The International Journal of High Performance Computing Applications
XX(X):1–13
©The Author(s) 0000
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

A Framework for Genomic Sequencing on Clusters of Multicore and Manycore Processors

Héctor Martínez[1], Sergio Barrachina[1], Maribel Castillo[1], Joaquín Tárraga[2], Ignacio Medina[3], Joaquín Dopazo[2], and Enrique S. Quintana-Ortí[1]

Abstract

The advances in genomic sequencing during the past few years have motivated the development of a fair number of fast and reliable software for DNA/RNA sequencing on current high performance architectures. Most of these efforts target multicore processors, a few only can also exploit graphics processing units (GPUs), and a much smaller set will run in clusters equipped with any of these multi-threaded architecture technologies. Furthermore, the examples that can be used on clusters today are all strongly coupled with a particular aligner.

In this paper we introduce an alignment framework that can be leveraged to coordinately run any "single-node" aligner, taking advantage of the resources of a cluster, without having to modify any portion of the original software. The key to our transparent migration lies in hiding the complexity associated with the multi-node execution (such as, e.g., coordinating the processes running in the cluster nodes) inside the generic-aligner framework. Moreover, following the design and operation in our MPI version of HPG Aligner RNA BWT, we organize the framework into two stages in order to be able to execute different aligners in each one of them. With this configuration, for example, the first stage can ideally apply a fast aligner to accelerate the process, while the second one can be tuned to act as a refinement stage that further improves the global alignment process with little cost.

Keywords

Genomic Sequencing, DNA-seq and RNA-seq, High Performance Computing, Clusters, Multi-threaded Architectures

1 Introduction

New DNA sequencing technologies, also known as next generation sequencing (NGS), have unleashed an unprecedented revolution in biology. In particular, high-throughput sequencers have turned transcriptome sequencing into a matter of days, instead of years, with other costs also in continuous decrease. Among the numerous applications of NGS, DNA sequencing (DNA-seq) is widely employed because of its important clinical implications. Concretely, DNA-seq aims to unveil the causes of rare diseases and, ultimately, to improve medical care by inspecting the genome of individual patients (Biesecker 2010). In addition, RNA sequencing (RNA-seq) has become a key analysis tool for biological and clinical research, replacing conventional expression microarrays in most practical scenarios (Garber et al. 2011). In more detail, RNA-seq is currently applied to quantify the expression of genes that are activated/repressed by a disease, which helps to understand the etiology of a disease.

In the last years, there has been a continuous flow of research to develop fast and reliable software for genomic sequencing on ("single-node") computers equipped with multi-threaded processors. Among efforts from others, in Tárraga et al. (2014) we introduced a DNA-seq aligner for multicore architectures; and in Martínez et al. (2013); Martínez et al. (2015b) we adapted the DNA pipeline to deal with the specifics of RNA-seq on multicore sockets. Following a common principle of computer architecture,

[1]Department of Computer Science and Engineering, University Jaume I, 12006–Castellón, Spain
[2]Genomics Institute, Research Center Príncipe Felipe, 46012–Valencia, Spain
[3]HPC Services, University of Cambridge, CB2 1TN–Cambridge, United Kingdom

Corresponding author:
Héctor Martínez, Department of Computer Science and Engineering, University Jaume I, 12006–Castellón, Spain.
Email: martineh@uji.es

all these aligners "*make the common case fast*", applying a simple yet inaccurate method in order to rapidly map DNA/RNA fragments (reads), while relying on a costlier but more accurate approach to tackle more difficult cases.

More recently, in Martínez et al. (2015a), we ported one of our RNA-seq aligners to operate on clusters of computer nodes equipped with multicore processors. In that MPI version, the aligner is re-organized into a series of computational stages, that each MPI rank (process) can work on independently of processes running in other nodes. At the end of each stage, the local information produced up to that point is compiled into global structures and broadcasted, to be leveraged by all processes in subsequent stages. Proceeding in this asynchronous manner, the MPI version of the RNA-seq aligner delivers fair scalability when mapping a file of 80 million RNA reads on a cluster with up to 12 nodes and almost 100 processor cores. Moreover, the aggregation of global information on splice junctions offers a sensitivity close to that of the original single-node (multi-threaded) aligner.

The operation of our MPI aligner, with independent stages interleaved with communication of local information, is general and can also be applied to other single-node aligners for DNA/RNA-seq. Following this idea, in this paper we propose a *"generic-aligner" framework for DNA/RNA sequencing* on clusters of multicore processors. In doing so, we make the following specific contributions:

- The alignment framework offers a skeleton consisting of two stages, which can accommodate a number of existing aligners for either DNA/RNA-seq, taking care, among others, of the distribution of the data, the partitioning of the work, the exchanges of information among the processes, and the compilation of the results into standard output files.
- The aligners to execute at each stage of the framework can be selected by the user among a varied collection of single-node multi-threaded software. No modification is required in the aligner code. The use of different aligners at distinct stages, or the same aligner with different options, yields a research tool to improve the efficiency, scalability and/or sensitivity of the global alignment process.
- We investigate the scalability of different DNA and RNA aligners on a cluster of multi-threaded processors. Furthermore, for RNA-seq we expose a clear improvement of accuracy by refining the results from an initial aligner with a second stage with a more reliable mapper.

The rest of the paper is structured as follows. In Section 2 we briefly introduce a few basic concepts in genomic sequencing. In Section 3 we provide a short survey of parallel software for genomic sequencing, and we describe the operation of our multi-threaded HPG Aligner. In Section 4 we first revisit the MPI version of HPG Aligner, to then present the "generic-aligner" framework for DNA/RNA-seq. In Section 5 we experimentally evaluate the scalability and sensitivity of different combinations of aligners on Maverick, a cluster from Texas Advanced Computing Center (TACC). Finally, in Section 6 we summarize our work and the key insights gained from this study.

2 Genome Sequencing

NGS technology is currently applied to sequence short DNA/RNA reads, generally comprising between 50 to 400 nucleotides (nts), or longer with more recent sequencers. These NGS tools produce data consisting of hundreds of millions, or even billions of reads. The first step for genome re-sequencing consists in mapping the reads onto a reference genome, with the purpose of locating the genomic coordinates of each read (Langmead et al. 2009). This computational step is highly expensive and sensible to errors. In particular, sensitivity is a serious concern due to natural variations or error sequencing, which yields frequent mismatches between the reads and the reference genome, in general increasing the computational cost of the mapping procedure (Li and Homer 2010).

The mapping stage is particularly difficult for RNA-seq. The genes in eukariotes may be split into small regions known as exons, which are separated by intron zones composed of thousands of nucleotides. When the exons are transcribed into the RNA, they are brought together to form the transcripts in a splicing process. Special care is thus needed when mapping reads from RNA transcripts onto a reference genome, as some reads may contain a splice junction and, therefore, involve exons which lay thousands of nucleotides apart (gapped alignment).

