Concurrent and Accurate Short Read Mapping on Multicore Processors

Héctor Martínez, Joaquín Tárraga, Ignacio Medina, Sergio Barrachina, Maribel Castillo, Joaquín Dopazo, Enrique S. Quintana-Ortí

Abstract—We introduce a parallel aligner with a work-flow organization for fast and accurate mapping of RNA sequences on servers equipped with multicore processors. Our software, HPG Aligner SA[1], exploits a suffix array to rapidly map a large fraction of the RNA fragments (reads), as well as leverages the accuracy of the Smith-Waterman algorithm to deal with conflictive reads. The aligner is enhanced with a careful strategy to detect splice junctions based on an adaptive division of RNA reads into small segments (or seeds), which are then mapped onto a number of candidate alignment locations, providing crucial information for the successful alignment of the complete reads.
The experimental results on a platform with Intel multicore technology report the parallel performance of HPG Aligner SA, on RNA reads of 100–400 nucleotides, which excels in execution time/sensitivity to state-of-the-art aligners such as TopHat 2+Bowtie 2, MapSplice, and STAR.

Index Terms—RNA, Short-read alignment, suffix array search, Smith-Waterman's algorithm, high performance computing, multicore processors.

1 Introduction

Over the last few years, biology has experienced a revolution as a result of the introduction of new DNA sequencing technology, known as Next-Generation Sequencing (NGS), that nowadays makes it possible to sequence the genomic DNA or RNA transcripts, or transcriptome, in a matter of days instead of years, at a very low cost. These recent high-throughput sequencers produce data at unprecedented rates and scale, with associated sequencing costs in continuous decrease. In particular, RNA sequencing (RNA-seq) technology [1] has arisen as a crucial analysis tool for biological and clinical research, as it can help determine and quantify the expression of genes, the RNA transcripts, that are activated or repressed due to different diseases or phenotypes, therefore providing an unbiased profile of a transcriptome that helps understand the etiology of a disease. In consequence, RNA-seq is increasingly replacing conventional expression microarrays in most practical scenarios [1].

- *H. Martínez, S. Barrachina, M. Castillo, and E. S. Quintana-Ortí are with the Computer Science and Engineering Department, Jaume I University, 12006–Castellón, Spain. E-mails: {martineh,barrachi,castillo,quintana}@uji.es.*
- *J. Tárraga, and J. Dopazo are with the Computational Genomics Institute, Prince Felipe Research Center, 46012–Valencia, Spain. E-mails: {jtarraga,jdopazo}@cipf.es.*
- *I. Medina is with the European Bioinformatics Institute, CB10 1SD–Cambridge, United Kingdom. E-mail: imedina@ebi.ac.uk.*

1. HPG Aligner SA is an open-source application. The software is available at http://www.opencb.org/.

Current NGS technology can sequence short DNA or RNA fragments, usually of length between 50 and 400 nucleotides (nts), though new sequencers with longer fragment sizes are being developed. Primary data produced by NGS sequencers consists of hundreds of millions or even billions of short DNA or RNA fragments which are called reads. The first step in NGS data processing in many comparative genomic experiments, including RNA-seq or genome resequencing [2], involves mapping the reads onto a reference genome, in order to locate the genomic coordinates these fragments come from. This step constitutes a highly expensive process from the computational point of view.

Genes code the sequence of proteins with a four letters alphabet. However, in higher eukaryotes, this message is not coded in a unique continuous stretch, but is divided into small stretches (average length 140 nts) called exons, interrupted by longer (average length 3,400 nts) non-coding stretches called introns. (On average, there are 8.8 exons and 7.8 introns per gene.) The transcription machinery transcribes both stretches in a unique nascent pre-messenger RNA (pre-mRNA) transcript. This transcript undergoes a process, known as splicing, in which introns are removed and exons are joined. Splicing is needed for the eukaryotic messenger RNA (mRNA) before it can be used to produce a correct protein through translation. RNA-seq experiments aim to sequence the mRNA. These fragments correspond to already concatenated introns, which should be mapped over the complex intron/exon structure of the genes.

The mapping process is more challenging for RNA-seq than for DNA-seq due to a variety of aspects.

First, as stated above, the genes in eukariotes are split into exons, which can be separated by intron zones comprising up to thousands of nucleotides. In consequence, when mapping the reads from RNA transcripts onto a reference genome, one faces the difficulty that these reads may contain a splice junction and, therefore, involve different exons, which in practice may lie thousands of nucleotides apart. This situation is referred to as a spliced alignment. This is even more complex for longer reads, because they have a higher probability of spanning two or more exons, complicating the alignment process. In addition, the existence of pseudogenes may cause wrong alignments of those exon-spanning reads, especially for the human genome. Sensitivity is also a serious concern at this point [3], given that natural variations or sequencing error may occur, yielding frequent mismatches between the reads and the reference genome, which increase the computational complexity of the procedure.

The Burrows-Wheeler transform (BWT) [4] is a popular technique that has been successfully applied to speed up ungapped alignment in genome index-based searches. A strategy that combines index-based read mapping with splice junction detection is implemented in TopHat [5], a software for the analysis of RNA-seq experiments. TopHat internally leverages Bowtie [2], a program for read mapping but with no support for gapped alignment. To tackle this, TopHat combines read mappings that lie in close genomic locations to reconstruct putative exon junctions, where unmapped reads are tried again. In the last years, a number of alternative aligners have been introduced that outperform TopHat+Bowtie in many scenarios; e.g., TopHat 2+Bowtie 2 [6], [7], GSNAP [8], MapSplice [9], SpliceMap [10], RUM [11], ContextMap [12], CUSHAW2 [13], and STAR [14].

In [15] we presented a concurrent algorithm for mapping short RNA sequences on multicore processors. This prototype mapper processed the data, initially stored on disk, in batches of reads which were passed between the consecutive stages of a pipeline. BWT was used initially to identify all the reads that could be mapped either correctly or with one error. Then, BWT was re-applied to map small fragments (referred to as seeds) of unaligned reads in order to locate their possible mappings. The experimental results showed high sensitivity (on average, around 97% for our mapper vs 61.1% in the best case for TopHat 2) and performance (with our mapper being more than one order of magnitude faster than TopHat 2). We believe that the poor sensitivity reported for TopHat 2 was partially due to the use of datasets automatically generated with the `dwgsim` application from the SAM tools [16].

In [17] we introduced an improved version of the BWT-based aligner. Compared with the original algorithm, the new aligner presented two key differences: *i)* the use of a "meta-exon" data structure to record information about successful mappings; and *ii)* a re-design of the mapping procedure, including an organization of the process into three work-flows in order to exploit the meta-exon information. These factors rendered increases in speed and sensitivity with respect to the aligner in [15] that were close to $2\times$ and 2%, respectively. Indeed, the sensitivity improvement was higher, as the new sensitivity tests also considered whether the reads correctly expanded over all the covered splice junctions.

The results in [17] showed that our BWT-based aligner was superior to a variety of mappers either in sensitivity, speed or both. However, we also noticed in our experiments that, for large datasets, STAR attained higher computational performance and parallel scalability than the rest of the mappers, including ours.

