Journal of Computational Science xxx (2017) xxx–xxx

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

parSRA: A framework for the parallel execution of short read aligners on compute clusters

Jorge González-Domínguez[a,*], Christian Hundt[b], Bertil Schmidt[b]

[a] Computer Architecture Group, University of A Coruña, Spain
[b] Parallel and Distributed Architectures Group, Johannes Gutenberg University Mainz, Germany

ARTICLE INFO

Article history:
Received 12 October 2015
Received in revised form
22 September 2016
Accepted 21 January 2017
Available online xxx

Keywords:
Short read alignment
High performance computing
Multicore clusters
Bioinformatics
PGAS

ABSTRACT

The growth of next generation sequencing datasets poses as a challenge to the alignment of reads to reference genomes in terms of both accuracy and speed. In this work we present *parSRA*, a parallel framework to accelerate the execution of existing short read aligners on distributed-memory systems. *parSRA* can be used to parallelize a variety of short read alignment tools installed in the system without any modification to their source code. We show that our framework provides good scalability on a compute cluster for accelerating the popular *BWA-MEM* and *Bowtie2* aligners. On average, it is able to accelerate sequence alignments on 16 64-core nodes (in total, 1024 cores) with speedup of 10.48 compared to the original multithreaded tools running with 64 threads on one node. It is also faster and more scalable than the *pMap* and *BigBWA* frameworks. Source code of parSRA in C++ and UPC++ running on Linux systems with support for *FUSE* is freely available at https://sourceforge.net/projects/parsra/.

1. Introduction

Short read alignment (SRA) is a crucial step in many bioinformatics pipelines. It consists in mapping DNA fragments (called reads) onto a reference genome, in order to locate the genomic coordinates these fragments come from. The rapid progress of next generation sequencing (NGS) technologies has led to large-scale datasets containing hundreds of millions or even billions of reads, which makes the SRA step time consuming.

Although efficient *seed-and-extend* based algorithms that provide high-quality alignments have been developed, their associated runtimes are still high. Examples include *GASSST* [1], *Bowtie2* [2], *GEM* [3], *SeqAlto* [4], *BWA-MEM* [5] and *CUSHAW3* [8]. The main algorithmic idea applied by all these tools are based on the fact that significant alignments usually contain short exact matches (so called *seeds*). Typical short read aligners thus map a given read by first identifying such seeds on the given reference genome. This is usually accomplished by using a pre-computed index data structure that allows for fast retrieval of short exact matches between query and reference genome. Subsequently, these seeds are extended and refined under certain constraints, such as minimal percentage identity or length, in order to filter out irrelevant seeds. Finally, more sophisticated but also computationally more expensive approaches (*e.g.*, dynamic programming based alignment algorithms) are employed to obtain the final alignments from the seeds. It should be noted that the time needed for the alignment of each read can vary as it depends on the number of associated seeds.

Parallelization can be used to accelerate this procedure. Most existing SRA tools only provide shared memory parallelism based on multi-threading, which limits their execution to single compute nodes. In order to overcome this limitation, there exist parallel implementations of certain SRA tools that can be executed on multicore clusters and exploit the computing capabilities of several nodes (*e.g., pBWA* [9] and *merAligner* [8]). However, the accuracy of the results provided by these multinode implementations is limited to only one mapping approach and they do not offer portability for the underlying aligner. For instance, *pBWA* is limited to a particular version of the *BWA* aligner [9].

Research on SRA approaches is still evolving and new methods with better accuracy in some scenarios are continuously developed. Therefore, it is not advisable to limit parallel frameworks to work with only one type of alignment (e.g. *pBWA* only works with the outdated 0.5.9 version of the *BWA method*). *pMap* [10] is a parallel framework that allows for working with several existing and previously installed tools (*e.g., Bowtie2*, *BWA-MEM* or *CUSHAW3*). It splits the workload among nodes and uses the selected method to complete the alignment in parallel on multiple compute nodes within a cluster. However, as will be shown in

* Corresponding author.
E-mail addresses: jgonzalezd@udc.es (J. González-Domínguez), hundt@uni-mainz.de (C. Hundt), bertil.schmidt@uni-mainz.de (B. Schmidt).

http://dx.doi.org/10.1016/j.jocs.2017.01.008
1877-7503/

the experimental evaluation, the scalability of *pMap* is low even for a moderate number of nodes. Recently, approaches based on the map-reduce paradigm have been presented for distributed execution of the BWA aligner [11–13]. However, their scalability is limited to a small number of compute nodes.

In this paper we present *parSRA*, a novel framework to parallelize SRA on multicore clusters which can work with different underlying methods and provides significantly better scalability than *pMap*. *parSRA* can use the most suitable alignment method for each situation, and even more accurate methods that can be developed in future. Moreover, *parSRA* is even more portable than *pMap* as its configuration file allows for the users to parallelize the execution of existing SRA tools without the need to modify the source code of *parSRA* or the aligner.

The rest of the paper is organized as follows. Section 2 reviews some related work. Our parallelization approach is described in Section 3. Experimental evaluations are presented in Section 4. Section 5 concludes the paper.

2. Related work

The implementation of parallel tools for SRA that resort to accelerators to reduce their runtime has attracted extensive research interests. The most popular accelerators for SRA are GPUs, and some examples of GPU-based tools are *CUSHAW* [14], *CUSHAW2-GPU* [15], *BarraCUDA* [16], *SOAP3* [17], *SOAP3-db* [18] and *nvBowtie* [19]. Other examples include FPGA and Xeon Phi implementations such as [20,21].