3 High Performance Software for DNA/RNA Alignment

There exists a considerable variety of software packages for computational DNA- and RNA-seq, many of them based on matching techniques and structures such as the Burrows-Wheeler transform (BWT) (Burrows and Wheeler 1994), the *full-text index in minute space* (FM-index) (Ferragina and Manzini 2000), the suffix array (SA) (Manber and Myers 1993), or the Smith-Waterman algorithm (SWA) (Smith and Waterman 1981). In rough details, BWT is a technique to rearrange a character string into segments of similar characters. SA is a sorted array (index) containing all possible suffixes of a string, with memory requirements considerably higher than those of BWT, but usually faster operation. FM-index is a compressed full-text substring index based on BWT, which

Table 1. Short list of DNA/RNA alignment software.

	Aligner (last update)	Target	Open
DNA Aligners	Bfast (2011)	Multicore	Yes
	Blat (2012)	-	Yes
	Bowtie (2015)	Multicore	Yes
	Bowtie2 (2015)	Multicore	Yes
	BWA (2014)	Multicore	Yes
	BWA Mem (2014)	Multicore	Yes
	Cushaw2-GPU (2013)	Multicore/GPU	Yes
	Cushaw3 (2014)	Multicore	Yes
	ContextMap2 (2015)	Multicore	Yes
	GEM (2013)	Multicore	No
	Gnumap (2011)	Multicore	Yes
	Halvade (2015)	Multicore/Cluster	Yes
	HPG Aligner DNA SA (2015)	Multicore	Yes
	Soap (2007)	Multicore	Yes
	Soap2 (2011)	Multicore	No
	Soap3-dp (2011)	Multicore/GPU	Yes
RNA Aligners	HPG Aligner RNA BWT (2014)	Multicore	Yes
	HPG Aligner RNA SA (2015)	Multicore	Yes
	MapSplice (2011)	Multicore	Yes
	Cluster HPG Aligner RNA BWT (2015)	Multicore/Cluster	Yes
	Olego (2015)	Multicore	Yes
	RNASEQR (2012)	Multicore	Yes
	RUM (2015)	Multicore/Cluster	Yes
	SpliceMap (2010)	Multicore	Yes
	STAR (2015)	Multicore	Yes
	TopHat (2012)	Multicore	Yes
	TopHat2 (2015)	Multicore	Yes

can be used to efficiently find the number of occurrences of a pattern within the compressed text as well to as locate the position of each occurrence. Finally, SWA quantifies the similarity between (regions of) two strings, comparing segments (i.e, substrings) of all possible lengths.

Table 1 provides a survey of recent high performance aligners, including those targeted in our work, for multi-threaded architectures. Among other aspects, the table exposes that most of these software packages are designed to exploit the hardware of a single-node system, usually a server equipped with multicore processors and, possibly, a GPU. Halvade, RUM and our MPI extension of HPG Aligner can also exploit all the resources in a multi-node cluster. Halvade relies on Hadoop MapReduce 2.0. During the map phase, each task independently processes a chunk of the input data and produces intermediate pairs. Next, all these intermediate pairs are sorted in parallel accordingly to their key. During the reduce phase, each task independently processes a single key and its corresponding values. RUM processes reads in three phases. First, it uses Bowtie to map reads into the genome; next, it re-applies Bowtie to map onto a transcriptome database; and finally it relies on Blat to map onto the genome. The information from the three mappings is finally merged into one mapping. RUM can be run in parallel on a Sun Grid Engine-based cluster. It splits the input in chunks so that each node applies the previous procedure to its chunks, and finally merges the results produced by all the nodes. The use of this software in other types of clusters requires the creation of shell scripts or an extension of the RUM source code. With respect to our MPI extension of HPG Aligner, it is later described in Section 4.1. Compared with the generic-aligner framework for clusters presented in this work, those three cluster aligners are strongly coupled with/can only apply their own mapping algorithm whereas our aligner framework can run any single-node alignment software.

In order to illustrate the flexibility of our multi-node sequencing framework for clusters, we selected Cushaw2-GPU (Liu and Schmidt 2014), Cushaw3 (Liu et al. 2014), Bowtie2 (Langmead and Salzberg 2012), and HPG Aligner DNA SA (Tárraga et al. 2014) for DNA-seq. The three first aligners perform the mapping via the BWT or an FM-index. Cushaw2-GPU accelerates the mapping process via inter-task hybrid CPU-GPU parallelism and a tile-based SWA for CUDA GPUs, while Cushaw3 and Bowtie2 run on multicore processors. HPG Aligner DNA SA relies on an SA array as well as a software pipeline organization of the procedure tailored for multicore processors.

In addition, we chose Tophat2 (Kim et al. 2013), MapSplice2 (Wang et al. 2010), HPG Aligner RNA BWT (Martínez et al. 2013), STAR (Dobin et al. 2013), and HPG Aligner RNA SA (Martínez et al. 2015b) for RNA-seq. Among these, the first three aligners exploit the BWT while the last two rely on SA. All of them commence by splitting the reads into short segments (seeds), to then map these fragments onto reference genome. In Tophat2, MapSplice2 and HPG Aligners RNA (BWT and SA), the initial aligning stage is followed by a re-mapping to identify split alignments in the presence of small exons. All the cases selected for RNA-seq run on multicore processors.

3.1 Multi-threaded HPG aligners

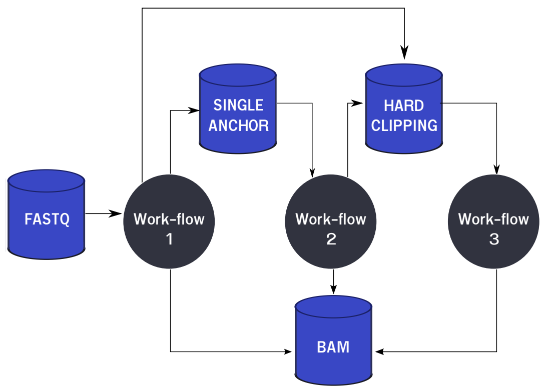

Figure 1. Work-flow organization of HPG Aligner RNA BWT.

Our family of high-performance aligners for multi-threaded architectures are organized into a number of consecutive workflows. For RNA-seq in particular, the first workflow of HPG Aligner RNA BWT maps the reads onto the reference genome, and separates those reads for which a reliable mapping could not be found into two groups, "single anchor reads" and "hard clipping reads", to be processed by Workflows 2 and 3, respectively. The first group contains those reads that were partially mapped, while the remaining ones are placed in the second group. This process is graphically illustrated in Figure 1.

HPG Aligner RNA BWT relies on a series of matching data structures that store information about "meta-exons" and splice junctions. These data are updated with knowledge from those reads which were accurately mapped during the process. The alignment structures are then leveraged by subsequent workflows to speed up as well as improve the accuracy of the mapping for difficult reads.

From the operation perspective, HPG Aligner RNA BWT relies on the BWT to perform a fast initial alignment, allowing up to one error or indel (insertion or deletion) per read. If a read is successfully mapped, this stage creates an *alignment record* and updates the meta-exon structure. Those reads which could not be mapped are split into several seeds, and the BWT is re-applied to map each seed into the reference genome, resulting in a potentially large collection of *candidate regions*. Finally, the SWA is employed to align pending reads using certain candidate regions as potential mapping targets. HPG Aligner RNA SA basically differs from this in that an SA index, instead of the BWT, is used for the initial mapping.