In this paper we present a refurnished work-flow-based aligner, which combines an efficient strategy for junction detection and a mapping technique with high sensitivity that correctly aligns reads with a high rate of mismatches (errors), insertions or deletions (indels). In particular, the new aligner features the following major changes with respect to [17]:

- The integration of a Suffix Array (SA) [18] search, instead of BWT, which significantly improves the speed of the initial stages of the aligner.
- A complete re-organization of the work-flow structure, including a new approach to control inter-stage interactions with reduced synchronization overhead and improved data locality.
- A re-design of the parallelization scheme that renders very high scalability.
- A new adaptive (dynamic) "seeding" strategy to process conflictive reads.

As a result of these enhancements, the new mapper, HPG Aligner SA, shows high sensitivity and great parallel performance for RNA-seq reads of length 100, 150, 250 and 400 nts even with large datasets (e.g., 80 million reads), outperforming state-of-the-art alternative aligners like TopHat 2+Bowtie 2, MapSplice and STAR, both in sensitivity and speed.

The rest of the paper is structured as follows. In section 2 we describe the mapping algorithm underlying HPG Aligner SA and describe its pipeline organization into two work-flows. In section 3 we review the parallelization of the pipeline using PThreads, a standard for shared-memory parallel programming. A detailed experimentation is reported next, in section 4, where we explore the computational performance (speed) and sensitivity of the full work-flow sequence on a server equipped with multicore technology from Intel; moreover, in that section we include a comparison of HPG Aligner SA against TopHat 2+Bowtie 2, MapSplice, and STAR on that platform. We finally close the paper with a discussion of conclusions and future work in section 5.

2 The Mapping Algorithm

In HPG Aligner SA, reads are aligned using a combination of mapping with SA [14] and local alignment with the Smith-Waterman algorithm (SWA) [19]. As SA is faster than SWA but does not allow indels, we employ the former in the early stages of the process, to obtain a rapid mapping of a large number of reads (those which contain no indels). On the other hand, SWA is applied in the final stages, to reliably map conflictive reads. Furthermore, in our approach splice junctions are detected by dividing unmapped reads into multiple seeds [6], [7], [9], of variable length [14], which are then mapped using SA at distances compatible with the length of an intron. The potential mapping regions detected using these seeds are next identified and brought together to perform the SWA alignment.

The new HPG Aligner SA mapping algorithm is organized as two consecutive work-flows (see Figure 1) and relies on a meta-exon structure that is updated with information about those reads that are mapped with high confidence during the process. Inside the second work-flow, this structure is used to speed up the mapping of unprocessed reads as well as to correctly align those reads that could not be mapped by the first work-flow.

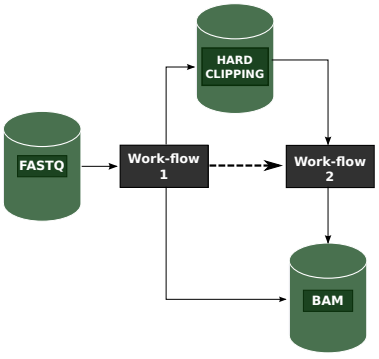


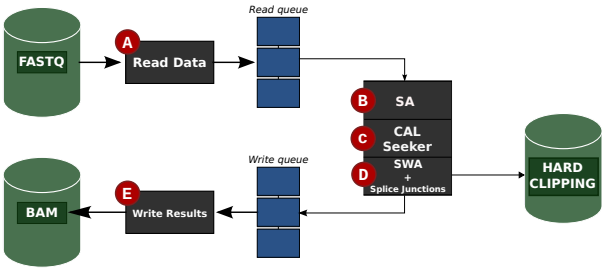
Fig. 1: Work-flow organization of HPG Aligner SA.

The detailed pipelines of the two work-flows are shown in Figure 2. The first work-flow maps the reads to the reference genome, and sets apart those reads for which a reliable mapping could not be found. These unmapped cases are marked as "hard clipping reads", and stored into a file that will be later processed by Work-flow 2; see Figure 2a.

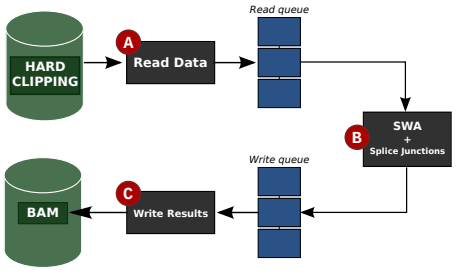
The second work-flow processes the hard clipping reads using the meta-exon structure as a back-up; see Figure 2b. Specifically, the information on those parts of a read that were mapped to positions present in the meta-exon structure is leveraged to test the mappings.

2.1 Work-flow 1

The mapping process of this work-flow is divided into five stages, A–E, as illustrated in Figure 2a. The interactions between the read stage (A) and stages B–D,



(a) Work-flow 1



(b) Work-flow 2

Fig. 2: Detail of the pipelines corresponding to the two work-flows embedded in HPG Aligner SA.

and between these stages (B–D) and the write stage (E) follow a producer-consumer relationship, and is synchronized through two shared data structures (the Read and Write queues), where the producer inserts work for the consumer to process. This organization allows the input/output to be overlapped with the computational stages. Our previous versions of HPG Aligner [15], [17] also regulated the interactions between stages B, C and D following this producer-consumer strategy, in an attempt to improve concurrency. However, we have found that, to improve data locality and reduce synchronization overheads, it is more efficient that these three stages are logically "amalgamated" from the point of view of their processing.

In the following paragraphs we discuss the tasks performed at each one of the stages together with a few relevant algorithmic details.

Stage A (Read Data). The data corresponding to the RNA reads are initially stored in a disk file following the standard FASTQ format [20]. Due to the large number of reads per experiment (typically, dozens of millions), this file is quite big, in general exceeding the capacity of the main memory. (Although the file sizes can vary a lot depending on the case study, in our experiments we often had to deal with files of 20 to 60 GBytes, which obviously do not fit in the memory of most of todays' desktop servers.)

Therefore, the principal task of this stage consists in retrieving data from the disk in blocks, hereafter referred to as *batches of reads*, of about 20 KBytes per batch, which e.g. leads to approximately 60 reads

if the reads contain 100 nts. Each batch of reads is stored in the *Read queue* for latter consumption in the subsequent stage, and it is placed in an array list in main memory, which expedites fast serial and indexed accesses. Among other information, each array records the header, sequence, size, and quality of each read.

Stage B (SA). This stage performs a fast mapping of reads to the genome, using our own implementation of the SA search.

The procedure extracts a batch from the read queue, and then applies an SA-based algorithm, allowing no indel per read.

If a read is successfully mapped, this stage creates an *alignment record* for each mapping that identifies the chromosome, the initial and final positions of the read within the chromosome, and the strand. Otherwise, it records those parts of the read that were found to map into the reference genome, if any.

Whenever a complete match is found for a given read, this stage updates the meta-exon structure with this information.

Stage C (Region and CAL Seeker). This stage processes those reads that were not mapped by the previous stage. It divides each unmapped read into several small fragments, or seeds, and searches for candidate alignment regions (CALs) for these reads.

