Check for updates

SOFTWARE TOOL ARTICLE

REVISED QUARTIC: QUick pArallel algoRithms for high-Throughput sequencIng data proCessing [version 3; peer review: 2 approved]

Frédéric Jarlier[1-4], Nicolas Joly[5+], Nicolas Fedy[1-4,6], Thomas Magalhaes[1-4,6], Leonor Sirotti[1-4,6], Paul Paganiban[1-4,6], Firmin Martin[1-4,6], Michael McManus[7], Philippe Hupé[1-4,8]

[1]Institut Curie, Paris, F-75005, France
[2]U900, Inserm, Paris, F-75005, France
[3]PSL Research University, Paris, France
[4]Mines Paris Tech, Fontainebleau, F-77305, France
[5]Institut Pasteur, Paris, F-75015, France
[6]Université Paris Descartes, Paris, F-75006, France
[7]Intel Corporation, Hudson, Massachusetts, USA
[8]UMR144, CNRS, Paris, F-75005, France

+ Deceased author

v3 **First published:** 06 Apr 2020, **9**:240
https://doi.org/10.12688/f1000research.22954.1

Second version: 23 Jun 2020, **9**:240
https://doi.org/10.12688/f1000research.22954.2

Latest published: 08 Oct 2020, **9**:240
https://doi.org/10.12688/f1000research.22954.3

Abstract
Life science has entered the so-called 'big data era' where biologists, clinicians and bioinformaticians are overwhelmed with high-throughput sequencing data. While they offer new insights to decipher the genome structure they also raise major challenges to use them for daily clinical practice care and diagnosis purposes as they are bigger and bigger. Therefore, we implemented a software to reduce the time to delivery for the alignment and the sorting of high-throughput sequencing data. Our solution is implemented using Message Passing Interface and is intended for high-performance computing architecture. The software scales linearly with respect to the size of the data and ensures a total reproducibility with the traditional tools. For example, a 300X whole genome can be aligned and sorted within less than 9 hours with 128 cores. The software offers significant speed-up using multi-cores and multi-nodes parallelization.

Keywords
High-Throughput Sequencing, Alignment, Sorting, High-Performance Computing, MPI

Open Peer Review

Reviewer Status ✓ ✓

	Invited Reviewers	
	1	**2**
version 3 (revision) 08 Oct 2020	✓ report	✓ report
version 2 (revision) 23 Jun 2020	✓ report	✓ report
version 1 06 Apr 2020	? report	

1. **Ramon Amela Milian**, Barcelona Supercomputing Center (BSC), Barcelona, Spain

   Salvador Capella-Gutierrez, Barcelona

Supercomputing Center, Barcelona, Spain

2. **Yupu Liang**, The Rockefeller University, New York, USA

Any reports and responses or comments on the article can be found at the end of the article.

Corresponding authors: Frédéric Jarlier (frederic.jarlier@curie.fr), Philippe Hupé (philippe.hupe@curie.fr)

Author roles: Jarlier F: Conceptualization, Software, Validation, Writing – Original Draft Preparation; **Joly N**: Software; **Fedy N**: Software; **Magalhaes T**: Software; **Sirotti L**: Software; **Paganiban P**: Software; **Martin F**: Software; **McManus M**: Conceptualization; **Hupé P**: Conceptualization, Supervision, Validation, Writing – Original Draft Preparation

Competing interests: The authors declare that Institut Curie and Intel Corporation have a partnership.

Grant information: This work was supported by national infrastructure France Génomique (contract ANR-10-INBS-09) and ABS4NGS project (ANR-11-BINF-0001) funded as part of Investissements d'Avenir program managed by the Agence Nationale de la Recherche.
The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

How to cite this article: Jarlier F, Joly N, Fedy N *et al.* **QUARTIC: QUick pArallel algoRithms for high-Throughput sequencIng data proCessing [version 3; peer review: 2 approved]** F1000Research 2020, **9**:240 https://doi.org/10.12688/f1000research.22954.3

First published: 06 Apr 2020, **9**:240 https://doi.org/10.12688/f1000research.22954.1

REVISED **Amendments from Version 2**

This version improves upon the main comments raised by the reviewer2. We have added pseudo-codes for both mpiBWA and mpiSORT algorithms in two new Figures. As suggested, we added in the mpiSORT github repository additional scripts that help the user to set the computer resources that are required to sort the data depending on the characteristics of the computing cluster and the size of the file.

The reponses to the reviewer has not been added at the end of the document

Introduction

Since the first next generation sequencing technology was released in 2005 (Kchouk *et al.*, 2017), considerable progress has been made in terms of sequencing quality, diversity of protocols and throughput of the machines. As of today, the most recent generation of sequencers can easily produce terabytes of data each day and we expect this exponential growth of the sequencing to continue. This data tsunami raises many challenges, from data management to data analysis, requiring an efficient high-performance computing architecture (Lightbody *et al.*, 2019). Indeed, the throughput capacity of the sequencers tends to overwhelm the capacity of common computer architectures and the data analysis workflow to handle such amount of data in a reasonable time. As we have entered the era of genomic medicine, delivering the results to the clinicians within a short delay to guide the therapeutic decision is a challenge of the utmost importance in daily clinical practice. Several national initiatives worldwide such as France, USA, UK or Australia (Stark *et al.*, 2019) promote the use of genomics into healthcare. There is no doubt that exascale informatic architecture and software are required to tackle the challenges raised by the genomic medicine.

A typical bioinformatics workflow to analyze high-throughput sequencing (HTS) data consists of a set of systematic steps of pre-processing to i) align (or map) the sequencing reads on a reference genome and ii) to sort the alignments according to their coordinates on the genome. Those steps are fundamental for the efficiency and relevance of the downstream analysis to decipher genomic alterations such as mutations or structural variations. Traditional tools for aligning the reads are BWA-MEM (Li & Durbin, 2010) and SOAP2 (Li *et al.*, 2009a), and the sorting is usually performed with Samtools (Li *et al.*, 2009b), Sambamba (Tarasov *et al.*, 2015), Picard tools and GATK (McKenna *et al.*, 2010). Most of the time, these steps are very time consuming (up to several days for whole genome analysis) as they suffer from bottlenecks at the CPU, IO and memory levels. Therefore, removing these bottlenecks would make it possible to reduce the time-to-delivery of the results such that they could be available within a reasonable delay when very large data are produced by the sequencers.

In order to tackle the aforementioned challenges to align and sort the sequencing data, we have developed software based on the Message Passing Interface (MPI) communication protocol (Gropp *et al.*, 1996) that makes it possible to fully benefit of parallel architecture of supercomputers (Jarlier *et al.*, 2020a; Jarlier *et al.*, 2020b). MPI can therefore reduce the different bottlenecks by capitalizing on the low-latency network fabrics generally available on modern supercomputers. This allows an efficient distribution of the workload over the available resources of the supercomputers thus providing the expected scalability.

Methods

Alignment

For the alignment, the software consists of a MPI wrapper to the BWA-MEM software such that the original code remains unchanged. The wrapper uses parallel IO and shared memory (Figure 1). It first parses the input FASTQ files in order to define chunks of equal size. The different chunks are actually represented as two offsets (start and end) from the original FASTQ file thus reducing the amount of information to store. The reference genome of your choice is loaded into the shared memory on each computer node. Then, each chunk is processed in parallel threads by the original BWA-MEM algorithm. The results are finally written thanks to a shared file pointer in a unique SAM file. The pseudo-code is available in (Figure 2). For a human genome, the size of the reference genome file in the shared memory takes around 5 GB and the total memory used by a BWA-MEM thread is around 300 MB. The original BWA-MEM processes a chunk of reads at a time that contains the same number of nucleotides which is also the case with our parallel implementation in order to allow the reproducibility with the original algorithm. We name our code alignment `mpiBWA` (Jarlier *et al*., 2020a).