As argued, HPG Aligners RNA (BWT and SA) are both divided into three consecutive workflows connected via queues that act as data buffers and synchronize the relative processing throughputs of the (threads running on the) workflows. Each stage of the workflow "receives" a collection of reads (or *batch*), tries to map its contents, and "sends" the result to the next stage as a batch to be processed there. Threads running in different (CPU) cores process batches concurrently, leveraging the inter-stage concurrency intrinsic to the operation.

On the other side, in case there exists a large number of CPU cores, our scheme also exploits the concurrency implicit to each one of the stages. Concretely, by running more than one thread per stage, several batches can be simultaneously processed in the same stage (intra-stage concurrency). The purpose of exploiting intra-stage concurrency is to accelerate/balance the processing speeds of the different stages, as the throughput of each pipeline is dictated by the performance of the slowest stage.

4 Genomic Sequencing in Clusters

The motivation for our framework is the MPI version of the HPG Aligner RNA BWT (Martínez et al. 2015a). We next briefly describe that solution before presenting the generic-aligner framework developed for this work.

4.1 HPG Aligner RNA BWT for clusters

The MPI extension of HPG Aligner RNA BWT is organized as three consecutive workflows, with the same division of work conceived for the single-node version; see Figure 1.

To commence, the reads are statically distributed among the MPI ranks/processes that were spawned in order to perform the task. (Hereafter, we will assume that there is one process per cluster node.) For this purpose, one process computes the initial/final reads of the file which correspond to each node, and broadcasts this information to the remaining processes/nodes. Upon receiving this, each process reads its portion of the input file into the local RAM. For the class of input files appearing in RNA-seq (frequently stored in FASTQ format (Cock et al. 2010), and usually with more than 50 million reads of 100–400 nts each), in our experiments we found that this "multi-reader" approach is often faster than a solution which reads and distributes the contents of the input file from a single node; see Martínez et al. (2015a).

Once the workload has been distributed, each MPI rank independently executes the first workflow and stores the output on local files. These results comprise the unmapped reads, divided into single-anchor and hard clipping read files, a meta-exon structure, and the detected splice junctions; see Martínez et al. (2015a) for details.

In parallel with the execution of the first workflow, the structures local to each process are merged, via message-passing, into global structures in order to aid in the mapping of elusive reads during the subsequent workflows. This information is fused following an inter-node minimum-spanning tree communication scheme, as illustrated in Figure 2 for an example involving 10 nodes.

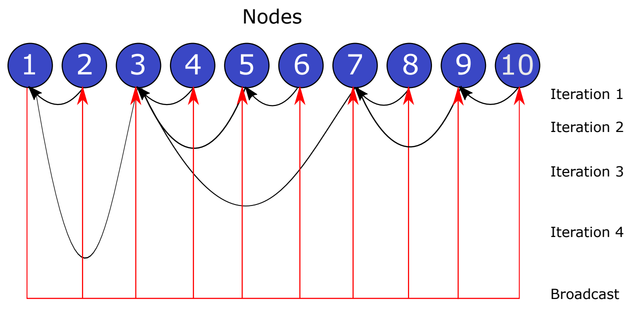

Figure 2. Message-passing fusion of the meta-exon and detected splice junctions structures.

The MPI ranks continue next with the second workflow, independently processing their local "single anchor reads" file; update their local versions of the meta-exon and detected splice-junctions structures; and add those reads which could not be correctly mapped to their local "hard-clipping reads" file. A merging process of the meta-exon and detected splice junctions structures proceeds concurrently with this workflow.

Finally, the processes execute the third workflow, operating on their local "hard-clipping reads" file; and the local detected splice-junctions structures are merged and subsequently written to a file on disk in BED format.*

4.2 Overview of the generic-aligner framework

The generic-aligner framework is composed of two stages connected via a temporary buffer that stores those reads which could only be partially mapped or not mapped at all during the first stage. The second stage aims to re-map those "difficult" reads using information generated from the first stage, and generally, a different mapper.

Each stage of the framework can execute any choice among a number of mappers (see Section 3), leading to a rich variety of combinations. For example, for RNA-seq, the framework can apply Tophat2 during the first stage, and next refine the unsatisfactory results (i.e, unmapped reads or mapped reads but with a low confidence) with HPG Aligner RNA SA.

In short, the framework performs the following tasks:

- spawns the number of MPI ranks indicated by the user;
- splits and distributes the reads among the MPI ranks;
- executes the aligner(s) selected by the user in the cluster nodes at the appropriate execution time;
- if necessary, creates the appropriate connection buffers to pass information from the first to the second stage;
- implements the communications among MPI ranks including, if needed, the merge of intermediate results and the final output; and
- synchronizes the termination.

Thus, the generic-aligner framework performs the same tasks as the MPI version of HPG Aligner RNA BWT, with a slightly different organization. However, we emphasize that our new solution has the appealing advantage of being mostly "agnostic" of the specific aligners that operate within the framework. Compared with this, MPI HPG Aligner RNA BWT (for RNA-seq), Halvade (for DNA-seq) and RUM (for RNA-seq) cannot be decoupled from a concrete aligner.

Although our previous parallelization of HPG Aligner RNA BWT for clusters inspired the new generic-aligner framework, the code for the latter was developed almost from scratch, due to its different design principles.

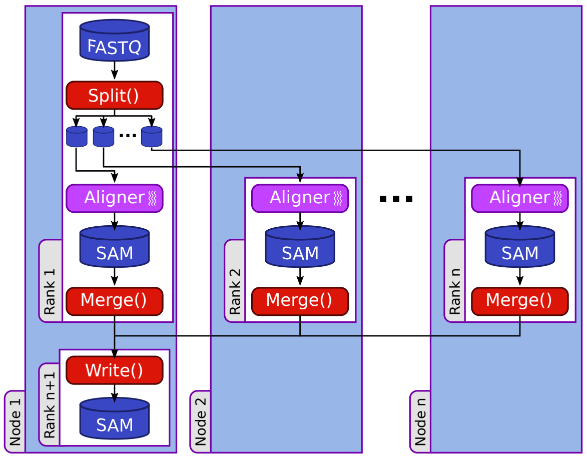

Figure 3. Operation of the generic-aligner framework (without a second stage). The aligner underlying each MPI rank is executed after its part of the input dataset has been written to disk. The merge procedure in each MPI rank is executed after the aligner in its rank has completed its task.

Concretely, the MPI extension of HPG Aligner RNA BWT deals with the communication of the data structures that are required or produced by each of the HPG Aligner RNA BWT workflows and, therefore, is tightly integrated into the mapper code, which was developed with this specific purpose in mind. In other words, the MPI version is a message-passing application for RNA alignment that relies on MPI primitives for communication. Compared with this, in the generic-aligner framework for DNA/RNA, processes running in different nodes of a cluster also exchange data via MPI. However, the mapping process is performed by MPI ranks (see Figure 3), each of which invokes for this purpose a single-node (sequential or multi-threaded) aligner code, possibly developed by others (e.g., Bowtie 2, STAR, MapSplice 2, etc.). The framework then is in charge of collecting/combining the results from this individual executions to produce a global result. In addition, there are three other implementation differences:

- The MPI extension of HPG Aligner RNA BWT does an initial *logical* partition of the work, and broadcasts the initial and final reads assigned to each MPI rank. Conversely, the generic-aligner framework *physically* splits the work into separate files, one per MPI rank.

