Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: 2 April 2015
Application Note

Sequence analysis

ParDRe: Faster Parallel Duplicated Reads Removal Tool for Sequencing Studies

Jorge González-Domínguez[1,*], Bertil Schmidt[2]

[1]Grupo de Arquitectura de Computadores, Universidade da Coruña, Campus de Elviña, 15071 A Coruña, Spain
[2]Parallel and Distributed Architectures Group, Johannes Gutenberg University Mainz, Staudingerweg 9, 55128 Mainz, Germany

*To whom correspondence should be addressed.

Abstract

Summary: Current next generation sequencing technologies often generate duplicated or near-duplicated reads that (depending on the application scenario) do not provide any interesting biological information but can increase memory requirements and computational time of downstream analysis. In this work we present *ParDRe*, a *de novo* parallel tool to remove duplicated and near-duplicated reads through the clustering of Single-End or Paired-End sequences from *fasta* or *fastq* files. It uses a novel bitwise approach to compare the suffixes of DNA strings and employs hybrid MPI/multithreading to reduce runtime on multicore systems. We show that *ParDRe* is up to 27.29 times faster than *Fulcrum* (a representative state-of-the-art tool) on a platform with two 8-core Sandy-Bridge processors.
Availability and Implementation: Source code in C++ and MPI running on Linux systems as well as a reference manual are available at https://sourceforge.net/projects/pardre/
Contact: jgonzalezd@udc.es

1 Introduction

The progress of Next Generation Sequencing (NGS) technologies has led to large datasets that are used in a wide range of bioinformatics applications. Preprocessing of NGS datasets is often required to either reduce their sizes or improve data quality. One such preprocessing step is the removal of duplicated and near-duplicated reads (Zhou and Rokas, 2014). There are two approaches to remove these type of reads: mapping-based and *de novo* strategies. The first approach initially maps the reads to a reference genome and discards those reads that are aligned to the same position (Pireddu *et al.*, 2011). Unfortunately, it requires a complete genome as reference, which is not always available. The *de novo* approach only needs the NGS input data and has gained attention in recent years. Examples of *de novo* tools include *FastUniq* (Xu *et al.*, 2012) (not able to remove near-duplicated reads), *Fulcrum* (Burriesci *et al.*, 2012) (parallelized for multicore and distributed systems with MapReduce) and *G-CNV* (Manconi *et al.*, 2015) (parallelized for CUDA-enabled GPUs).

In this paper we describe *ParDRe* a fast *de novo* tool to remove duplicated and near-duplicated reads with support for both Single-End and Paired-End datasets. *ParDRe* uses a novel bitwise approach to compare DNA strings and exploits the computational power of current multicore CPUs by employing both multithreading and Message Passing Interface (MPI). Mutithreading support is part of all compilers that follow the C++11 standard while there exist many MPI open public compilers.

2 Implementation

ParDRe is based on the prefix-clustering approach (Burriesci *et al.*, 2012; Manconi *et al.*, 2015), where the first l bases of a read are considered the prefix. The procedure starts by clustering all reads according to their prefix. Reads with the same prefix are stored in the same cluster and each MPI process is in charge of different clusters. All processes read the input file in parallel with efficient MPI I/O routines. For each read the processes apply a hash function to the prefix that returns a value h. If $h \ mod \ P$ (P denotes the number of processes) is equal to the process ID, it stores the read in the corresponding cluster. Otherwise, it discards the read and continues with the next one.

Once a process has finished the clustering, it compares the suffixes of the reads that belong to the same cluster. Three optimization techniques have been applied to this step. First, instead of comparing all possible pairs of reads within the cluster, we compare the first read to all other reads. We save the calculated number of mismatches for each read in an array $dist$. Subsequently, we only compare the suffixes of those reads i, j for which $|dist[i] - dist[j]|$ is less equal than the number of allowed mismatches, as otherwise we can directly conclude that reads i and j are not similar. The second optimization stores each base of the suffixes with a 4-bit encoding in an array of 64-bit integers (16 bases per array entry). Instead of comparing the suffix bases one by one, we use a novel approach that applies a bitwise `XOR` operation. This operation returns a 64-bit mask with exactly two bits equal to one for each mismatch. Then, we apply the `popcount` routine to count the number of bits equal to one. If the result divided by two is lower or equal than the number of allowed mismatches, one of the reads is removed. We keep the read with the highest average quality score among the mismatches. Finally, *ParDRe* allows to generate a second level of parallelization by creating several threads per MPI process,

Table 1. Experimental results for *ParDRe* and *Fulcrum* (v 0.43) removing near-duplicated reads of `SRR921889` on two multicore platforms. Two runtime values are presented for *ParDRe*: using only multithreading and using a hybrid MPI/multithreading approach. Speedups shown are the best *ParDRe* runtime (hybrid) over *Fulcrum*. Runtime for *G-CNV* (*) is obtained from its corresponding reference (Manconi et al., 2015).