The previous versions of HPG Aligner [15], [17] split each unmapped read into *seeds* of fixed length, of 18 nts each. For example, 9 seeds were obtained from a read comprising 100 nts: 6 starting from the first nucleotide of the read, an additional seed to include the nucleotides remaining at the end of the read, and 2 more seeds to improve the coverage at the beginning and the end of the read.

The new HPG Aligner SA follows an adaptive seeding approach. The mapper first splits an 18-nt seed from the beginning of the read, and this seed is mapped using SA. If successful, the seed is extended one nucleotide at a time for as long as there still exists a match. The next 18–nt seed will thus begin right after the nucleotide next to the last one for which a map was found. If a seed cannot be mapped, the next 18–nt seed is split from the 9^{th} nucleotide from the beginning of the unmapped seed. This procedure is repeated until the whole read is processed (see Figure 3). The choice of 18 as a splitting basis for the seeds is due to this value matching the length for the SA index that we are employing.

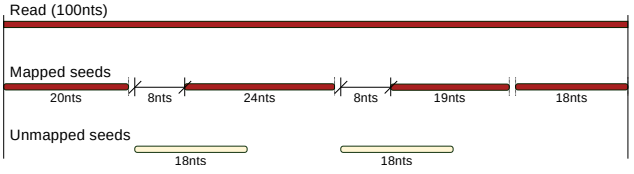


Fig. 3: A read split into multiple seeds.

The rationale of seeding the unmapped reads is the following. The most likely reason a given read was not mapped in stage B is that it contained at least one indel. By dividing that read into shorter seeds, we can expect that all the indels of the read are concentrated in only a few places of the read. Therefore, a majority of the seeds will be successfully mapped onto the reference genome.

Note that the seeds are quite small, which leads to a new problem, since now it is very probable that a seed maps to a large number of locations in the reference genome. The result of mapping these seeds is therefore a large collection of *regions*, which identify all the places in the reference genome where these seeds were successfully mapped.

Nevertheless, we can expect than only those regions that are related to a correct mapping of the read lie close together. We consider these areas as potential mappings of some part of the read, and we mark them as *candidate alignment locations* (CALs).

Concretely, in order to obtain the CALs for a given read, this stage first merges the regions of one read that are less than a certain number of nucleotides apart; and then classifies each merged region as a CAL. Figure 4 shows two CALs, CAL_0 and CAL_1, that have been identified from the regions obtained by mapping the seeds of a given read. In this figure, the gaps in the regions represent areas where some of the seeds could not be properly matched to the reference genome.

At the end of this process, the CALs are passed to the next stage.

Stage D (SWA+Splice Junctions). For each unmapped read, this stage receives the CALs associated with it. Those CALs that could lead to a splice junction are grouped together as CAL groups. Then, for each CAL and group of CALs a simple score is obtained by counting how many nucleotides of the read they covered. Only the CALs or group of CALs with the best score are kept.

As the read could not be completely mapped in previous steps, parts of the read will not match the CAL or CAL group. We differentiate between inner gaps in a CAL and the ends of the read not covered by the CAL.

For each CAL or CAL group, the inner gaps of the read will be filled first and then a mapping decision will be made depending on whether there are ends not covered by the CAL or CAL group or we can confidently find a mapping for these ends.

The inner gaps are filled following the next procedure. If the inner gap length is comparable to the corresponding unmapped part of the read, the SWA is applied to obtain an alignment of the unmapped part with the reference genome that corresponds to that inner gap.

Otherwise, if the inner gap length is greater than the corresponding unmapped part, we try to detect

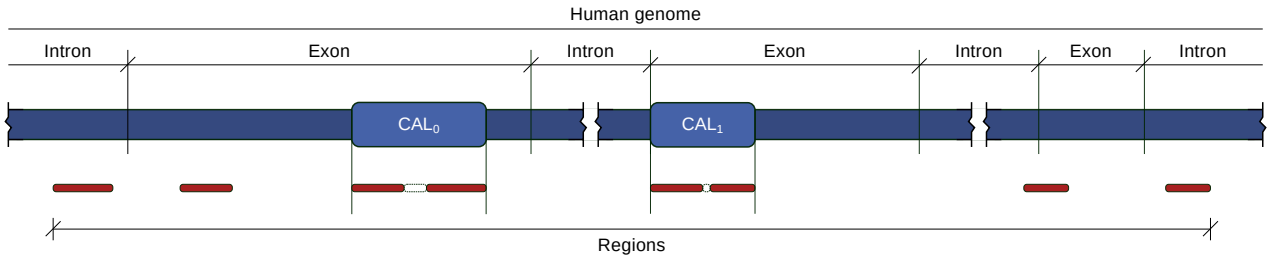


Fig. 4: Identification of CALs from regions.

a splice junction using a search which simply compares the unmapped nucleotides of the read with the reference genome until the intron marks (either the canonical ones, "GT-AG" or "CT-AC", or the semi-canonical ones, "AT-AC", "GT-AT", "GC–AG", or "CT-GC") are found (see Figure 5).

Once the inner gaps are mapped, if the read has no unmapped ends, it is reported as mapped.

Otherwise, if the read has unmapped ends and the CAL or CAL group cannot be found on the meta-exon structure, the read and the computed partial alignment are stored in the "Hard clipping" file, that will be later processed by the second work-flow.

Finally, if the read has unmapped ends, but the CAL or CAL group are found in the meta-exon structure, the information about known exons is used to try to map the unmapped ends with previous and next exons, respectively. If the ends can be successfully mapped, then the read is reported as mapped. Otherwise, the read and the computed partial alignment are stored in the hard-clipping file, to be later processed by the second work-flow.

All the splice junctions and the number of times each one has been detected are written to disk when all the batches are processed by the two work-flows. Therefore, this information must be maintained and updated during the whole process (i.e., not only for the current batch, but also for the current work-flow). To reduce the memory space necessary to store the information on detected splice junctions, as well as to rapidly find whether the last splice junction had already been detected, a self-balancing binary search tree data structure (an AVL tree) is used to keep this information.

Stage E: Write Results This stage completes the processing of a read batch, writing to disk whether the read was mapped or not using the SAM format [21]. The output file contains, among other information, the read id (the first component of the header), the sequence, its quality, the chromosome, and its initial position. The reported score for each mapping is computed following the Smith-Waterman score function:

$$score = (w_m \cdot matches - w_i \cdot mismatches - w_o \cdot indels_{open} - w_c \cdot indels_{close}) \cdot 100/(w_m \cdot read\_length),$$

where w_m, w_i, w_o and w_c are set to the default values used in the Smith-Waterman algorithm (5, 4, 10, and 0.5, respectively), though these can be adjusted by the user.

The above mapping process, performed by Work-flow 1, is summarized in Figure 6.

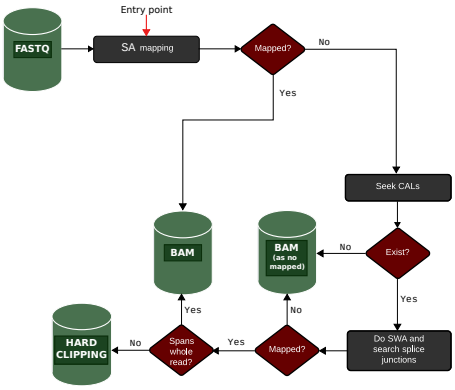


Fig. 6: Algorithmic description of the Work-flow 1 in HPG Aligner SA.