- Each MPI rank of the MPI extension of HPG Aligner RNA BWT sends its local results to a global writer process. Compared with this, the generic-aligner framework collects and processes the output

*`http://genome.ucsc.edu/FAQ/FAQformat.html#format1`

information generated by the single-node mappers invoked by each MPI rank.

- The MPI extension of HPG Aligner RNA BWT deals with the type of data that the first stage from the same aligner produces. Compared with this, the framework has to be able to process the output from a variety of aligners employed in the first stage, transforming these data in order to produce the appropriate information to pass to the second stage.

4.3 Details of the generic-aligner framework interface and operation mode

The alignment framework is invoked by specifying the following information:

- number of stages to execute (1 or 2);
- type of sequencing problem to solve (DNA or RNA);
- specific aligner(s) to use at each stage (and configuration options);
- names of the input files containing the reference genome and reads;
- directory for intermediate results; and
- name of the output path.

For example, the command:

```
mpirun -np n+1 -hosts host1,host2,...,hostn \
       ./hpg-multialigner rna \
       -c "tophat2 -o %O -p 16
               --no-convert-bam
                 index_bwt/homo_sapiens %I" \
       -i index_hpg/ \
       -f reads.fq
       -o final-output/ \
       --tmp-path /tmp/output/
       --second-phase
```

launches the alignment framework with $n + 1$ MPI ranks on n nodes, to perform RNA sequencing using TopHat2 in the first stage (with the configuration parameters specified in the string following the command name, "`-o %O ... %I`" for `tophat2`). The remaining parameters in the command line specify that the reference genome is in `index_hpg`; the input file containing the reads is in `reads.fq`; and the output and temporary files are to be stored in `final-output/` and `/tmp/output`, respectively. In this case, the second stage is activated (`--second-phase`) which, by default, invokes our HPG Aligner RNA SA.

After this, n processes become "workers", and will take care of executing the aligner selected for the first stage, while the remaining "writer" process will be in charge of compiling the outputs from this stage. As the writer process is mapped to the same node as one of the workers, only n nodes are actually used. Among the workers, one queries the size of the input read file, calculates the portion of the dataset that will be mapped by each worker (including itself), and splits the dataset creating one file per worker, containing the portion assigned to each.

Next, each worker reads the appropriate file into the local RAM, and processes its contents (reads) via the selected aligner, taking advantage of the hardware resources present in the local node (multicore or GPU, depending on the aligner). The partial results from the local alignment (that is, per worker) are stored into a temporary buffer, allocated in a shared or local hard drive, using the SAM format.

Once the execution of the alignment process selected for the first stage is complete, the local results are merged into a global file. For this purpose, each worker sends the contents of its own SAM file to the writer, which combines this information into a single global file.

Starting from this point, the next steps differ, depending on the type of sequencing problem (DNA or RNA) and whether a second stage was requested or not.

DNA. In case the user selected a single stage, the alignment procedure is complete and no further action is necessary. Otherwise, those reads which remained unmapped after the first stage are re-processed with the aligner selected for the second stage. This refinement is performed locally to each process (worker), and proceeds following analogous steps to those applied during the first stage: apply the selected aligner exploiting the local resources, store the results in a temporary file, and send them to be combined by the writer into the final output file.

RNA. After the first stage of RNA-seq, the (mapped reads and) splice junctions identified by the aligner lay in local files which need to be merged into a single file. Unfortunately, different aligners specify splice junctions using distinct formats at the end of this stage. In order to produce a consistent result, our alignment framework determines the splice junctions directly from the alignment information in the local SAM files, storing the outcome into separate local files using our own format. To complete the execution of the first stage, these structures are merged, and a single file containing all splice junctions is generated.

In case a second stage is due, a filtering process such as that described for RNA is performed. Moreover, when the second stage is to be done via our HPG Aligner, before this process commences, the alignment framework creates a global meta-exon structure to improve the mapping sensitivity. For this purpose, each process creates its local meta-exon and this information is exchanged to build a global database.

4.4 Optimization of the generic-aligner framework

In subsection 4.1 we referred that the MPI HPG Aligner RNA BWT splits the input file among the MPI worker ranks by *i)* computing and broadcasting the start/end position that will be read by each worker, and *ii)* leveraging this information from each worker to read its part of the shared input file. In addition, the MPI aligner overlaps the execution of the mapping procedure with the merge of the results generated by the workers. These two features, retrieval of dataset from the original file and overlapped merge/mapping, were easy to implement because, in that case, there exists a strong coupling between the MPI version of HPG Aligner RNA BWT and the underlying aligner.

The generic alignment framework employs, by default, a more cautious (aligner-agnostic) approach to tackle these two issues in order to accommodate any aligner. First, there is only one process in charge of splitting the input dataset and creating separate files for each worker. Second, all workers must complete their work and synchronize before merging those results generated by a stage. As argued, this approach presents the strong advantage that it works for any aligner, as its operation mode mimics the way this generic aligner expects its "environment" to behave. In particular, the aligner reads an input, which corresponds to an actual file, and dumps its output into a file with no concurrent access from other processes.

Unfortunately, this flexible implementation of the generic-aligner framework embeds two major drawbacks: first, it introduces a serialization point due to the split and creation of the original dataset into separate files (one per worker), which involves a copy, and is performed by a single process. Second, it creates a synchronization point between the execution of the first (and second) stage(s) and the fusion of this information. Thus, in case the time required to split-and-copy the dataset and/or merge the intermediate information is comparable to the cost of the mapping procedure, the scalability of the alignment framework will be compromised.

Although it is considerably more difficult than in the case of the MPI HPG Aligner RNA BWT, the generic-aligner framework can also integrate a "parallel" split (with 0-copy on disk) of the input file as well as a merge that proceeds concurrently with the mapping. These two improvements will fit most aligners, and can be optionally activated when the actual cost of the mapping process is small, yielding significant gains in such scenario.

Our current implementation enhances the alignment framework with both optional features. From the interface point of view, the command line option '`--fifo-enable`' instructs the framework to "parallelize" the distribution of the input file. Furthermore, the command line option '`--fast-merge`' forces an overlapped merge with the actual processing. When these options are selected, the processing scheme is reorganized as described in the following (see also Figure 4).

Let us assume the user activates both options. Each MPI rank then creates a *reader* thread to handle the parallel file distribution, plus a *merger* thread to attain a concurrent execution of the merge stage(s).

To distribute the retrieval of the input file, one of the MPI ranks computes the start/end positions that correspond to each worker and broadcasts this information. The *reader* thread in each worker then creates a FIFO file (also known as a named pipe) in its local RAM, and streams its part of the input, from the original file, into the local FIFO file. Upon invocation, the name of the FIFO file is also passed to the aligner executable (via its command line), which simply retrieves its input from there. Proceeding in this manner, the input file is read concurrently by the *reader* threads, running at the MPI ranks, which split the dataset at the same time. Furthermore, using this procedure, there is no need to modify the source code of the aligners.