So far, not much effort has been made to develop tools able to exploit the characteristics of compute clusters. For instance, there is no parallel SRA implementation using workflow systems such as Swift/T [22] or SciCumulus [23]. Three examples of map-reduce based SRA aligners are *BigBWA* [11], SEAL [12] and *SparkBWA* [13], which are limited to the BWA method [9]. Regarding the message-passing paradigm, *pBWA* [9] and *pMap* [10] use MPI to distribute the reads among the processes and align the assigned reads on each process. While *pBWA* is also limited to the *BWA* aligner, *pMap* is portable enough to be able to work with several different aligners. The current publicly available version of *pMap* provides support for some popular aligners. Moreover, the source code can be modified in the case that we want to work with a new aligner. Both *pBWA* and *pMap* suffer from two major problems that limit their scalability. First, the overhead of their initial file splitting is significant, especially when increasing the number of processes. Moreover, they apply a static distribution that assigns the same number of reads to each MPI process. As the time to align one read can vary, a simple static distribution cannot achieve good load balancing.

In this paper we describe *parSRA*, a novel framework to execute short read aligners on compute clusters. Our parallel implementation overcomes the scalability issues of *pMap* thanks to:

1 A fast splitting of the input reads using the *FUSE* kernel module [24].
2 Gathering of results into a unique output file using OS commands.
3 A balanced on-demand distribution of the reads based on the shared locks of UPC++ [25].

UPC++ is an extension of C++ for parallel computing which has evolved from Unified Parallel C (UPC) [26]. PGAS languages (such as UPC, Co-Array Fortran [27] or Titanium [28]) are often easier to use than their message passing counterparts [29,30] and can also obtain better performance by using efficient one-sided communication [31–33]. UPC++ combines these advantages of the PGAS model with object oriented programming. Both UPC and UPC++ have recently been used for the parallelization of bioinformatics applications [8,34,35].

merAligner [8] is a parallel UPC-based sequence aligner for distributed-memory architectures which obtains good scalability on multicore clusters. Although this tool is also an aligner, it is focused on scenarios where the reference genome is very large and thus represented as a (distributed) collection of contigs. According to the results provided by the authors, the whole procedure (index construction and sequence mapping) is faster in *merAligner* than in *pMap*. However, the scalability of the alignment step on several nodes is lower than that of *pMap*. *merAligner* also optimizes the distribution of the genome in case that it is too large to fit in one node. However, the goal of our work is the parallelization of the type of aligners presented in the previous section, that work with genomes that fit in the memory of one node (which is typically the case for a human reference genome; the most common use case). There is no restriction related to the size of the dataset with the reads to align in *parSRA*. Furthermore, *merAligner* does not provide portability to existing aligners.

3. Implementation

The aim of *parSRA* is to accelerate the SRA while preserving the quality of the results provided by the underlying aligner. Therefore, we do not modify the source code of the aligners. Fig. 1 shows the workflow of a *parSRA* execution. The procedure starts with one process splitting the files through *FUSE* as will be explained in Section 3.1. Once all the virtual files have been created, all processes simultaneously align the reads. Each process calls the underlying aligner (*e.g., BWA-MEM* or *Bowtie2*) several times with different *FUSE* files. The assignment of virtual files to processes is described in Section 3.2. Each process writes its results into a different intermediate file (*i.e.*, there are as many intermediate files as processes). Finally, once all reads have been aligned, the results are gathered into a unique output file. This is carried out again by only one process using the OS commands to concatenate files.

All the information to perform the alignment is indicated by the user in a configuration file. One important parameter that must be included in this configuration file is the number of blocks (virtual files) that will be generated from the original input file. Increasing

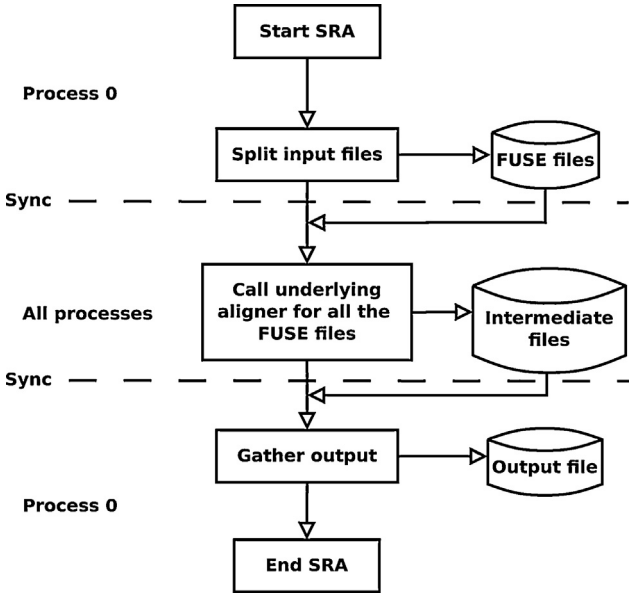

Fig. 1. Workflow of *parSRA*. All processes work in parallel to align different reads while Process 0 splits the input file and gathers the output.

Please cite this article in press as: J. González-Domínguez, et al., parSRA: A framework for the parallel execution of short read aligners on compute clusters, J. Comput. Sci. (2017), http://dx.doi.org/10.1016/j.jocs.2017.01.008