Sorting

For the sorting, the software implements a parallel version of the bitonic sort proposed by Batcher (1968) for sorting any sequence of elements of size $n = 2^k$ being of power of 2. Its complexity is $\mathcal{O}(n \log^2 n)$ that is higher

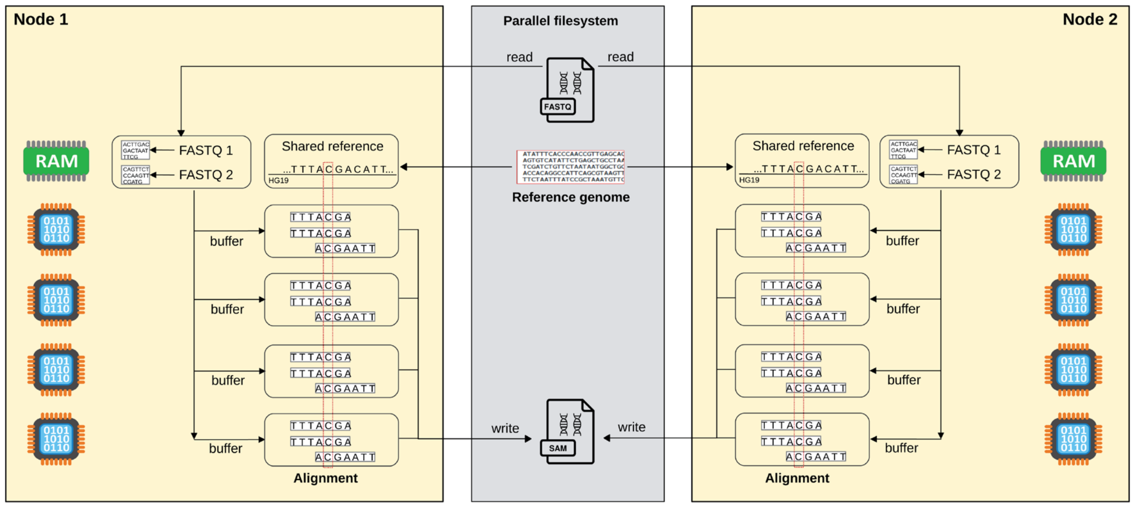


Figure 1. Alignment with `mpiBWA` using two computing nodes and four cores per node. Each core is in charge of aligning a chunk of the FASTQ file. The reference genome is stored in shared memory of each node of the computing cluster. Once aligned, the results are written into a SAM file. To ensure scalability, the read and write operations require a parallel filesystem.

```
Pseudo-code mpiBWA
  // $f$: FASTQ file
  // $g$: reference genome
  // $k$: block to be processed by the job
  // $p$: The FASTQ file is split into $p$ blocks
  // $s$: output SAM file
  function MPIBWAJOB($f, k, p, g, s$)                  ▷ Start one MPI job on the block $k$ of the FASTQ file
      $B_k \leftarrow readFASTQFile(f, k)$                ▷ Read the block $k$ of the file $f$
      $O_k \leftarrow []$                                  ▷ Initialize offsets chunks
      $O_k \leftarrow computeStartEndOffsetChunk(B_k)$    ▷ Chunks have equal number of nucleotides
      $RG \leftarrow loadRefGenomeIndexInMem(g)$          ▷ Load the reference genome index into the shared memory
      $FP \leftarrow createSAMFileSharedPointer(s)$       ▷ Create a shared file pointer on the SAM file $s$
      for each chunk $i$ in $O_k$ do                      ▷ Start the standard BWA-MEM algorithm
          $R \leftarrow getReadChunk(i, f)$               ▷ Load a chunk of reads in memory
          $A \leftarrow BWA\text{-}MEM(R, RG)$            ▷ Align the reads
          $writeToSAMFileSharedPointer(i, A, FP)$        ▷ Write results in the SAM file
      end for
      $waitAllBlocksToBeCompleted(p)$                    ▷ Wait that the $p$ MPIBWAJOB functions are completed
      $closeSAMFileSharedFilePointer(FP)$                ▷ Free pointer
      $displayStats()$                                   ▷ Write logs
  end function
  procedure MPIBWA($f, p, g, s$)                       ▷ Align the FASTQ file $f$ with $p$ processors in parallel
      MPIBWAJOB($f, 0, p, g, s$) MPIBWAJOB($f, 1, p, g, s$) ... MPIBWAJOB($f, p-1, p, g, s$)
  end procedure
```

Figure 2. Pseudo-code of the `mpiBWA` which launches the function `mpiBWAjob` in parallel over p processors.

than $\mathcal{O}(n \log n)$ from the popular merge sort algorithm. However, the bitonic sort is very suitable for parallel implementation since it always compares elements in predefined sorting network that is independent of the input data. To understand how the algorithm works, we first defined in what follows some concepts (see Grama *et al.* (2003) for details).

The algorithm relies on a *bitonic sequence* that is a sequence of values $\langle a_0, a_1, \ldots, a_{n-1} \rangle$ with the property that 1) there is an index i, $0 \leq i \leq n-1$ such that $\langle a_0, a_1, \ldots, a_i \rangle$ is monotonically increasing and $\langle a_{i+1}, \ldots, a_{n-1} \rangle$ is monotonically decreasing, or 2) there exists a cyclic shift of indices so that the condition 1) is satisfied. From any bitonic sequence $s = \langle a_0, a_1, \ldots, a_{n-1} \rangle$, the *bitonic split* operation consists in transforming the input sequence s into these two subsequences:

$$s_1 = \left\langle \min(a_0, a_{n/2}), \min(a_1, a_{n/2+1}), \ldots, \min(a_{n/2-1}, a_{n-1}) \right\rangle$$
$$s_2 = \left\langle \max(a_0, a_{n/2}), \max(a_1, a_{n/2+1}), \ldots, \max(a_{n/2-1}, a_{n-1}) \right\rangle$$

Batcher (1968) proved that both s_1 and s_2 are bitonic sequences and the elements of s_1 are smaller than the elements of s_2. Thus, a recursive bitonic split from a bitonic sequence of size $n = 2^k$, until the sequence obtained are of size one allows the sorting of the input bitonic sequence in k splits as shown in Figure 3. The procedure of sorting a bitonic sequence from bitonic split operation is called *bitonic merge*. For each split of a bitonic merge, $n/2$ comparisons are performed during which the two numbers are exchanged if not in the right order using a *compare-exchange* operation. A bitonic merge procedure of a sequence of size n is noted $\mathrm{BM}_n^{\oplus}$ if the comparisons sort the number in monotonically increasing order or $\mathrm{BM}_n^{\ominus}$ for monotonically decreasing order.

Sorting any sequence of unordered number thus requires to convert the input sequence into a bitonic sequence. This is obtained using a *bitonic sorting network* (see Figure 3) that sorts $n = 2^k$ numbers using $k - 1$ bitonic sorting stages, where the i-th stage is composed of $n/2^i$ alternating increasing $\mathrm{BM}_{2^i}^{\oplus}$ and decreasing $\mathrm{BM}_{2^i}^{\ominus}$ bitonic merges. The final stage being $\mathrm{BM}_n^{\oplus}$ to obtain the sorted sequence.