As part of the merge process, the *merger* thread at each MPI rank waits until a local output file is created. As soon as this event is detected, the *merger* commences to read blocks from there. When this thread has retrieved enough data to fill a buffer, it sends this information to the global writer running in MPI rank $n + 1$. The *merger* proceeds in this manner till the mapping process in its local MPI rank has been completed, and the information on the corresponding output files has been streamed to the global writer.

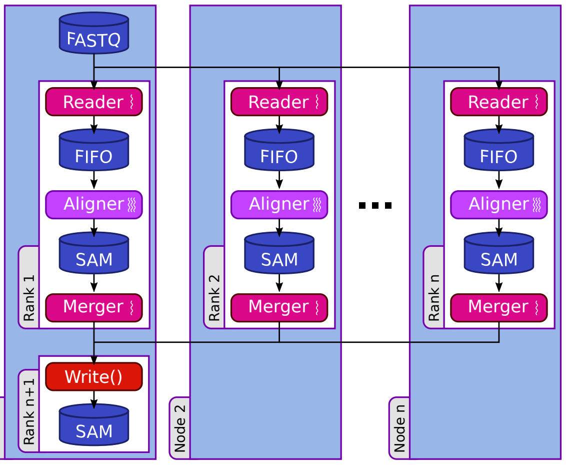

Figure 4. Operation of the generic-aligner framework (without a second stage), and with "parallel" retrieval of the input file and concurrent merge of the local results with the mapping process. The aligner underlying each MPI rank is executed as soon as the FIFO in its rank has been created. The merger thread on each MPI rank is started as soon as the output files of the aligner in its rank are created.

Table 2. Command lines used to execute each aligner.

	Mapper	Command Line
DNA	Bowtie 2	`bowtie2 -x indexB2/hs -p 16 -S ./out/align.sam data.fq`
	Cushaw2-GPU	`cushaw2-gpu -r indexC2/hs.fa -f data.fq -t 16 -o ./out/align.sam`
	Cushaw3	`cushaw3 align -r indexC3/hs.fa -f data.fq -t 16 -o ./out/align.sam`
	HPG Aligner DNA SA	`hpg-aligner dna -i indexSA/ -f data.fq --cpu-threads 16 -o ./out/`
RNA	HPG Aligner RNA BWT	`hpg-aligner rna -i indexBWT/ -f data.fq --cpu-threads 16 -o ./out/`
	HPG Aligner RNA SA	`hpg-aligner rna -i indexSA/ -f data.fq --cpu-threads 16 -o ./out/`
	MapSplice 2	`mapsplice.py -c indexM/ -x indexM/hs -1 data.fq -p 16 -o ./out/`
	STAR	`STAR --genomeDir indexST/ --readFilesIn data.fq --runThreadN 16 --outFileNamePrefix ./out/`
	Tophat 2	`tophat2 -p 16 --no-convert-bam --no-sort-bam -o ./out/ iT/hs data.fq`

5 Scalability and sensitivity

In this section we describe the setup for the experimentation, and evaluate the performance and sensitivity of the alignment framework under different configurations.

5.1 Experimental setup

The following experiments were performed using the cluster Maverick at TACC. Each cluster node integrates two 10-core Intel Xeon E5-2680 v2 "Ivy Bridge" CPUs, 256 GB of RAM, and an NVIDIA Tesla K40 "Atlas" GPU with 12 GB G-RAM connected via PCI-e Gen 2. The cluster interconnect employs Mellanox FDR Infiniband technology, and the same network allows the compute nodes to gain access to a 20 PB Lustre parallel file system.

The generic-aligner framework code, which is available at https://github.com/opencb/hpg-aligner, branch `develop-multi-fifo-output`, was compiled with GNU `gcc` version 4.4.7, using the package default compile options. The MPI version used was MVAPICH2 2.0b.

The experimental evaluation of the generic-aligner framework for DNA was performed using a single-end dataset consisting of 40 million reads of 100 nts, generated with the dwgsim simulator from the SAMtools (Li et al. 2009). The mutation rate was set to 0.1%, with 10% of these being indels (option '`-R 0.1`'). We will refer to this dataset as `D40M`. The experimental evaluation of the alignment framework for RNA was performed using three single-end datasets, two of them consisting of 10 million reads and the remaining one with 80 million reads, with reads of 100 nts in all cases. These were generated with the beers simulator (Grant et al. 2011). The mutation rate was set to 0.1% for one of the 10-million datasets (`R10M0.1`) and the 80-million one (`R80M0.1`); this rate was raised to 2% for the second 10-million dataset (`R10M2.0`). The indel frequency was fixed to the default value (0.05%) for all three datasets. These datasets can be downloaded from http://lorca.act.uji.es/dataset/2016_ijhpca/.

When our HPG Aligner (BWT or SA) is applied in the second stage of the framework, its execution parameters are fixed as follows: the minimum and maximum intron dimensions were set to 40 and 500,000 nts, respectively; the minimum CAL size to 20 nts; and the configuration for the (EMBOSS) SWA to its default values (match: 5, mismatch: -4, gap open: 10, and gap extend: 0.5).

The experiments with the small datasets were repeated a number of times (around 10), revealing little variability in the results. The large experiments were also executed in a smaller/slower cluster confirming the small variations between repeated executions. Unfortunately, these cases were too expensive to be repeatedly executed on a production facility.

Table 3. Execution time (in seconds) of the multi-threaded aligners, using a single node of Maverick, applied to `D40M` and `R10M0.1`.

Target	Mapper	Time
DNA/`D40M`	Bowtie 2	732
	Cushaw2-GPU	926
	Cushaw3	2146
	HPG Aligner DNA SA	367
RNA/`R10M0.1`	HPG Aligner RNA BWT	65
	HPG Aligner RNA SA	66
	MapSplice 2	1470
	STAR	60
	Tophat 2	1715

5.2 Execution time and parallel performance

In order to assess the performance of the generic-aligner framework, we first evaluate the selected multi-threaded aligners, listed in Table 2, using 16 threads on a single node of Maverick. (In a separate experiment, we determined that 16 was the optimal number of threads for most aligners.) The specific command line parameters for this experiment are reported in the same table.

The results in Table 3 show that the execution time of the DNA aligners ranges from 367 to 2,146 seconds, and the cost of the RNA aligners varies between 60 and 1,715 seconds. While we note here the large differences between STAR/HPG Aligner and the remaining software, we emphasize that this initial evaluation does not intend to compare the performance of these single-node multi-threaded aligners. In particular, the results will surely change if a different dataset is used, or in case the aligner parameters are fine-tuned for a particular dataset. Instead, we emphasize that the purpose of this evaluation is to ensure a fair analysis of the alignment framework. Concretely, our experiments are conceived to expose whether our framework is able to reduce the execution time of a generic aligner, by efficiently exploiting the resources in a cluster, while preserving the sensitivity (accuracy) of the single-node mapping process.