2.2 Work-flow 2

The second work-flow aligns the reads that could not be completely mapped by the first work-flow. It uses the meta-exon structure generated by the first work-flow, and consists of three stages that adhere to the producer-consumer paradigm; see Figure 2b.

Stage A (Read Data) of this work-flow simply reads the unmapped reads and their partial mapping from the "Hard clipping" file in blocks, and stores each batch of unmapped reads in the *Read queue* for later consumption by the subsequent stage.

Stage B (SWA and splice junction) proceeds as follows for each read in a batch. First, the current read is searched in the meta-exon structure. If the read is found and it is fully covered by the meta-exon structure, it is reported as mapped. Otherwise, if the read is found but any of its ends are not already covered by the meta-exon structure, the SWA is applied from that end to the part of the reference genome next to the mapped exon.

If the alignment score obtained by the SWA is higher than a given threshold, the read is reported

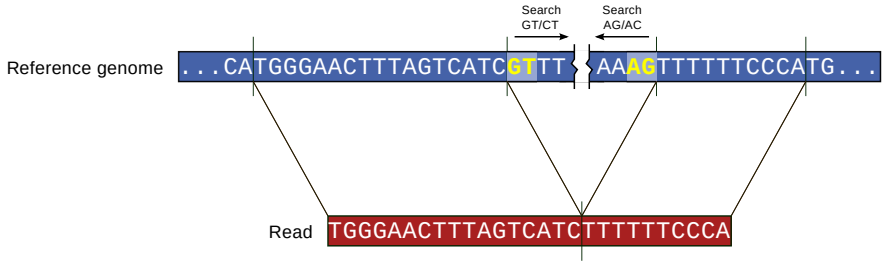


Fig. 5: Splice junction detection.

as mapped. Otherwise, the read is split into seeds, as described during the presentation of the first work-flow, and these seeds are grouped into CALs using a computationally less demanding procedure than that used for the first work-flow. Concretely, this new procedure commences by mapping only the first seed. The mappings thus obtained are searched in the meta-exon structure, which should be nearly complete at this stage. If an entry that matches the whole read is found in the meta-exon structure, we proceed to the next step; otherwise, the same procedure used in the first work-flow is performed. On the next step, a similar algorithm as that previously described is followed. The only difference lies in how the splice junctions are detected. Specifically, instead of using a simple search, the SWA is applied to align the unmapped parts of the read and the parts of the reference genome that extends the already mapped parts till any intron marks. If the alignment score obtained by the SWA is higher than the threshold for this stage, then the read is reported as mapped. Otherwise it is discarded.

Finally, when the last batch has been processed, all the splice-junctions detected in any of the two work-flows are written to an output file on disk using the BED format [22].

The process implemented by Work-flow 2 is summarized in Figure 7.

2.3 The meta-exon structure

The meta-exon data structure aims to reduce the soft-clipping and to improve the alignment of reads covering splice junctions under some difficult situations, such as when a read expands the junction in a very unbalanced manner (i.e., when one of the two ends of the read has a small part of less than 15–20 nts). For example, a read with a CIGAR [23] "92M627N8M" can be aligned in some RNA-seq aligners as "92M8S". The meta-exon stores the coordinates of the exons and splice junctions helping HPG Aligner SA to tackle these difficult alignments. In order to do so, the meta-exon is updated at run time as follows: i) When a read is completely mapped with high confidence to one exon (i.e., no splice site covered), a new meta-exon is created. As an exception, if the new meta-exon overlaps with one previously found, instead of

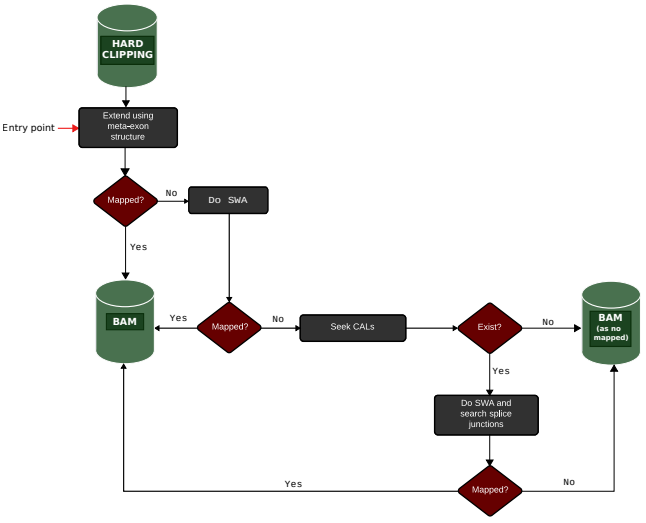


Fig. 7: Algorithmic description of the Work-flows 2 in HPG Aligner SA.

creating a new meta-exon, the old one is extended with the coordinates of the new one. ii) When a read is mapped with high confidence to a splice junction in a symmetric manner, two new meta-exons are created and the splice junction is stored in the meta-exon structure. As an exception again, in case there exist previously identified meta-exons which overlap with the new ones, they are simply extended and no new meta-exon is created. This procedure aims to confidently record any exons and splice junctions that can be obtained from the processed reads.

At execution time, HPG Aligner SA exploits high confidence mappings to learn where are the exons and splice sites. When a read covers an exon in an asymmetric manner, e.g. like in the earlier example "92M8S", HPG Aligner SA knows that there is a splice site around "92M" with one or more other possible exons. It then looks for the best match for those 8 nts, finding that there is one exon at 627 nt where the 8 nt can be aligned with high confidence (done via SWA). If a read with soft-clipping cannot be solved or improved because the meta-exon is not complete enough (as it is likely to happen with the first millions of reads), it is temporarily saved to be evaluated again after the whole file has been processed.

The meta-exon structure is implemented as a double linked list for each chromosome, where each element stores the start and end position of a discovered exon. To accelerate a direct access to some inner elements of the double linked list, it uses an additional array with shortcuts to the first and last exons for a given range of positions. This meta-exon implementation features a small memory footprint, is very fast to query, and has shown to significantly improve difficult alignments.

Although the meta-exon structure can be built from scratch, a structure assembled from known exons for the reference genome at hand can also be used to accelerate the whole process. Users can initialize the meta-exon structure by providing a transcriptome file with well-known exons from the command line Otherwise, this structure is initially empty and filled during runtime with the processed reads.

3 Leveraging Inter and Intra Stage Concurrency

The pipelined organization of the two work-flows in HPG Aligner SA naturally accommodates a task-parallel design that is well-suited for the hardware concurrency of current multicore processors. Concretely, both work-flows are divided into two types of stages, I/O and computational, with the interaction between them regulated via queues that act as data buffers, synchronizing the relative processing speeds (throughput) of the disk and the actual mapping procedure. Moreover, as the second work-flow operates with the meta-exon information produced by the initial work-flow, the parallel execution of HPG Aligner SA is further modulated by inserting a barrier which blocks the second work-flow till all work corresponding to the initial one has been completed.

