaneous multithreading improves the performance of BWA for all systems, increasing performance by up to 87% for Spark on Power7 with 80 threads as compared to 16 threads (# of physical cores). The results also show that while the Hadoop version is faster by up to 17% using 4 threads, the Spark version gets faster by up to 27% at higher number of threads. Finally, the results also indicate that the Spark system is more capable of handling higher number of threads as it is able to continue to reduce its run time when over-saturating the thread capacity of the cores.

References

D. Decap, J. Reumers, C. Herzeel, P. Costanza and J. Fostier, "Halvade: scalable sequence analysis with MapReduce", *Bioinformatics*, btv179v2-btv179, 2015.

E.J. Houtgast, V.-M. Sima, K. Bertels and Z. Al-Ars, "An FPGA-Based Systolic Array to Accelerate the BWA-MEM Genomic Mapping Algorithm", *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, 2015.

D.C. Jones, W.L. Ruzzo, X. Peng and M.G. Katze, "Compression of next-generation sequencing reads aided by highly efficient de novo assembly", *Nucleic Acids Research*, 2012.

B.J. Kelly, J.R. Fitch, Y. Hu, D.J. Corsmeier, H. Zhong, A.N. Wetzel, R.D. Nordquist, D.L. Newsom and P. White,"Churchill: an ultra-fast, deterministic, highly scalable and balanced parallelization strategy for the discovery of human genetic variation in clinical and population-scale genomics", *Genome Biology*, vol. 16, no. 6, 2015.

H. Li, "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM", arXiv:1303.3997 [q-bio.GN], 2013.

H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, "The Sequence Alignment/Map format and SAMtools", *Bioinformatics*, vol. 25, no. 16, pp. 2078-2079, 2009.

S. Pabinger, A. Dander, M. Fischer, R. Snajder, M. Sperk, M. Efremova, B. Krabichler, M.R. Speicher, J. Zschocke, Z. Trajanoski, "A survey of tools for variant analysis of next-generation genome sequencing data", *Brief Bioinformatics*, bbs086v1-bbs086, 2013.

L.D. Stein, "The case for cloud computing in genome informatics", *Genome Biology*, vol. 11, no. 207, 2010.

M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, I. Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing", *NSDI*, April 2012.