Figure 5 reports the speed-ups attained with the framework configured to execute the first stage only, using the selected DNA aligners and up to 12 nodes (with 16 threads per node). Figures 6 and 7 offer the results for an analogous experiment with the RNA aligners and the `R10M0.1` and `R80M0.1` datasets, respectivelly. In all executions in this section, the aligners were invoked from the framework with the command line parameters listed in Table 2. Moreover, unless otherwise stated, the parallel read of the input dataset and the concurrent merge were not enabled (see subsection 4.4).

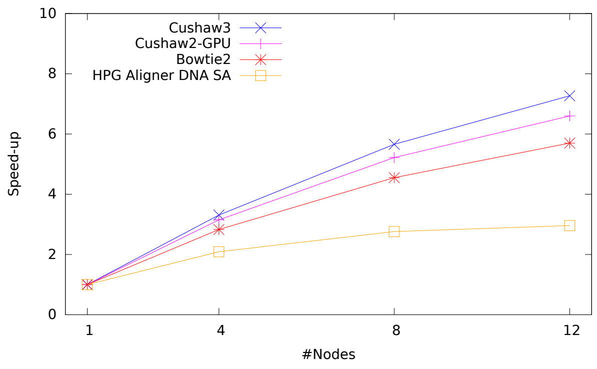

Figure 5. Speed-up for the first stage of the generic-aligner framework invoked to apply the DNA aligners to `D40M`.

Let us focus on the results reported for the RNA-seq problem in Figure 6. (The trends shown in the figure for DNA-seq and the motivation are similar.) The results when the framework executes the RNA-aligners show a moderate speed-up for MapSplice 2 and Tophat 2, but scarcely any for STAR and the HPG Aligner variants. In order to determine whether this was a problem of scale, we ran the framework with STAR and HPG Aligner on `R80M0.1`. Unfortunately, Figure 7 reveals that, even with this much larger dataset, the speed-ups were nearly nonexistent.

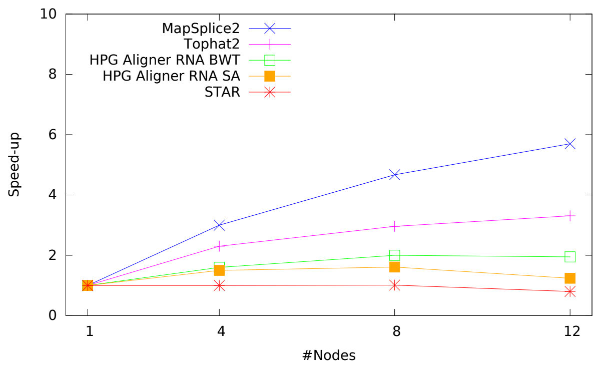

Figure 6. Speed-up for the first stage of the generic-aligner framework invoked to apply the RNA aligners to `R10M0.1`.

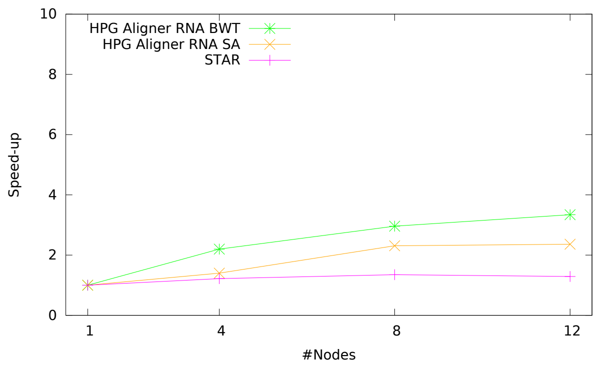

Figure 7. Speed-up for the first stage of the generic-aligner framework invoked to apply the RNA aligners to `R80M0.1`.

To gain some insight on why these aligners present such a poor scalability, when embedded into the framework, we conducted a finer-grain time analysis of HPG Aligner RNA BWT. The break down of costs considered four phases: split-and-copy of the input dataset into one file per worker, load of the index, mapping, and merge of the results. In addition, the experiment was run on one and 12 nodes. Table 4 reveals that the mapping time was reduced from 478.38 s on one node to 37.36 s on 12 nodes, which in turn raised the cost of splitting the input and merging the results to become over 66% of the total execution time for the 12-node execution. This clearly indicates that these two phases must be improved if any significant speed-up is to be gained when the faster aligners are invoked from the framework.

This is indeed the purpose of the enhancements described in subsection 4.4. Concretely, the framework offers options to optimize the input file distribution as well as to merge the results while they are generated, mimicking the behaviour of MPI HPG Aligner RNA BWT.

The speed-ups obtained by the framework, with the two aforementioned options active, using STAR and the HPG Aligner variants, are given in Figure 8. The plot there shows

Table 4. Execution time (in seconds) for the first stage of the generic-aligner framework invoked to apply HPG Aligner RNA BWT to `R80M0.1`, using one and 12 nodes. The numbers inside parenthesis indicate the percentage of the total cost.

#Nodes	Split input	Index load	Mapping	Merge results	Total time
1	–	14.95 (3%)	478.38 (97%)	–	493.33
12	55.25 (37.4%)	12.67 (8.6%)	37.36 (25.3%)	42.41 (28.7%)	147.69

a significant increase of the speed-ups for the enhanced framework compared with the meager results in Figure 7 (disabled optimizations). Interestingly, the framework with these options activated for HPG Aligner RNA BWT and the MPI HPG Aligner RNA BWT itself now offer similar scalability patterns. This is a positive sign as it indicates that the framework delivers a performance that is close to that obtained with a strongly-coupled solution.

After these changes, we believe that the bottleneck was shifted to the index load phase, which we found comparable in cost to the processing time when HPG Aligner RNA SA and STAR are applied in the first stage of the framework, using 12 nodes, with the optimizations enabled.

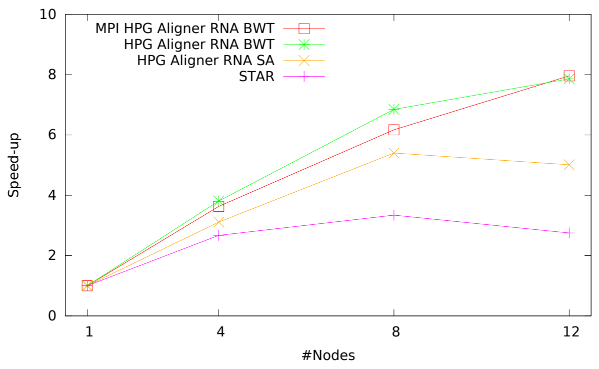

Figure 8. Speed-up for the first stage of the generic-aligner framework, with optimized input file distribution and overlapped merge, invoked to apply the RNA aligners to `R80M0.1`.

5.3 Sensitivity

We next discuss the accuracy of the alignment framework. A potential concern when performing the mapping on a cluster system is that, by distributing the work among several nodes/processes, a certain degree of sensitivity may be lost, because the nodes work independently of each other, with local information, during most of the time. The merge process at the end of the first stage aims to tackle this issue by compiling the local information learned from each stage and broadcasting that to all nodes. On the other hand, we can expect that the combination/invocation of two different aligners (or the same aligner with different options) from the framework may improve the sensitivity and/or reduce the cost of the process.