It is straightforward to generalize the bitonic sort algorithm on parallel architecture as shown in Figure 4. Each processor is in charge of an even number $m = n/p$ elements (where p is the number of processors that is a power of 2) that are first sorted using a merge sort algorithm. Then, each comparison over the bitonic sorting network is replaced by a pair of processors which performs a *compare-split* operation. During the *compare-split* operation, the two sorted sequences from each processor are merged into one monotonic sorted list, then bisected into two (lower and higher) sequences. After the *compare-split*, one processor will keep the lower m elements from sequence and the other processor will keep the higher m elements according to the direction of the arrow in the step (see Kim *et al.* (2001) & Grama *et al.* (2003) for details). This parallelization allows the sorting of n elements in $\mathscr{O}(\log^2 n)$ time using a *bitonic sorting network*.

The workflow for the sorting is described in Figure 5 and the pseudo-code is available in Figure 6. The SAM file is read and split into p blocks that are dispatched across the p processors. Each block is parsed such that for each line of the SAM file, we extract the genomic coordinates of each read, its sequence (and all other information contained in the SAM file) and the line is indexed according to its offset in the SAM memory buffer of a processor. Then, the genomic coordinates are sorted with the bitonic sort as shown in Figure 4. During the sorting, a vector with five values follows the bitonic sorting network including the genomic coordinate c, the rank r_i of

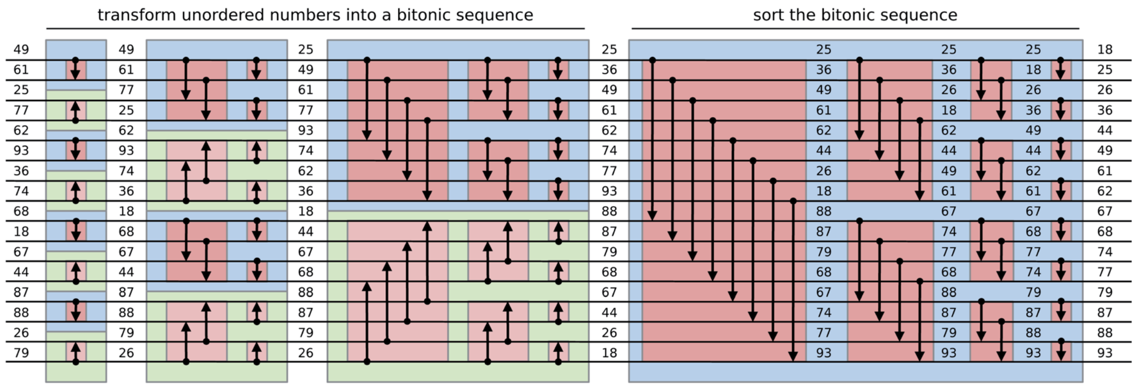


Figure 3. Sorting network for the bitonic sort. Each horizontal line represents an input and each arrow represents a two-input/two-output comparator which performs *compare-exchange* operation. Depending on its direction, the outputs are sorted in increasing (decreasing) order with a downwards (upwards) arrow. First, the comparator network transforms an input sequence of 16 unordered numbers into a bitonic sequence alternating $\mathrm{BM}_2^{\oplus}/\mathrm{BM}_2^{\ominus}$, $\mathrm{BM}_4^{\oplus}/\mathrm{BM}_4^{\ominus}$ and $\mathrm{BM}_8^{\oplus}/\mathrm{BM}_8^{\ominus}$ bitonic merges. Then, the obtained bitonic sequence is sorted with a recursive bitonic merge procedure $\mathrm{BM}_{16}^{\oplus}$ (adapted from Grama *et al.* (2003)).

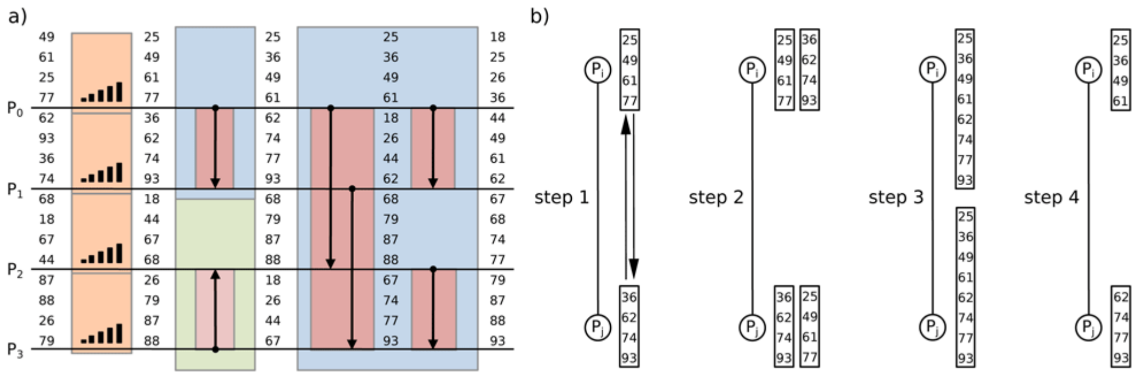


Figure 4. Parallel bitonic sort of 16 elements with four processors. (**a**) Each horizontal line represents an input of elements attached to a processor. Each processor first locally sorts its list of 4 elements, then, the parallel sort performs three steps of *compare-split* operations over the bitonic sorting network. (**b**) During the *compare-split* operation, each pair of processors first exchanges their information: each processor sends its block to the other process, then it merges the two sorted blocks into one list and stores only the appropriate half of the merged block to be used in the next step (adapted from Kim *et al.* (2001) & Grama *et al.* (2003)).

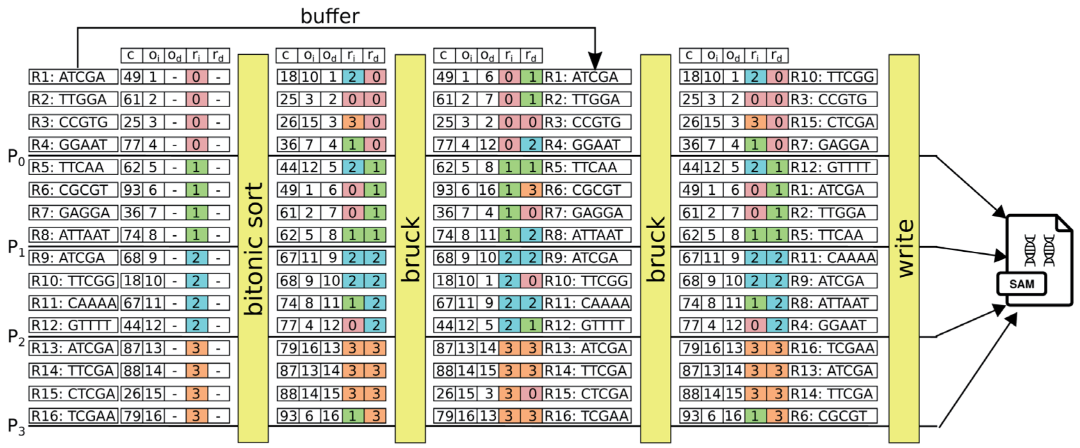


Figure 5. All-to-all communications with the Bruck algorithms for the sorting of 16 reads on four processors. The SAM file is read and split over the 4 processors P_0 to P_3 that sort the data with the bitonic sort. Then, a first Bruck phase sent the r_d and o_d values back to processor from where the data originated. The data from the SAM file are sent by a second Bruck phase to the processor that has been assigned to write the contiguous blocks (during this step, the original data is copied into the memory buffer and then exchanged). Finally, the data can be written on a parallel filesystem. *c*: genomic coordinate. r_i: rank of the processor that parsed the block of the input SAM file. o_i: offset of the line from the input SAM file. r_d: rank of the processor that will write the block of the sorted data in the destination SAM file. o_d: offset of the line in the sorted destination SAM file.