The parallel aligner initially spawns a (user-defined) number of PThreads among which, at any given instant, only two can be performing I/O from/to disk. Upon creation, one of the threads immediately becomes an "I/O reader", and it starts to retrieve read batches (i.e., tasks) from the disk into the read queue. The remaining threads start polling this queue for tasks to process. When the number of batches on the read queue exceeds a certain threshold, the input from disk is momentarily detained, and the reader thread turns into an additional "worker" that processes read batches. When the number of pending tasks in the read queue falls below a threshold, the first idle worker becomes the I/O reader. Similarly, when a new chunk of data is introduced into the write queue, one of the threads momentarily becomes an "I/O writer", and copies that information to disk.

From the operational point of view, the mapper in [15] separated the computational stages using synchronization queues as well, in an attempt to balance the distribution of the workload at the granularity of "read batches". Threads inspected the synchronization queues between stages, e.g. in descending order of complexity and/or queue length, for new tasks. Upon encountering a non-empty queue, the thread extracted a task from there, executed the corresponding mapping operations, and inserted the result in the output queue. A mutual exclusion mechanism was in place to prevent race conditions when accessing the shared queues.

We have significantly re-modeled and simplified the operation of the pipeline in the new Work-flow 1, with the purpose of improving its throughput while maintaining the decoupling between the disk read rate and the computational throughput. Concretely, an idle thread now polls the read queue for work, dequeues a new task upon encountering it, and performs *all* processing corresponding to stages B, C, and D on the batch. We have experimentally found that, due to the large number of read batches (tasks) that are to be processed, this design does not reduce concurrency significantly. On the other hand, the reorganized operation mode improves data locality as well as reduces the synchronization overhead due to the use of shared, mutex-controlled data structures for inter-stage communication.

Figure 8 illustrates a fragment of the execution trace of Work-flow 1 of HPG Aligner SA on a 12-core Intel Xeon E5 platform. The trace illustrates that the pipelined execution of the I/O and computational stages of the mapper yields a perfect overlap, and how the role of (I/O) reader/writer is taken by different threads during the execution. The second work-flow presents a similar behavior.

Although the parallelization of the framework and stages using OpenMP (or PThreads) may seem straight-forward, we remark that this is only possible because of the careful organization of the complete HPG Aligner SA pipelines, and the design of clean procedural interfaces and specialized shared data structures.

We have also exploited a certain degree of intra-stage concurrency in our implementation of SWA. Our routine consists of two phases: i) The alignment score matrix is calculated; and ii) the optimal local alignment is reconstructed by going backwards from the element in the matrix with the maximum score. The are two manners of parallelizing the computation of the alignment score matrix: a) Intra-task [24], [25], by parallelizing within a single pair of sequences; and b) inter-task [26], [27], by processing multiple pairs of sequences in parallel. Our implementation exploits the first option using the Streaming SIMD Extensions (SSE) of Intel processors. Concretely, this approach processes up to four alignments (since a score is a 32-bit float value) within each single core by using 128-bit SSE registers.

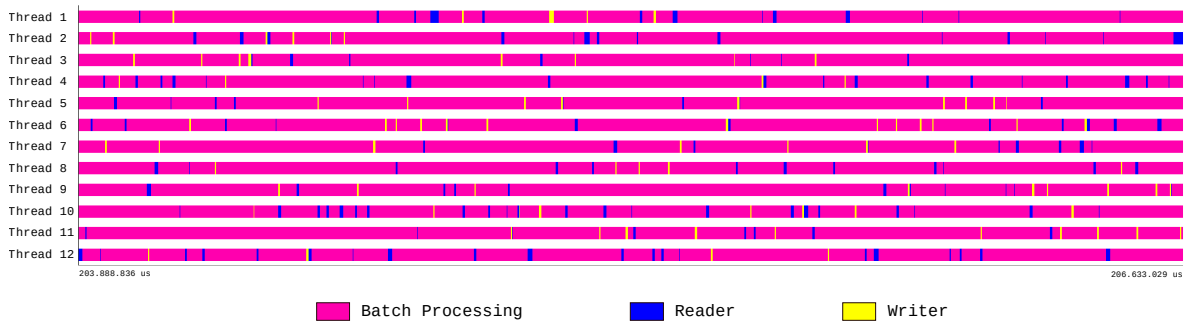


Fig. 8: Trace (fragment) illustrating the pipelined execution of Work-flow 1 with 12 threads/cores. Each horizontal bar shows the type of activity performed by one of the threads at a given time. The execution yields an interleaved execution with, e.g., I/O from/to disk proceeding in parallel with the batch processing. The trace shows there are at most one reader and one writer at any time.

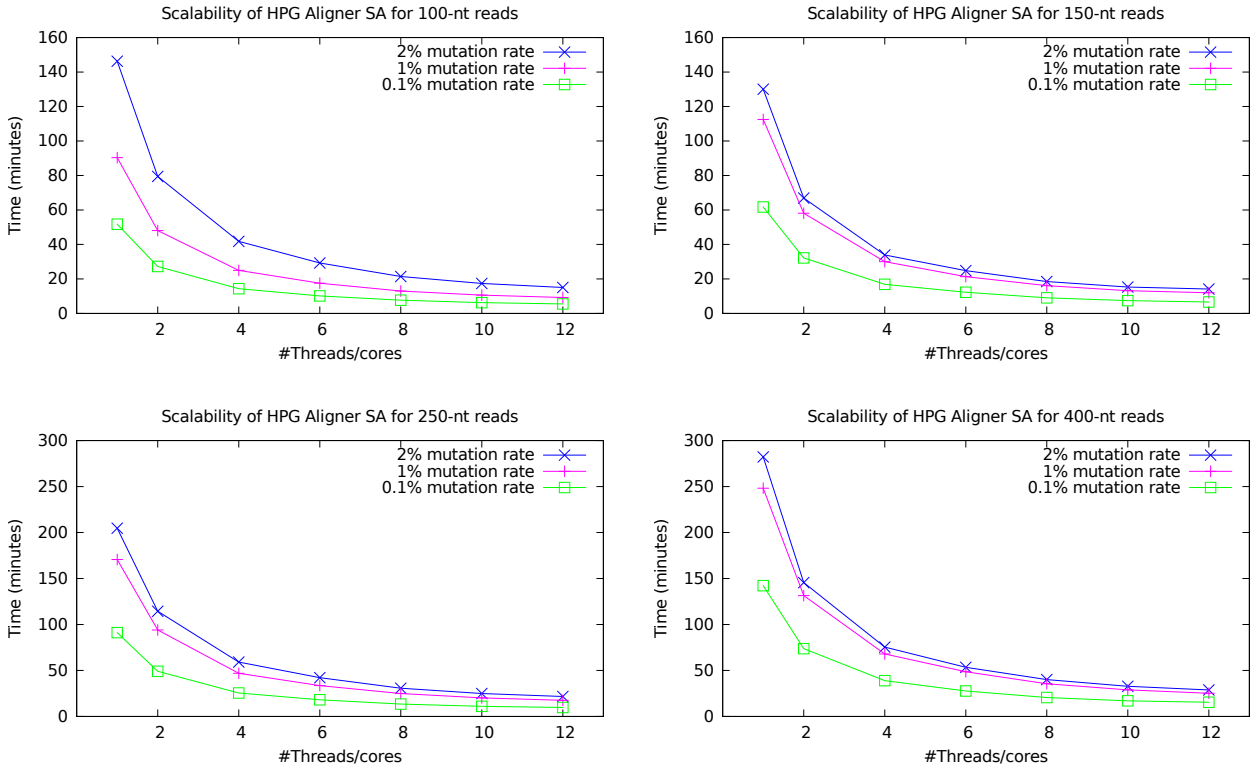


Fig. 9: Parallel performance (execution time in minutes) of HPG Aligner SA for the scenarios determined by the read length and the mutation rate.