Table 5 shows the sensitivity and execution time of the generic-alignment framework, applied to a DNA-seq problem, on one and four nodes. (The sensitivity did not differ significantly for a larger number of nodes.) These results show that, for all the invoked DNA aligners, the percentage of mapped and correct reads is maintained when executed on four nodes, but the execution time is reduced. A second stage is pointless in this case, as virtually all the reads are reported as mapped by the first one.

Table 6 shows the sensitivity of the alignment framework, applied to an RNA-seq problem, on one and four nodes. The following four RNA aligners were invoked in the first stage: HPG Aligner RNA BWT, MapSplice 2, STAR, and TopHat 2; and HPG Aligner RNA BWT, MapSplice 2, and STAR were next applied to complete the process in the second stage. For this particular experiment, we employed the `R10M2.0` dataset. The reason to target this case was to “enforce” a lower number of successful mappings during the first stage, in order to highlight the effects of the refinement (second) stage.

Table 6a displays the accuracy rates obtained from a single-node execution with and without the second stage in place. The outcome from this experiment exposes the benefits of applying the refinement stage, showing gains, most of them important, in the percentage of correct reads for all the combinations. The slight decrease in the percentage of mapped reads when refining the initial results from STAR occurs because this particular aligner reports a number of reads as successfully mapped, but with a very low confidence. The framework re-visits (rescues) these low-confidence mapped reads, and decides whether they are actually mapped depending on the criteria defined by the aligner applied in the second stage. This particular case unveils that the criteria integrated in HPG Aligner RNA BWT and in MapSplice 2 to declare a successful mapping are more restrictive than that employed by STAR.

Table 6b offers the outcome from the analogous experiment using four nodes, showing similar result to those already discussed in the single-node evaluation: when the refinement stage is applied, the rates of correct reads are increased in all cases. The decrease in the percentage of mapped reads when STAR is applied during the first stage is again due to the reevaluation by HPG Aligner RNA BWT and MapSplice 2 of those reads mapped with a very low confidence.

Table 5. Sensitivity and execution time (in seconds) of the generic-aligner framework invoked to apply the DNA aligners to D40M, with one and four nodes.

	One node			Four nodes		
Mapper	%Mapped	%Correct	Time	%Mapped	%Correct	Time
Bowtie 2	99.12	94.65	732	99.12	94.65	258
Cushaw2-GPU	99.94	90.09	926	99.94	90.09	293
Cushaw3	100.00	92.69	2146	100.00	92.69	647
HPG Aligner DNA SA	99.91	98.74	367	99.91	98.74	175

Table 6. Sensitivity of the generic-aligner framework invoked to apply the RNA aligners to R10M2.0, with and without a second stage via HPG Aligner RNA BWT, MapSplice 2, and STAR.

a) Single-node multi-thread

	Without second phase		With second phase					
			HPG Aligner		MapSplice 2		STAR	
Mapper	%Mapped	%Correct	%Mapped	%Correct	%Mapped	%Correct	%Mapped	%Correct
HPG Aligner	99.00	93.25	—	—	99.72	93.68	99.70	93.70
MapSplice 2	98.50	92.05	99.42	94.05	—	—	99.40	93.50
STAR	98.86	82.38	98.64	88.62	98.59	87.69	—	—
TopHat 2	63.86	63.26	99.07	95.35	98.86	91.53	99.01	89.18

b) 4 nodes multi-thread

	Without second phase		With second phase					
			HPG Aligner		MapSplice 2		STAR	
Mapper	%Mapped	%Correct	%Mapped	%Correct	%Mapped	%Correct	%Mapped	%Correct
HPG Aligner	98.97	93.12	—	—	99.71	93.56	99.70	93.61
MapSplice 2	98.45	90.74	99.38	93.37	—	—	99.35	92.70
STAR	98.86	82.38	98.64	88.68	98.56	87.08	—	—
TopHat 2	63.69	63.01	99.06	95.26	98.79	89.18	99.01	89.00

Comparing the results obtained with one and four nodes, in Table 6a and b respectively, the percentage of mapped/correct reads is almost the same for both configurations.

As for the increase of execution time due to the introduction of the second stage, Table 7 displays the execution time obtained from single-node and 4-node executions with and without the second stage via HPG Aligner RNA BWT. In this case, the increase of time due to the usage of HPG Aligner RNA BWT as a second stage is perfectly acceptable both in the single-node and in the 4-node scenarios. Furthermore, the speed-ups attained by the 4-node execution are significant for MapSplice 2 and TopHat 2, although much lower for STAR. The reason is that, for this particular experiment, the optimized file distribution and overlapped merge were not enabled, producing an increase in the execution time of the alignment framework that is especially visible for STAR.

6 Conclusions

We have presented a generic-aligner framework to accelerate DNA-seq and RNA-seq analysis on clusters. Our solution actually implements an execution skeleton which can accommodate a number of existing aligners in any of its two stages. In this design, the framework takes care of the multi-node aspects of the execution (partitioning and distributing work and data, synchronizing the processes running at each node, collecting the intermediate and final results, etc.). The selected aligners, on the other hand, potentially exploit the hardware resources internal to each node (e.g., multicore processors and/or GPUs), unaware of the collaborative work being performed. The framework has been successfully validated by running it with a number of single-node multi-threaded state-of-the-art aligners, which did not require any modification of their code.

The experiments with our alignment framework show that this approach can be efficiently applied to enhance

Table 7. Execution time (in seconds) of the generic-aligner framework invoked to apply the RNA aligners to `R10M2.0`, on a single-node multi-thread and on 4 nodes multi-thread, with and without a second stage via HPG Aligner RNA BWT.

	Single-node multi-thread		4 nodes multi-thread	
Mapper	One phase	Two phases	One phase	Two phases
MapSplice 2	1299	1343	436	451
STAR	79	165	69	87
TopHat 2	2163	2275	856	892

the sensitivity of the mapping process by accommodating a second stage that refines those reads which could not be aligned with high confidence by the aligner applied in the first stage. Furthermore, by activating optimization techniques similar to those available in our MPI HPG Aligner RNA BWT, we demonstrate that the framework can match the performance of a strongly-coupled parallel solution. Finally, the experimentation shows that the parallel operation of the framework on several nodes does not impair the sensitivity, and reduces the execution time.

Acknowledgements

Access to the Maverick system at Texas Advanced Computing Center (TACC) is gratefully acknowledged.

Funding

The researchers from the University Jaume I were supported by the MINECO/CICYT [grant numbers TIN2011-23283, TIN2014-53495-R] and FEDER.

References

Biesecker LG (2010) Exome sequencing makes medical genomics a reality. *Nature Genetics* 42(1): 13–14.

Burrows M and Wheeler D (1994) A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California.

Cock PJ, Fields CJ, Goto N, Heuer ML and Rice PM (2010) The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic acids research* 38(6): 1767–1771.

Dobin A, Davis CA, Schlesinger F, Drenkow J, Zaleski C, Jha S, Batut P, Chaisson M and Gingeras TR (2013) STAR: ultrafast universal RNA-seq aligner. *Bioinformatics* 29(1): 15–21.