the processor that parsed the block of the input SAM file, the offset o_i of the line from the input SAM file, the rank r_d of the processor that will write the block of the sorted data in the destination SAM file and the offset o_d of the line in the sorted destination SAM file. Obviously, the values r_d and o_d are known when the bitonic sort is completed. It is important to highlight that the parallel computation is performed by different processors that are located on different compute nodes. This means that a block of data that has been read by a given processor is not accessible by another processor. Moreover, to optimize the writing of the sorted destination SAM file, it is essential to write the data in contiguous block. Thus, it is necessary that a processor in charge of the writing owns locally the data from a block of contiguous offsets o_d in the sorted destination file. This implies that all the data have to be shuffled across all the processors that have to exchange data in an all-to-all communication step. To optimize the communication between the processors during the shuffle, we have implemented a Bruck algorithm

```
Pseudo-code mpiSORT
  // $k$: block to be processed by the job
  // $p$: The SAM file is split into $p$ blocks
  // $s$: input SAM file
  function mpiSORTjob($s, k, p$)        ▷ Start one MPI job on the block $k$ (among $p$) of the SAM file $s$
      $C_k \leftarrow []$        ▷ Initialize a matrix to store read coordinates
      $S_k \leftarrow []$        ▷ Initialize a variable to store read sequences
      for each read $j$ in block $k$ do        ▷ Parse the block $k$ of the file $s$
          $seq, (c, o_i, o_d, r_i, r_d) \leftarrow extract(s, k, j)$        ▷ Extract read sequence and coordinates
          if $j$ is unmapped then
              $appendUnmapped(j)$        ▷ File for unmapped reads
          else if $j$ is discordant then
              $appendDiscordant(j)$        ▷ File for discordant reads
          else
              $C_k \leftarrow [C_k; (c, o_i, o_d, r_i, r_d)]$        ▷ Add a row to the matrix
              $S_k \leftarrow [S_k; seq]$        ▷ Add a new element
          end if
      end for
      for each chromosome $l$ do        ▷ Sort is performed chromosome by chromosome
          if $k$ is even then        ▷ Check if the processor has to sort by increasing or decreasing order
              $decreasing = TRUE$
          else
              $decreasing = FALSE$
          end if
          $C_k^l, S_k^l \leftarrow subsetChr(C_l, k)$        ▷ Select reads from chromosome $l$
          $C_k^l \leftarrow mergeSort(C_k^l, decreasing)$        ▷ Sort the block
          $C_k^l \leftarrow bitonicSort(\bigcup_{k=0}^{p-1} C_k^l)$        ▷ Communication between processors
          $C_k^l(:, "r_d") \leftarrow setDestinationRank(C_k^l)$        ▷ Modify the "$r_d$" column
          $C_k^l(:, "o_d") \leftarrow setDestinationOffset(C_k^l)$        ▷ Modify the "$o_d$" column
          $C_k^l \leftarrow exchangeOffsetsAndRankBruck(\bigcup_{k=0}^{p-1} C_k^l)$        ▷ Communication with all processors
          $B_k^l \leftarrow []$        ▷ Initialize a buffer variable with read sequences and coordinates
          $B_k^l \leftarrow mergeReadsBruck(\bigcup_{k=0}^{p-1} C_k^l, \bigcup_{k=0}^{p-1} S_k^l)$        ▷ Communication with all processors
          $\tilde{B}_k^l \leftarrow compressBlock(B_k^l)$        ▷ Compress the data
          $writeBlock(\tilde{B}_k^l)$        ▷ Each processor write its own block of data in parallel
      end for
  end function
  procedure mpiSORT($s, p$)        ▷ Sort the SAM file $s$ with $p$ processors in parallel
      mpiSORTjob($s, 0, p$) mpiSORTjob($s, 1, p$) ... mpiSORTjob($s, p-1, p$)        ▷ Launch the $p$ jobs in parallel
  end procedure
```

Figure 6. Pseudo-code of the `mpiSORT` which launches the function `mpiSORTjob` in parallel over p processors.

(Bruck *et al.*, 1997) of time complexity (log^2 p). The Bruck algorithm is performed twice as shown in Figure 5 and can be seen as a joint procedure between two tables (the elements of the table being located on different processors). During the first Bruck phase, the r_d and o_d are sent back to the corresponding processor from where the data originated, and then, during the second Bruck phase, all the data (i.e. the appropriate lines of the SAM file) are sent to the processor that has been assigned to write the data. During the second phase the original data is copied into the memory buffer and then exchanged.

Note that when the SAM file contains several chromosomes, each chromosome is sorted successively. Therefore, the upper memory bound depends on the size of the whole SAM file plus internal structures plus the size of the biggest chromosome. In this case, the total amount of memory used is around 1.5 the size of the original SAM file if the file contains reads on all the chromosomes of the human genome. If the SAM file contains only one chromosome, then the total amount of memory required is 2.5 the size of the original SAM file. To be efficient, the sorting requires a number of cores being a power of 2. We name our code for sorting `mpiSORT` (Jarlier *et al.*, 2020b).

Benchmark

We benchmarked mpiBWA and mpiSORT on the HG001 NA12878 sample from GIAB (Zook *et al.*, 2014). This sample is a whole genome with 300X depth of coverage composed of 2.13 billions 2x250 pair-ended reads. The BAM file has been downloaded from:

url: ftp://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/data/NA12878/
folder: NIST_NA12878_HG001_HiSeq_300x/NHGRI_Illumina300X_novoalign_bams/

The BAM file has been converted into FASTQ files using bedtools bamtofastq. In order to test the scalability of our software with respect to the size of the data, we downsampled the original 300X sample to obtain FASTQ files corresponding to depths of coverage ranging from 28X to 300X with a geometric progression with a common ratio of 1.6. The sequences have been aligned on the human genome GRCh38.

We ran the benchmark on a computing cluster equipped with Intel Xeon® Gold 6148 CPU @ 2.40GHz (Skylake architecture). The nodes are interconnected with Intel® Omni Path Architecture (OPA) of 100 Gbps speed. The parallel file system is BeegFS with two servers. We compile the programs with GCC 8.3 and use Open MPI 3.1.4. Jobs have been submitted using slurm scheduler.

Implementation

The code has been written in C programming language using MPI directives. mpiBWA encapsulates the original BWA-MEM version 7.15. Two implementations for mpiBWA exist: the first outputs one SAM file with all the chromosomes, the second (named mpiBWAByChr) outputs one SAM file per chromosome. Moreover, mpiBWA comes along with mpiBWAIdx, which is responsible for creating a binary image of the reference genome. This image is subsequently loaded in the shared memory by mpiBWA. Then, every mpiBWA process on the same computing node will share the same genome reference in order to save memory usage. mpiBWAIdx does not need MPI to run.

Operation

As our software rely on the MPI standard, mpirun must be available to run the program. Several MPI implementations exist such as mpich, open-mpi or Intel® MPI Library.