4 Experimental Results

In this section we describe the experimental setup, we analyze the parallel performance of HPG Aligner SA, and we report a comparative study of the computational performance (speed) and sensitivity of the proposed algorithm against three state-of-the-art aligners.

4.1 Experimental setup

The experimental evaluation of the aligner was performed using simulated single-end datasets, obtained with the beers [28] simulation engine, for 80 million reads from the human genome and read lengths of 100, 150, 250, and 400 nts. For each one of these cases, three scenarios were designed, corresponding to mutation rates of 0.1%, 1%, and 2%. The indel frequency being the default (0.05%).

In the previous section we referred a number of parameters that affect the speed of HPG Aligner SA. For this particular setup, we set the minimum and maximum intron dimensions to 40 and 500,000 nts, respectively; the minimum CAL size was set to 20 nts; and the (EMBOSS) SWA configuration employed default values (match: 5, mismatch: -4, gap open: 10,

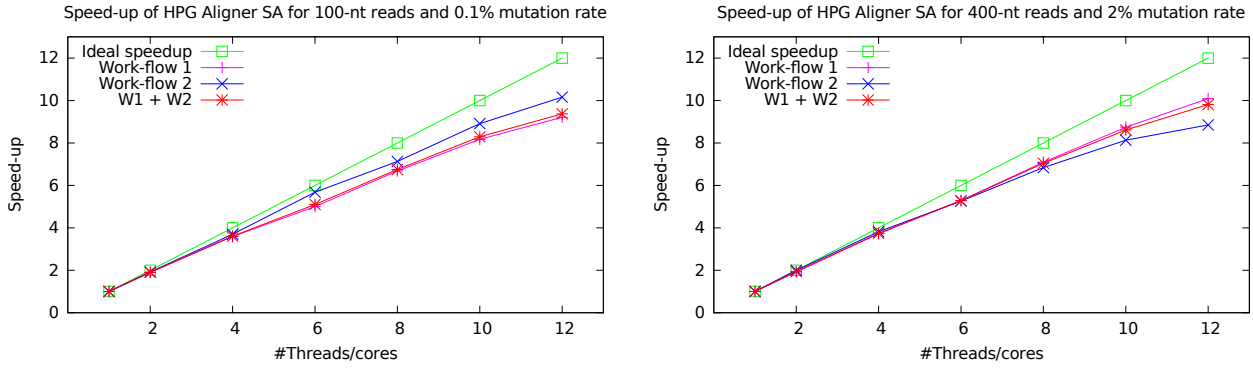


Fig. 10: Speed-ups of HPG Aligner SA for 100–nt reads, 0.1% mutation rate (left); and 400–nt reads, 2% mutation rate (right).

Read length (nts)	Mutation rate (%)	Number of threads/cores					
		2	4	6	8	10	12
100	0.1	1.90	3.61	5.10	6.75	8.29	9.37
	1	1.88	3.62	5.15	6.96	8.51	9.75
	2	1.84	3.50	4.99	6.83	8.42	9.71
150	0.1	1.92	3.66	5.02	6.81	8.28	9.32
	1	1.94	3.75	5.27	6.99	8.54	9.40
	2	1.94	3.84	5.24	7.03	8.48	9.19
250	0.1	1.86	3.57	5.04	6.77	8.26	9.35
	1	1.82	3.64	5.09	6.86	8.44	9.73
	2	1.79	3.46	4.85	6.66	8.22	9.45
400	0.1	1.93	3.66	5.14	6.93	8.37	9.23
	1	1.89	3.65	5.11	6.98	8.59	9.85
	2	1.94	3.74	5.28	7.04	8.61	9.81

TABLE 1: Speed-up of HPG Aligner SA for the scenarios determined by the read length and the mutation rate.

Read length (nts)	Mutation rate (%)	Work-flow 1 stages					Work-flow 2 stages		
		A	B	C	D	E	A	B	C
100	0.1	2.14	6.19	18.41	11.59	4.60	0.20	7.77	0.43
	1	1.99	5.25	38.62	20.40	3.56	0.57	15.76	1.39
	2	1.92	5.02	59.28	28.45	2.97	0.92	42.44	2.30

TABLE 2: Execution time (in minutes) of the stages of work-flows 1 and 2 for reads of 100 nt and different mutation rates.

and gap extend: 0.5).

All the experiments were performed on a platform equipped with two Intel Xeon E5645 processors (6 cores per processor) at 2.4 GHz, and 48 GBytes of RAM. In all cases, each thread is pinned to a different physical core.

4.2 Execution time and parallel performance

We first analyze the performance and parallel scalability of HPG Aligner SA. Figure 9 shows the execution time of the work-flow-based aligner for 1, 4, 6, 8 and 12 threads. These results expose the linear dependencies between the mutation rate/read length and the execution time of HPG Aligner SA. An aspect to note is the significant variation of the execution time when the mutation rate is increased from 1% to 2% for the 100–nt case (top-left plot), which contrasts with the behavior for the remaining three cases (150, 250 and 400–nt read length). For example, in the experiments with a single thread, we observe an increase of time by a factor of 1.62× when the mutation rate raises from 1 to 2% for the 100–nt case, but a considerably smaller one, e.g., around 1.15× for the 150–nt one. On the other hand, when the mutation rate increases from 0.1 to 1%, the behavior for all four read lengths is quite similar, with variation rates that range from 1.63× (400–nt reads, 12 threads) to 1.91× (250–nt, 2 threads). From the point of view of the read length, there is a close relation between the performance and the mutation rates 0.1% and 1% as both show increases of about 1.2×, 1.8× and 2.7× when the length varies from 100 to 150, 250 and 400 nts, respectively.

When the mutation rate is 2%, these magnitudes vary notably though, with factors around 0.9×, 1.4× and 1.9× when moving from 100 to 150, 250 and 400 nts, respectively.

Table 1 displays the speed-ups for HPG Aligner SA, defined as the ratio between the execution time obtained with 1 and t threads/cores. From the point of view of concurrency, these results reveal the notable parallel performance and scalability of HPG Aligner SA when up to 12 cores (the highest possible number in this platform) are involved, with values that are quite independent of the read length and mutation rate. Figure 10 illustrates the scalability of the complete aligner as well as the two embedded work-flows for two of the testbed cases (100–nt reads, 0.1% mutation rate; and 400–nt reads, 2% mutation rate). These plots show that the performance of the complete aligner (labeled there as W1+W2) closely follows that of Work-flow 1, while the throughput of Work-flow 2 can be higher or lower than that of the first work-flow, depending on the read length and mutation rate.