Ferragina P and Manzini G (2000) Opportunistic data structures with applications. In: *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*. IEEE, pp. 390–398.

Garber M, Grabherr MG, Guttman M and Trapnell C (2011) Computational methods for transcriptome annotation and quantification using RNA-seq. *Nature methods* 8(6): 469–477.

Grant GR, Farkas MH, Pizarro AD, Lahens NF, Schug J, Brunk BP, Stoeckert CJ, Hogenesch JB and Pierce EA (2011) Comparative analysis of RNA-seq alignment algorithms and the RNA-seq unified mapper (RUM). *Bioinformatics* 27(18): 2518–2528.

Kim D, Pertea G, Trapnell C, Pimentel H, Kelley R and Salzberg SL (2013) TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome Biology* 14(4): R36.

Langmead B and Salzberg SL (2012) Fast gapped-read alignment with Bowtie 2. *Nature methods* 9(4): 357–359.

Langmead B, Trapnell C, Pop M and Salzberg SL (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology* 10(3): R25.

Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R and 1000 Genome Project Data Processing Subgroup (2009) The sequence alignment/map format and SAMtools. *Bioinformatics* 25(16): 2078–2079.

Li H and Homer N (2010) A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics* 11(5): 473–483.

Liu Y, Popp B and Schmidt B (2014) CUSHAW3: sensitive and accurate base-space and color-space short-read alignment with hybrid seeding. *PloS one* 9(1): e86869.

Liu Y and Schmidt B (2014) CUSHAW2-GPU: empowering faster gapped short-read alignment using GPU computing. *Design & Test, IEEE* 31(1): 31–39.

Manber U and Myers G (1993) Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing* 22(5): 935–948.

Martínez H, Barrachina S, Castillo M, Tárraga J, Medina I, Dopazo J and Quintana-Ortí ES (2015a) Scalable RNA sequencing on clusters of multicore processors. In: *Trustcom/BigDataSE/ISPA, 2015 IEEE*, volume 3. IEEE, pp. 190–195.

Martínez H, Tárraga J, Medina I, Barrachina S, Castillo M, Dopazo J and Quintana-Ortí ES (2013) A dynamic pipeline for RNA sequencing on multicore processors. In: *Proceedings of the 20th European MPI Users' Group Meeting*. ACM, pp. 235–240.

Martínez H, Tárraga J, Medina I, Barrachina S, Castillo M, Dopazo J and Quintana-Ortí ES (2015b) Concurrent and accurate short read mapping on multicore processors. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on* 12(5): 995–1007.

Smith TF and Waterman MS (1981) Identification of common molecular subsequences. *Journal of molecular biology* 147(1): 195–197.

Tárraga J, Arnau V, Martínez H, Moreno R, Cazorla D, Salavert-Torres J, Blanquer-Espert I, Dopazo J and Medina I (2014) Acceleration of short and long DNA read mapping without loss of accuracy using suffix array. *Bioinformatics* 30(23): 1–3.

Wang K, Singh D, Zeng Z, Coleman SJ, Huang Y, Savich GL, He X, Mieczkowski P, Grimm SA, Perou CM, MacLeod JN, Chiang DY, Prins JF and Liu J (2010) MapSplice: accurate mapping of RNA-seq reads for splice junction discovery. *Nucleic acids research* 38(18): e178–e178.

Author Biographies

Héctor Martínez received his BS in Computer Engineering from the University Jaume I, Castellón, Spain, in 2011. He is a Researcher at the High Performance Computing and Architectures group of the Department of Computer Science and Engineering at the University Jaume I, since October 2011. His research interests include energy consumption of clusters, cache performance, and parallel algorithms and applications on bioinformatics.

Sergio Barrachina received his BS in Telecommunications Engineering from the Polytechnic University of Valencia, Spain, in 1997. He obtained his PhD in Computer Science from the University Jaume I, Castellón, Spain, in 2003. He is an Associate Professor in the Department of Computer Science and Engineering at the University Jaume I, Castellón, Spain, since October 2006. His research interests include parallel algorithms for dense and sparse algebra and bioinformatics. He has authored 6 papers in international peer-reviewed journals, and 20 communications on relevant conferences.

Maribel Castillo received her BS from the Polytechnic University of Valencia, Spain, in 1991. She obtained her PhD in Computer Science in 2000 in the same University. Since July 2002 she is an Associate Professor in the Department of Computer Science and Engineering at the University Jaume I, Castellón, Spain. Her research interests include the optimization of numerical and bioinformatic algorithms for general processors as well as for specific hardware (GPUs), and their parallelization on both message-passing parallel systems (mainly clusters) and shared-memory multiprocessors (SMPs, CC-NUMA multiprocessors, and multicore processors).

Joaquín Tárraga received his BS in Science Computer from the Polytechnic University of Valencia, Spain, in 1995. He obtained his PhD in Computer Science from the École Polytechnique Féderal de Lausanne (EPFL), Switzerland. Since 2005, he is a programmer analyst in the Spanish Bioinformatics Institute (INB) at the Research Center Príncipe Felipe, Valencia, Spain. His research interests include high-performance computing, parallel algorithms and applications on bioinformatics. He has authored around 15 papers in international peer-reviewed journals, and participated on several relevant conferences.

Ignacio Medina received his BS in Biochemistry and Molecular Biology from the University of Valencia, Spain, in 2001, and the BS in Computer Science from the Polytechnic University of Valencia, Spain, in 2004. In 2006, he joined the Department of Bioinformatics and Genomics at the Research Center Príncipe Felipe, as a Bioinformatician and Researcher, and in 2008 became a Software and Systems Manager of the same department. Since 2010, he has been a Project Manager of several clinic and development projects. His current research interests include genomic-scale expression and variation data analysis, high performance software development, distributed computing, machine learning and pattern recognition. He has authored more than 30 papers in international peer-reviewed journals. Since 2014, he works as a Project Manager and Senior Software Architect at EBI Variation Archive team at EMBL-EBI in Cambridge.

Joaquín Dopazo received his BS in Chemistry from the University of Valencia, Spain, in 1985, and obtained his PhD in Biology from the same university in 1999. He is the Director of the Department of Computational Genomics at the Research Center Príncipe Felipe, Valencia, Spain, since 2005. His research include functional genomics, bioinformatics and systems biology. He has authored more than 200 papers in international peer-reviewed journals. He serves in the editorial board of several bioinformatics journals. He also serves as a member of the FGED Advisory board and in the "Infrastructure for Tools Integration" committee of the ELIXIR.

Enrique S. Quintana-Ortí received his BS in Computer Science from the Polytechnic University of Valencia (Spain), in 1992. He also obtained his PhD from the same university in 1996. He is currently Professor of Computer Architecture at the University Jaume I, Castellón (Spain). During these years, he has authored 200+ papers in journals and international conferences in the area of parallel scientific computing. His current research interests target the development of high performance computational methods for modern architectures, including graphics processors, multicore architectures, digital signal processors, and clusters. He is currently Subject Area Editor for the journal "Parallel Computing".