		Two 8-core Intel Xeon E5-2660 Sandy-Bridge				Four 16-core AMD Opteron 6272				K20 GPU*
Prefix Length	Num. Mis.	Fulcrum Runtime	ParDRe threads Runtime	ParDre hybrid Runtime	Speedup	Fulcrum Runtime	ParDRe threads Runtime	ParDre hybrid Runtime	Speedup	G-CNV* Runtime
10	1	3h 11min	35min	7min	27.29	2h 36min	29min	14min	11.14	2h
10	3	4h 10min	46min	10min	25.00	3h 39min	40min	18min	12.16	1h 50min
25	1	55min	8min	4min	13.75	1h 18min	18min	7min	11.14	16min
25	3	59min	7min	4min	14.75	1h 29min	17min	7min	12.71	8min

which analyze different clusters in parallel. The assignment of clusters to threads is performed through a dynamic distribution; i.e., once a thread finishes all the comparisons within one cluster, it looks for the next cluster of the process that has not been computed yet. The main advantage of the dynamic distribution is that the workload can adapt to the size of the clusters; i.e., threads analyze more clusters if they are smaller. However, this distribution requires thread synchronization to a list in shared memory that saves which clusters have already been analyzed.

After all the clusters have been analyzed, each process writes the remaining reads into an intermediate output file (one intermediate file per process). Therefore, this read printing is performed in parallel. Once all processes have finished, *ParDRe* gathers the information of all the intermediate files into the final output (with OS routines to concatenate files) and deletes the intermediate files. It means that the output provided to the user is written into a unique file. All the configuration parameters (input and output files, prefix length l, number of allowed mismatches, number of threads per process, etc) are specified in the command line. An explanation of all the arguments, as well as installation and execution instructions, are included in the reference manual available with *ParDRe*.

3 Results

Two multicore platforms, with two 8-core Intel Xeon E5-2660 Sandy-Bridge and four 16-core AMD Opteron 6272 processors, are used to compare the runtime of *ParDRe* and *Fulcrum*. Up to our knowledge, *Fulcrum* was the fastest available tool to remove duplicate reads that allows mismatches and exploits the computational power of CPU multicore systems. *ParDRe* is compiled with GCC v4.9.2 and OpenMPI v1.8.8 on the Intel machine, whereas GCC v4.8.1 and OpenMPI v1.6.5 are used on the AMD system. *Fulcrum* runs with Python v2.6.6 on both platforms. Table 1 summarizes the runtime to remove near-duplicated reads of the dataset `SRR921889` (named after its accession number in the NCBI sequence read archive) with 50 million reads of 100 bases each. We have used four different configurations. The accuracy of the prefix-clustering approach for near-duplicate removal has been analyzed in (Burriesci *et al.*, 2012; Manconi *et al.*, 2015). We have also verified that *ParDRe* returns similar results to *Fulcrum* for those reads that do not contain N bases. Concretely, they detect the same pairs of duplicated reads but, among them, they might select a different one to discard. Therefore, our experimental evaluation focuses on the speed of the tools. *Fulcrum* is executed using one thread per core (16 threads on the Intel system and 64 threads on the AMD platform). Furthermore, two runtime values are measured for our tool in order to assess the performance improvement obtained by the use of MPI: 1) using only threads (one MPI process); and 2) using the hybrid approach with the best combination of processes and threads. We also include in our table the runtime for *G-CNV* on an NVIDIA K20 GPU, obtained from (Manconi *et al.*, 2015). The selected configurations are the same as in the *G-CNV* reference in order to provide a fair comparison.

The results show that *ParDRe* consistently outperforms *Fulcrum*. Firstly, our C++ suffix comparison based on bitwise operations is faster than the Python implementation included in *Fulcrum*. Additionally, *Fulcrum* uses MapReduce for parallelization. Thus, it needs intermediate files to distribute the clusters among threads. *ParDRe* uses an efficient on-demand multithreaded implementation that only requires main memory and avoids the overhead of I/O operations. Moreover, the experimental results also show that launching MPI processes instead of only threads further improves performance. This is due to two reasons. On the one hand, as explained in Section 2, the efficient parallel MPI I/O routines allow us to parallelize the reading and clustering of the input dataset, as well as the writing of the results. On the other hand, the hybrid approach reduces the thread synchronization overhead to know which clusters have not been analyzed yet, but it still exploits all the available cores in the machine thanks to the MPI parallelization. The average speedups over *Fulcrum* are 20.20 and 11.79 on the Intel and the AMD system, respectively. Furthermore, we can assert that *ParDRe* executed on both systems is also faster than *G-CNV* running on specialized hardware (an NVIDIA K20 GPU). Regarding the memory consumption, *ParDRe* requires less than 7GB in the worst case (for all tests in Table 1), while *Fulcrum* and *G-CNV* require 1.6 and 17.3 GB, respectively.

References

Burriesci, M. S. *et al.* (2012). Fulcrum: Condensing Redundant Reads from High-Throughput Sequencing Studies. *Bioinf*, **28**(10), 1324–27.

Manconi, A. *et al.* (2015). G-CNV: a GPU-Based Tool for Preparing Data to Detect CNVs with Read-Depth Methods. *Frontiers in Bioengineering and Biotechnology*, **3**.

Pireddu, L. *et al.* (2011). SEAL: a Distributed Short Read Mapping and Duplicate Removal Tool. *Bioinf*, **27**(15), 2159–60.

Xu, H. *et al.* (2012). FastUniq: a Fast De Novo Duplicates Removal Tool for Paired Short Reads. *PLOS One*, **7**(12).

Zhou, X. and Rokas, A. (2014). Prevention, Diagnosis and Treatment of High-Throughput Sequencing Data Pathologies. *Molecular Ecology*, **23**, 1679–1700.