Use cases

The Figure 7 shows the scalability of both mpiBWA (with the mpiBWAByChr implementation) and mpiSORT with varying sample sizes (from 28X to 300X) using computation distributed over 128 cores. The output files from mpiBWAByChr have been used as input by mpiSORT. Both algorithms efficiently scale as the input data are bigger, the walltime to process the data being proportional to their input size. We assessed how much time was spent on pure computation versus IO (*i.e.* reading the input files and writing the output files): mpiBWA spent more than 95% of its walltime in computation while mpiSORT spent between 50% to 60% in IO. The walltime to analyze the biggest sample of 300X is less than 8h hours for the alignment and less than one hour for the sorting.

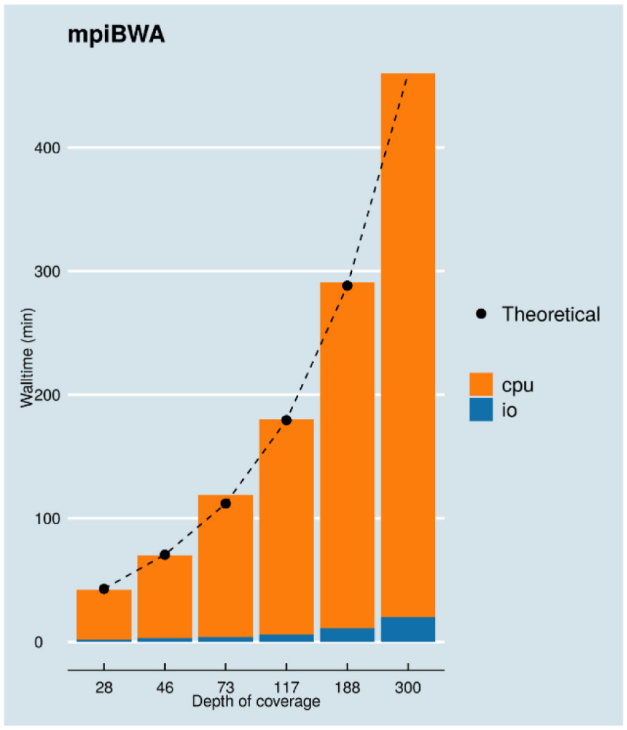
The Figure 8 compares the performance of both mpiBWA (with the mpiBWAByChr implementation) and mpiSORT with varying number of cores and nodes with respect to the classical tools BWA-MEM version 7.15 and samtools version 1.10. On a single node, the walltimes are very similar between BWA-MEM and mpiBWA whatever 8 or 16 threads are used with BWA-MEM to process the 28X sample. With mpiBWA, increasing the number of cores, either on the same node or over multiple nodes (from 2 to 8) allowing the distribution of the computation up to 128 cores, shows a linear scalability (the walltime is divided by 2 when the number of cores is doubled) that follows the expected theoretical walltimes. For the sorting, mpiSORT is very efficient since it offers a speed-up of 6.4 over samtools using 8 threads (or cores) on a single node to process the SAM file of the chr16 from the 300X sample. Doubling the number of threads did not change the walltime with samtools. With mpiSORT, increasing the number of cores, either on the same node or over multiple nodes, shows a linear scalability.

mpiBWA

The first step consists in building the binary image of the reference genome that will be used in the shared memory by mpiBWA:

```
mpiBWAIdx hg19.small.fa
```

This creates the file myReferenceGenome.fa.map.



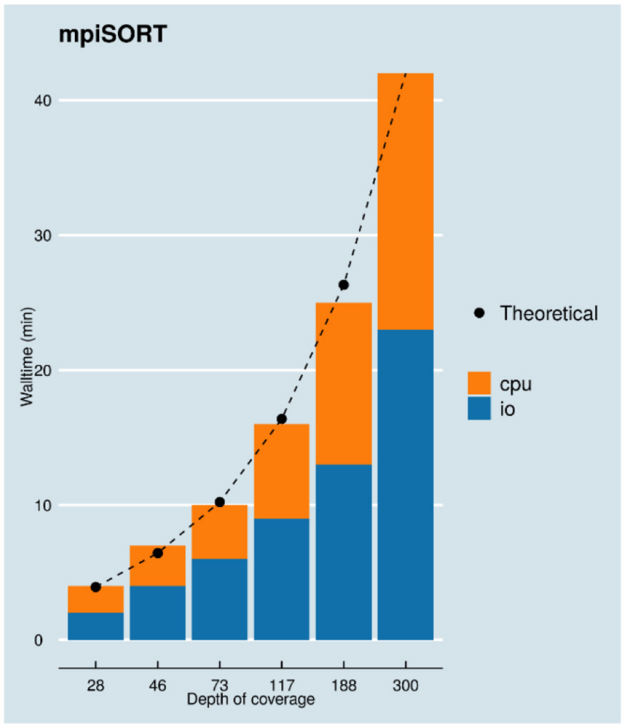


Figure 7. Walltimes for the alignment and sorting varying the sample sizes. The FASTQ files are first aligned with `mpiBWAByChr`, SAM files are generated as outputs and then sorted with `mpiSORT` and `-u` option. 128 cores have been used over 16 and four nodes with `mpiBWA` and `mpiSORT`, respectively. The black dots represent the theoretical values of the walltime with respect to the 300X depth of coverage being set as the reference.