Table 2 shows the sequential computing time breakdown of the different stages of each work-flow for the 100–nt dataset with mutation rates of 0.1%, 1%, and 2%. (Only the time actually spent in each stage is recorded. The total sequential time is a bit greater than the sum of these times, due to the inter stage communications.) As shown, when the mutation rate increases, more work is diverted to stages C and D in work-flow 1 (more time is spent in these stages), and more reads will go through the work-flow 2 (stage E of work-flow 1, which writes reads already mapped, takes less time; but the work-flow 2 stages take longer). The more time consuming stage is stage C, region and CAL seeker, of the first work-flow.

4.3 Comparative study

We next compare both the speed and sensitivity of HPG Aligner SA against three recent aligners: TopHat 2+Bowtie 2 (v2.0.10+2.1.0), MapSplice (v2.1.5), and STAR (v2.3.0e). In order to obtain the following results, the RNA-seq mappers were executed with the parameters listed in Table 3. In all cases, we tested the optimal number of threads, but we only report the results corresponding to the best (i.e., shortest) execution time. In general, this corresponds to the use of t=12 threads. Given their long execution time, we only evaluated TopHat 2+Bowtie 2 (v2.0.10+2.1.0) and MapSplice for 100–nt reads.

The results in the top-left plot of Figure 11 and the columns labelled as "Time" in Table 4 show that HPG Aligner SA and STAR are consistently faster than MapSplice and TopHat 2+Bowtie 2. The remaining plots in Figure 11 and Table 5 extend these results for the two fastest aligners, HPG Aligner SA and STAR, and the remaining testbed cases. Focusing the analysis on the two fastest aligners, the time differences between them are always in favor of HPG Aligner SA, and grow with the mutation rate and, especially, the read length.

Tables 4 and 5 report the sensitivity of the mappers using two different metrics. RM corresponds to the percentage of Reads Mapped, and RCM to the percentage of Reads Correctly Mapped. The beers simulator provides, for each read, its position (chromosome, position and strand), and its CIGAR [23]. Therefore, we report a read as correctly mapped only if it has the same position and CIGAR as these stated by beers, which implies that it matches the correct intron-exon gene structure. Table 4 shows the results obtained by the four aligners with reads of 100 nts and the three mutation ratios. As revealed there, for mutation rates of 1% and 2%, the pair TopHat 2+Bowtie 2 offers mapping percentages much lower than those reported for HPG Aligner SA, STAR, and MapSplice. For this particular read length, the latter three aligners deliver very similar sensitivity results in the RM metric, while HPG Aligner SA and MapSplice clearly outperform STAR in terms of RCM. Comparing the best two mappers, MapSplice seems to be able to map a slightly higher amount of reads (RM), but HPG Aligner SA offers a slightly better RCM rate.

Table 5 examines the sensitivities achieved by the two fastest aligners, HPG Aligner SA and STAR, for reads of 150, 250, and 400 nts. These results show that HPG Aligner SA consistently offers higher RCM percentages than STAR. At this point we emphasize that, given that the two mappers attain similar rates of mapped reads (RM), the key is how many of these alignments are correct (RCM).

Tables 6 and 7 analyze the sensitivity of the mappers using the RCM metric only, with the simulated reads of each dataset classified into five sub-sets. The first four sub-sets correspond to simulated reads that span 1, 2, 3 or more exons. The last subset correspond to simulated reads that extend into one of the exons by 10 nts or less (labeled as " m.e."). These tables report that HPG Aligner SA attains higher sensitivity than STAR in nearly all the cases and outperforms both MapSplice and TopHat 2+Bowtie 2 in 8 out of the 15 cases of the comparison.

Finally, Tables 8 and 9 report the number of splice junctions present in each dataset together with the percentage of splice junctions that each aligner detects. These results show that HPG Aligner SA is able to detect more splice junctions than the rest of the aligners for all the datasets evaluated.

5 Conclusions

We have introduced a sequence of two pipelined work-flows for RNA sequencing, HPG Aligner SA, that exploits current multicore technology to efficiently perform short read mapping onto a reference genome. Our solution leverages a well-known principle, that of "*making the common case fast*", to apply a

HPG Aligner SA	`hpg-aligner rna -i ~/GENOMES/SA_INDEX/ -f simulate_dataset.fq \ --cpu-threads t -o outputHPG --fast-mode --read-batch-size 20000`
STAR	`STAR --genomeDir STAR_INDEX/ --readFilesIn simulate_dataset.fq --runThreadN t \ --outFileNamePrefix outputSTAR/`
MapSplice	`mapsplice.py -c MAP_SPLICE/ -x MAP_SPLICE/hs.73 \ -1 simulate_dataset.fq -p t -o outputMapSplice/`
TopHat 2+Bowtie 2	`tophat -p t --no-sort-bam --no-convert-bam -o outputTopHat2 \ Bowtie2_index/Homo_Sapiens_Bowtie simulate_dataset.fq`

TABLE 3: Command lines for the execution of the aligners included in the experimental comparison. In the experiments, t was replaced by the specific number of threads/cores.