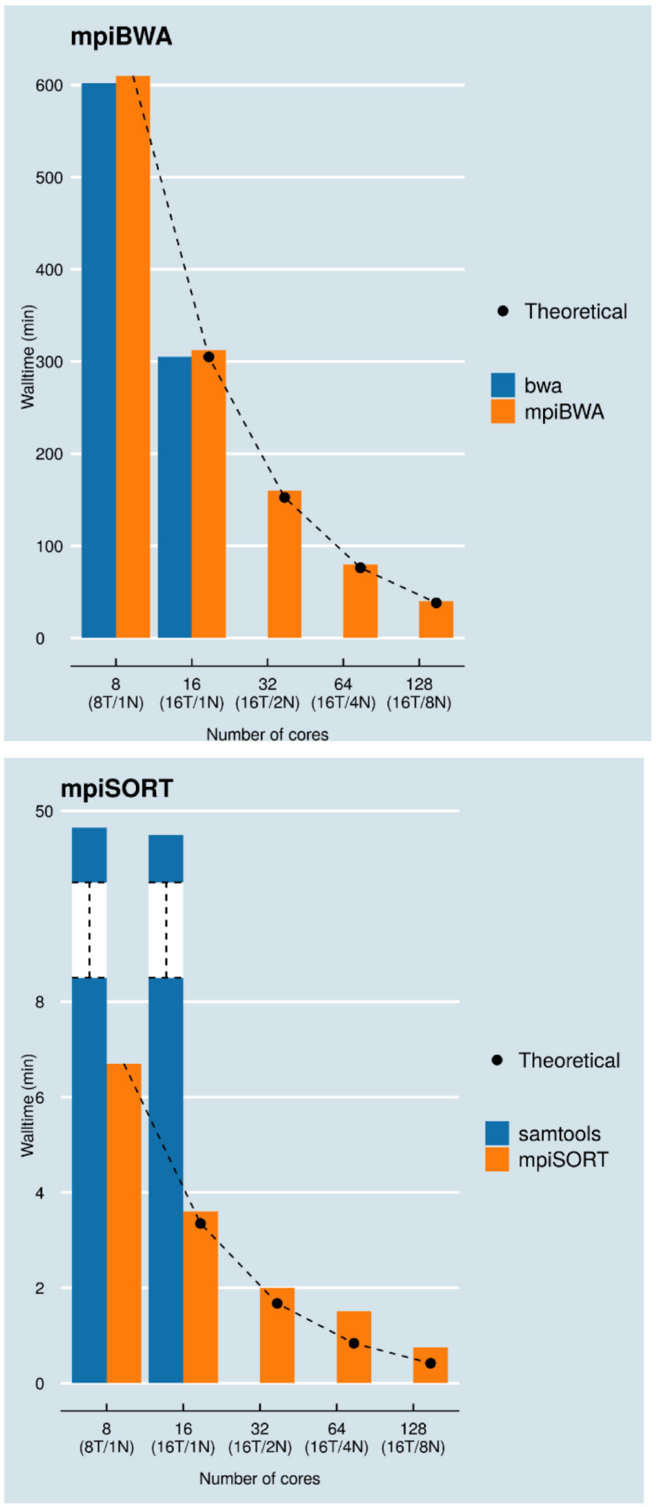


Figure 8. Walltimes for the alignment of the 28X depth of coverage sample and sorting of the chr16 from the 300X depth of coverage sample varying the number of cores. The number in brackets indicates how many threads (T) on how many nodes (N) have been launched, the number of cores used being the product of the two values. The black dots represent the theoretical values of the walltime with respect to the 8 cores value of the MPI implementation being set as the reference.

Importantly, `mpiBWAIdx` requires the following files that are generated by BWA index:

- `myReferenceGenome.fa.sa`
- `myReferenceGenome.fa.bwt`
- `myReferenceGenome.fa.ann`
- `myReferenceGenome.fa.pac`
- `myReferenceGenome.fa.amb`

The `.map` file only needs to be generated once. Then, `mpiBWA` is executed with the `mpirun` program, for example:

```
mpirun -n 2 mpiBWA mem -t 8 -o ./HCC1187C.sam hg19.small.fa \
    HCC1187C_R1_10K.fastq \
    HCC1187C_R2_10K.fastq
```

This command launches two MPI processes with eight BWA threads each (16 cores will be used). If you want to split the results of the alignment by chromosome, use `mpiBWAByChr`, for example:

```
mpirun -n 2 mpiBWAByChr mem -t 8 -o ./HCC1187C.sam hg19.small.fa \
    HCC1187C_R1_10K.fastq \
    HCC1187C_R2_10K.fastq
```

The total memory used during the alignment if approximately the size of the `.map` file plus the size of the SAM chunks loaded by each BWA tasks. A BWA thread takes around 300 MB of memory.

mpiSORT

From the results obtained with `mpiBWAByChr`, each SAM files can be sorted as follows:

```
mpirun -n 4 mpiSORT -u chr1.sam ${HOME}/mpiSORTExample
```

This command launches four MPI processes.

As `mpiSORT` requires a the entire input SAM file to be loaded into the memory, the program is memory bounded. Therefore, a lot of attention has to be paid to define how many cores are needed to process the data. For example, let's assume that the computing cluster consists of nodes with 190 GB of RAM memory with 40 cores each (thus 4.75 GB per core is available but this value can be rounded to the lower limit of 4.5 GB to leave some free memory on the node). To choose the number of cores, the following rule has to be applied: the total memory for sorting a SAM file that contains only one chromosome is around 2.5 times the size of the SAM file. For instance, sorting a chr1.sam file of size 209 GB with 4.5 GB per core requires 128 cores, for a of size 110 GB it requires 64 cores. We remind that the bitonic sort algorithm requires a number of cores that is a power of 2, this is the reason why the number of cores has to be set to the upper bound to the closest power of 2.

Conclusion

In this paper, we described parallel algorithms (Jarlier *et al.*, 2020a; Jarlier *et al.*, 2020b) to process sequencing data that fully benefit from high-performance architecture with a total reproducibility and linear speed-ups. Our implementation is based on bitonic sorting network, Bruck algorithm and IO optimizations with MPI directives. MPI efficiently removes POSIX barriers like the one-file per process or thread concurrent access. Indeed, this differs from other implementations using embarrassingly parallel approaches (Decap *et al.*, 2015; Kawalia *et al.*, 2015; Puckelwartz *et al.*, 2014) that rely on the MapReduce paradigm: in this case, input files are first split into small chunks, each one being analyzed by a process, and the different output files are then merged into a single output. With MPI, splitting the input file is not necessary since it can efficiently handle concurrent access on a single file.

The scalability of the software strongly relies on the underlying hardware architecture, such as the low latency interconnection and parallel filesystem. This implies that the hospitals or institutions need a powerful state-of-the-art informatic architecture that is either available locally or provided with mutualized resources through national infrastructures that are certified to process healthcare data. Interestingly, our implementation allows the generation of aligned read files for each chromosome that can be further sorted in parallel thus reducing the time of downstream analysis whenever an analysis per chromosome is possible. The performance we obtained showed that MPI is very relevant for the field of genomics. The tools we developed pave the way towards the use of whole genome sequencing in daily clinics in order to meet the deadline and deliver results to the clinician in real-time for precision medicine as the time to delivery can be reduced to several minutes if the processing is distributed over several hundreds of cores.

Data availability

All data underlying the results are available as part of the article and no additional source data are required.

Software availability

Source code for mpiBWA available at: https://github.com/bioinfo-pf-curie/mpiBWA.

Archived source code at time of publication: https://doi.org/10.5281/zenodo.3887185 (Jarlier *et al*. (2020a)).

Source code for mpiSORT available at: https://github.com/bioinfo-pf-curie/mpiSORT.

Archived source code at time of publication: http://www.doi.org/10.5281/zenodo.4061917 (Jarlier *et al*. (2020b)).

All documentation is available at: https://github.com/bioinfo-pf-curie/QUARTIC.

License: CeCILL Version 2.1.

Author contributions

F.J. and N.J. developed and tested `mpiBWA`. F.J., N.F., L.S., T.M, P.P. and F.M. developed and tested `mpiSORT`. M.M. provided technical expertise and access to computing cluster facilities to benchmark the code. F.J. coordinated the developments. F.J. and P.H. wrote the manuscript. P.H. supervised the study.

Acknowledgments

We are very grateful to the national infrastructure *France Génomique* for providing us with an access to their computing cluster and we especially thank Claude Scarpelli as the head of its informatics department. We thank our colleagues François Prud'homme and Sébastien Astruc from the ICT department and Marc Deloger from the Bioinformatics Core Facility at Institut Curie. We also thanks Intel Corporation collaborators for fruitfull discussions and help.

References

Batcher KE: **Sorting networks and their applications**. In: *Proceedings of the April 30–May 2,1968, spring joint computer conference*. 1968; 307–314.
Publisher Full Text

Bruck J, Ho CT, Kipnis S, *et al.*: **Efficient algorithms for all-to-all communications in multiport message-passing systems**. *Parallel and Distributed Systems, IEEE Transactions on*. 1997; **8**(11).
Publisher Full Text

Decap D, Reumers J, Herzeel C, *et al.*: **Halvade: scalable sequence analysis with mapreduce.** *Bioinformatics.* 2015; **31**(15): 2482–2488.
PubMed Abstract | Publisher Full Text | Free Full Text

Grama A, Karypis G, Kumar V, *et al.*: **Introduction to Parallel Computing**. Addison-Wesley, second edition, 2003. ISBN 0201648652 9780201648652.
Reference Source

Gropp W, Lusk E, Doss N, *et al.*: **A high-performance, portable implementation of the MPI message passing interface standard.** *Parallel Computing.* 1996; **22**(6): 789–828.
Publisher Full Text

Jarlier F, Joly F, Hupé P: **bioinfo-pf-curie/mpibwa: version-1.1**. 2020a.
http://www.doi.org/10.5281/zenodo.3887185

Jarlier F, Joly N, Magalhaes T: **bioinfo-pf-curie/mpisort: version-1.4.** 2020b.
http://www.doi.org/10.5281/zenodo.4061917

Kawalia A, Motameny S, Wonczak S, *et al.*: **Leveraging the power of high performance computing for next generation sequencing data analysis: tricks and twists from a high throughput exome workflow.** *PLoS One.* 2015; **10**(5): e0126321.
PubMed Abstract | Publisher Full Text | Free Full Text

Kchouk M, Gibrat JF, Elloumi M: **Generations of sequencing technologies: from first to next generation.** *Biology and Medicine.* 2017; **9**(3).
Reference Source

Kim YC, Jeon M, Kim D, *et al.*: **Communication-efficient bitonic sort on a distributed memory parallel computer**. In: *Proceedings. Eighth International Conference on Parallel and Distributed Systems. ICPADS 2001*. IEEE. 2001; 165–170.
Publisher Full Text

Li H, Durbin R: **Fast and accurate long-read alignment with burrows-wheeler transform.** *Bioinformatics.* 2010; **26**(5): 589–595.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

Li H, Handsaker B, Wysoker A, *et al.*: **The sequence alignment/map format and samtools.** *Bioinformatics.* 2009a; **25**(16): 2078–2079.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

Li R, Yu C, Li Y, *et al.*: **Soap2: an improved ultrafast tool for short read alignment.** *Bioinformatics.* 2009b; **25**(15): 1966–1967.
PubMed Abstract | **Publisher Full Text**

Lightbody G, Haberland V, Browne F, *et al.*: **Review of applications of high-throughput sequencing in personalized medicine: barriers and facilitators of future progress in research and clinical application.** *Brief Bioinform.* 2019; **20**(5): 1795–811.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

McKenna A, Hanna M, Banks E, *et al.*: **The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data.** *Genome Res.* 2010; **20**(9): 1297–1303.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

Puckelwartz MJ, Pesce LL, Nelakuditi V, *et al.*: **Supercomputing for the parallelization of whole genome analysis.** *Bioinformatics.* 2014; **30**(11): 1508–1513.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

Stark Z, Dolman L, Manolio TA, *et al.*: **Integrating genomics into healthcare: A global responsibility.** *Am J Hum Genet.* 2019; **104**(1): 13–20.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

Tarasov A, Vilella AJ, Cuppen E, *et al.*: **Sambamba: fast processing of NGS alignment formats.** *Bioinformatics.* 2015; **31**(12): 2032–2034.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

Zook JM, Chapman B, Wang J, *et al.*: **Integrating human sequence data sets provides a resource of benchmark SNP and indel genotype calls.** *Nat Biotechnol.* 2014; **32**(3): 246–51.
PubMed Abstract | **Publisher Full Text**

Open Peer Review

Current Peer Review Status: ✓ ✓

Version 3

Reviewer Report 12 October 2020

https://doi.org/10.5256/f1000research.30033.r72713

✓

Ramon Amela Milian
Computational Genomics. Life Sciences Department, Barcelona Supercomputing Center (BSC), Barcelona, Spain

Salvador Capella-Gutierrez
Spanish National Bioinformatics (INB) Coordination Node, Barcelona Supercomputing Center, Barcelona, Spain

We don't have further comments.

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: bioinformatics

We confirm that we have read this submission and believe that we have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Reviewer Report 09 October 2020

https://doi.org/10.5256/f1000research.30033.r72714

✓

Yupu Liang
Research Bioinformatics, CCTS, The Rockefeller University, New York, NY, USA

I am satisfied with the response and have no further comments.

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: bioinformatics and medical informatics

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Version 2

Reviewer Report 13 August 2020

https://doi.org/10.5256/f1000research.27364.r67518

✓ **Yupu Liang**
Research Bioinformatics, CCTS, The Rockefeller University, New York, NY, USA

Jarlier and co-authors developed an efficient method of short reads mapping and sorting through utilizing Message Passing Interface(MPI) . Specifically, they implement an MPI wrapper to the well-known mapper BWA-MEM existing method, named mpiBWA. They then implement a parallel version of the sorting method called mpiSORT through the implementation of bitonic sort. The authors have shown mathematically that their sorting method is �� (log2*n*) time. They have also done benchmark experiments to compare their software to BWA and samtools. They demonstrated that their tools could archive linear scalability by adding computing cores to the system. The authors did a great job of explaining the process through figures. They have also provided sample command lines to use the software in action. I do wish the authors would could also provide pseudo-code of their implementation, which will help readers to understand the process better. Another minor point is that given the software's performance strongly relies on the underlying hardware architecture, and it would also be helpful for the authors to provide helper script(s) that help to calculate the number of cores to use on particular data input and computing infrastructure.

In conclusion, the authors demonstrate that using bitonic sorting network, Bruck algorithm and IO optimizations with MPI directives can improve the performance of sequencing data handling.

Is the rationale for developing the new software tool clearly explained?
Yes

Is the description of the software tool technically sound?
Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow

replication of the software development and its use by others?
Partly

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?
Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: bioinformatics and medical informatics

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Author Response 02 Oct 2020

Philippe Hupé, Institut Curie, Paris, France

First, we would like to thank the second reviewer. We are very grateful for her time and very valuable comments that significantly helped to improve the article and the documentation of the tools.

You will find below a detailed answer to the different points that we have addressed in the revised manuscript.

Best regards,

Frédéric Jarlier and Philippe Hupé

Detailed response to the reviewer2's comments

> I do wish the authors would could also provide pseudo-code of their implementation, which will help readers to understand the process better.

Indeed to facilitate the reading and understanding of both mpiBWA and mpiSORT algorithms, we added 2 figures with the pseudo-code.

> Another minor point is that given the software's performance strongly relies on the underlying hardware architecture, and it would also be helpful for the authors to provide helper script(s) that help to calculate the number of cores to use on particular data input and computing infrastructure

In order to help the user when writing submission scripts to a computing cluster, we added two scripts in the example folder of the github repository of mpiSORT.
These scripts allow the users to explore the different partitions of the computing cluster (getSlurmNodesInfo.sh) and give advices in term of resources needed to run the MPI program (informaticResources.py).
We also updated the benchmarking section of the documentation with examples.

Competing Interests: No competing interests were disclosed.

Reviewer Report 14 July 2020

https://doi.org/10.5256/f1000research.27364.r65398

✓

Ramon Amela Milian
Computational Genomics. Life Sciences Department, Barcelona Supercomputing Center (BSC), Barcelona, Spain

Salvador Capella-Gutierrez
Spanish National Bioinformatics (INB) Coordination Node, Barcelona Supercomputing Center, Barcelona, Spain

Thanks to the authors for addressing point-by-point each comment. We have no further comments to make.

Is the rationale for developing the new software tool clearly explained?
Yes

Is the description of the software tool technically sound?
Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?
Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?
Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?

Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: bioinformatics

We confirm that we have read this submission and believe that we have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Version 1

Reviewer Report 04 May 2020

https://doi.org/10.5256/f1000research.25341.r62078

?

Ramon Amela Milian
Computational Genomics. Life Sciences Department, Barcelona Supercomputing Center (BSC), Barcelona, Spain

Salvador Capella-Gutierrez
Spanish National Bioinformatics (INB) Coordination Node, Barcelona Supercomputing Center, Barcelona, Spain

Jarlier and colleagues introduce their work on parallelizing popular bioinformatics programs, which have become the bottleneck for the current scenario of massive NGS data generation even in clinical settings.

Authors describe two MPI implementations for the alignment and sorting steps used as part of the NGS data preprocessing in order to better take advantage of parallel architectures. Algorithms are well described and the documentation is good, making really easy the reproducibility of their results [1]. In addition, authors provide software containers (singularity), which can be used in controlled environments including HPC installations.

Despite the existing benchmarking efforts carried by authors, a scalability study is missing. Ideally, benchmarking should include varying resources allocation so readers can benefit from such comparison. We would like to suggest, as you have done with varying deep coverage sizes, you make the scalability study through a varying number of processors e.g. 4, 8, 16, 32, 64, 128. It would be also interesting to use as a baseline the normal execution of those programs to illustrate the improvement of the MPI implementation regarding the standard approaches. This way, it would be possible to answer the following questions:

- Is the MPI version faster when using a single node?

- Otherwise, from which point is better to use the MPI version than the most common software. I.e. in case the MPI is 2 times slower but scales linearly, the execution time would be shorter when giving the MPI implementation at least 2 nodes.

This way, the potential users would know better the scenarios in which using this implementation becomes more interesting.

You have mentioned there are similar approximations to this problem. However, you just mention them in the conclusions. Wouldn't it make sense to discuss similarities and differences with other implementations in the text?

Apart from the benchmarking component, it would be interesting to further discuss the scenarios where implementations like this one will be used. It is not clear that clinical settings will have internally very powerful computational facilities for their daily use.

Finally, we would like to mention some minor points related to the text itself.

Abstract: Moving from the general problem to the specific solution without mentioning that the NGS data preprocessing (alignment and sorting) represent major bottlenecks to timely results deliver for diagnostic use.

Methods:

- **Alignment**. reference genome file > indicate a version.

- **Sorting section**. Apparently there is a miswriting of the theoretical computational cost of the algorithm > O (n log2 n) that is higher than O (n log n).

Minor:

- **Introduction**: data each days > data each day.

- **Introduction**: align and sort the sequencing data, we developed software > align and sort sequencing data, we **have** developed software.

- **Sorting section**: by a pair of processors with performs a compare-split operation > by a pair of processors **which** performs a compare-split operation.

References

1. Reproducibility of Results. 2020. Reference Source

Is the rationale for developing the new software tool clearly explained?

Yes

Is the description of the software tool technically sound?

Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?

Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?
Partly

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?
Partly

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: bioinformatics

We confirm that we have read this submission and believe that we have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however we have significant reservations, as outlined above.

Author Response 12 Jun 2020

Philippe Hupé, Institut Curie, Paris, France

First, we would like to thank the reviewers. We are very grateful for their time, contribution and very valuable comments that significantly helped to improve the article and the documentations of the tools.

You will find below a detailed answer to the different issues that we have addressed in the revised manuscript.

Best regards,

Frédéric Jarlier and Philippe Hupé

Detailed response to the reviewer1's comments

> Despite the existing benchmarking efforts carried by authors, a scalability study is missing. Ideally, benchmarking should include varying resources allocation so readers can benefit from such comparison. We would like to suggest, as you have done with varying deep coverage sizes, you make the scalability study through a varying number of processors e.g. 4, 8, 16, 32, 64, 128.

Indeed, this benchmark varying the number of cores was clearly missing. Therefore, a scalability study has been added for both mpiBWA and mpiSORT using 8, 16, 32, 64, 128 cores. The results of this benchmark are described in the section "Use cases" and in Figure 6. We took this opportunity of this new benchmark to use the last version (1.1) of our MPI implementations (Figure 5have been therefore updated).

> It would be also interesting to use as a baseline the normal execution of those programs

to illustrate the improvement of the MPI implementation regarding the standard approaches.

As suggested, we also compared our MPI implementations with respect to a reference baseline using the traditional tools (bwa and samtools). The walltimes of the reference baseline are presented in the section "Use cases" and in Figure 6.

> This way, it would be possible to answer the following questions:
>
> Is the MPI version faster when using a single node?
>

The results show that the walltimes are similar between mpiBWA and bwa. mpiSORT is much faster than samtools offering a speed-up over 6.

> Otherwise, from which point is better to use the MPI version than the most common software. I.e. in case the MPI is 2 times slower but scales linearly, the execution time would be shorter when giving the MPI implementation at least 2 nodes.
>

The results of the scalability benchmark and the comparison with respect to the traditional tools demonstrate that both mpiBWA and mpiSORT perform efficiently on a single node and on multiple nodes.

> This way, the potential users would know better the scenarios in which using this implementation becomes more interesting.
>

The results of the scalability benchmark show that the MPI implementation can be very versatile.

Our MPI implementations can easily address the two typical scenarii:

- when time-to-delivery matters, the scalability allows the processing of the data very quickly using multiple cores and nodes.
- when the throughput matters (i.e. number of samples that can be analyzed at the same time), you can process a sample on a single node.

We also added a benchmark section in the documentation of the github repository such that the user can reproduce the benchmark to figure out what is the best scenario according to the computing infrastructure that is used. The documentations on the github repository describe how to assess the memory and cpu usage for the traditional tools and the MPI implementations as we did for the scalabitity study. The documentation is available here:

- https://github.com/bioinfo-pf-curie/mpiSORT/blob/master/docs/README.md#benchmark
- https://github.com/bioinfo-pf-curie/mpiBWA/blob/master/docs/README.md#benchmark

> You have mentioned there are similar approximations to this problem. However, you just mention them in the conclusions. Wouldn't it make sense to discuss similarities and differences with other implementations in the text?

We added in the conclusion that:

Indeed, this differs from other implementations using embarrassingly parallel approaches (Puckelwartz et al., 2014; Decap et al., 2015; Kawalia et al.,2015} that rely on the MapReduce paradigm: in this case, input files are first split into small chunks, each one being analyzed by a process, and the different output files are then merged into a single output. With MPI, splitting the input file is not necessary.

> Apart from the benchmarking component, it would be interesting to further discuss the scenarios where implementations like this one will be used. It is not clear that clinical settings will have internally very powerful computational facilities for their daily use.
>

We agree that the access to a powerful and up-to-date computing infrastructure may be a bottleneck. We added in the conclusion of manuscript:

This implies that the hospitals or institutions need a powerful state-of-the-art informatic architecture that is either available locally or provided with mutualized resources through national infrastructures that are certified to process healthcare data.

> Finally, we would like to mention some minor points related to the text itself.
>
> Abstract: Moving from the general problem to the specific solution without mentioning that the NGS data preprocessing (alignment and sorting) represent major bottlenecks to timely results deliver for diagnostic use.

We agree that the Abstract needed to be improved. It has been substantially modified in the revised version.

> Methods:
>
> Alignment. reference genome file > indicate a version.
>

We added in the text "*The reference genome of your choice*" as the user can use any genome.

> Sorting section. Apparently there is a miswriting of the theoretical computational cost of the algorithm > O (n log2 n) that is higher than O (n log n).

The complexity are correct. Its is O (n log2 n) for the bitonic sort, it is higher that other sort algorithms but it can be easily parallelized.

>
> Minor:
>
> Introduction: data each days > data each day.
>
> Introduction: align and sort the sequencing data, we developed software > align and sort sequencing data, we have developed software.
>
> Sorting section: by a pair of processors with performs a compare-split operation > by a pair of processors which performs a compare-split operation.
>

The typos have been corrected.

Competing Interests: No competing interests were disclosed.