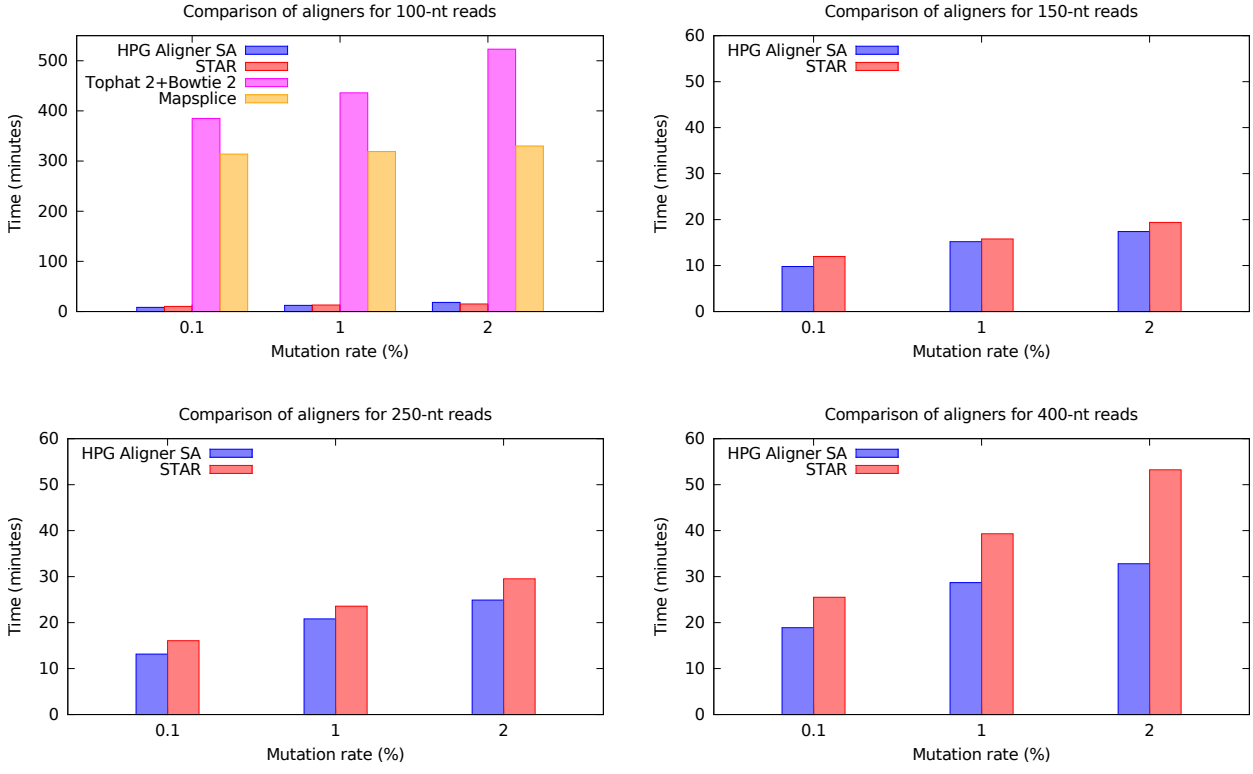


Fig. 11: Computational performance (execution time in minutes) of the four RNA-seq aligners for the scenarios determined by the read length and the mutation rate.

Read length (nts)	Mutation rate (%)	HPG Aligner SA RM	HPG Aligner SA RCM	HPG Aligner SA Time	STAR RM	STAR RCM	STAR Time	MapSplice RM	MapSplice RCM	MapSplice Time	TopHat 2+Bowtie 2 RM	TopHat 2+Bowtie 2 RCM	TopHat 2+Bowtie 2 Time
100	0.1	99.20	96.00	8.30	99.30	91.39	10.22	99.60	95.87	314.00	97.90	96.20	385.00
	1	98.80	94.80	12.30	99.20	86.70	13.07	99.30	94.90	319.00	87.98	86.95	436.00
	2	97.20	92.52	18.20	97.76	81.48	15.20	97.70	91.30	330.00	63.49	62.77	523.00

TABLE 4: Sensitivity (RM and RCM in %) and execution time (in minutes) of the four RNA-seq aligners for the scenarios determined by the read length and the mutation rate.

Read length (nts)	Mutation rate (%)	HPG Aligner SA			STAR		
		RM	RCM	Time	RM	RCM	Time
150	0.1	99.30	96.20	9.80	99.64	91.96	11.98
	1	98.86	94.57	15.20	99.50	86.95	15.78
	2	98.21	93.76	17.40	99.29	82.50	19.38
250	0.1	98.90	93.95	13.15	99.40	90.39	16.08
	1	98.30	92.40	20.80	98.90	85.65	23.57
	2	97.70	91.34	24.90	98.70	80.38	29.53
400	0.1	98.60	92.00	18.90	99.26	89.23	23.50
	1	97.70	89.20	28.70	98.70	84.16	39.33
	2	94.70	86.20	32.80	95.40	64.49	53.23

TABLE 5: Sensitivity (RM and RCM in %) and execution time (in minutes) of HPG Aligner SA and STAR for the scenarios determined by the read length and the mutation rate.

Read length (nts)	Mutation rate (%)	Exons	Reads (%)	HPG Aligner SA	STAR	MapSplice	TopHat 2 + Botwie 2
100	0.1	1	75.10	97.95	97.74	98.63	98.45
		2	22.62	92.19	76.04	88.66	90.14
		3	2.19	74.44	35.93	78.46	84.05
		>3	0.09	29.83	6.65	32.64	40.07
		m.e.	5.21	77.00	2.70	66.71	88.15
	1	1	74.49	96.42	94.13	97.91	89.61
		2	23.09	91.67	69.02	87.28	79.49
		3	2.35	76.00	28.67	75.92	76.86
		>3	0.07	52.03	12.85	61.84	65.15
		m.e.	5.18	79.37	1.97	65.74	78.88
	2	1	75.77	93.58	88.94	95.06	65.02
		2	22.09	90.91	61.80	81.43	56.06
		3	2.07	72.46	21.03	61.54	52.68
		>3	0.07	55.63	9.75	44.41	47.75
		m.e.	4.91	79.32	1.68	57.64	55.12

TABLE 6: Sensitivity (RCM in %) of the four RNA-seq aligners for the scenarios determined by the read length, the mutation rate, and the number of exons spanned by each read (1, 2, 3 or more). The label "m.e." denotes a case where the reads extend into one of the exons by 10 nts or less.

Mutation rate (%)	Exons	150 nts			250 nts			400 nts		
		Reads (%)	HPG Aligner SA	STAR	Reads (%)	HPG Aligner SA	STAR	Reads (%)	HPG Aligner SA	STAR
0.1	1	68.37	98.09	98.17	61.37	97.77	97.90	59.54	97.63	96.54
	2	23.99	94.92	85.74	15.82	93.35	89.02	8.63	91.89	87.76
	3	7.14	84.80	57.87	17.18	87.09	76.41	12.70	86.70	84.46
	>3	0.50	69.17	31.96	5.63	74.91	55.07	19.13	78.26	70.34
	m.e.	10.63	87.82	53.02	17.04	85.29	67.46	20.06	82.68	72.51
1	1	67.60	96.81	94.41	61.65	96.20	93.58	58.05	96.06	93.76
	2	24.42	92.53	79.16	15.53	91.46	84.24	8.86	89.16	82.17
	3	7.37	82.83	49.30	16.86	85.84	70.80	12.84	82.72	78.67
	>3	0.61	70.02	27.92	5.96	74.22	49.54	20.15	73.35	60.91
	m.e.	10.92	86.71	47.24	16.88	84.39	63.73	21.05	80.25	67.60
2	1	68.19	95.61	90.33	61.89	95.78	89.33	59.31	90.48	71.06
	2	24.26	92.71	73.80	15.51	88.99	77.70	8.90	85.85	63.49
	3	7.04	81.47	41.85	17.16	83.48	62.99	12.73	83.13	59.99
	>3	0.51	66.40	19.12	5.44	72.34	41.03	19.06	75.33	47.52
	m.e.	10.78	87.38	43.48	16.73	82.85	58.34	20.00	80.60	51.98

TABLE 7: Sensitivity (RCM in %) of HPG Aligner SA and STAR for the scenarios determined by the read length, the mutation rate, and the number of exons spanned by each read (1, 2, 3, or more). The label "m.e." denotes a case where the reads extend into one of the exons by 10 nts or less.

Read length (nts)	Mutation rate (%)	#Splice junctions	HPG Aligner SA	STAR	MapSplice	TopHat 2 + Bowtie 2
100	0.1	58,367	95.64	95.13	92.80	94.67
	1	56,527	96.69	95.90	93.87	93.96
	2	57,113	96.04	95.30	92.32	89.89

TABLE 8: Number of splice junctions present in each dataset and percentage of splice junctions detected by each aligner.

Read length (nts)	Mutation rate (%)	#Splice junctions	HPG Aligner SA	STAR
150	0.1	58,687	97.54	96.50
	1	59,040	97.65	96.43
	2	58,502	97.40	96.28
250	0.1	58,790	97.85	96.83
	1	59,776	98.00	96.77
	2	59,167	97.92	96.70
400	0.1	58,342	97.91	96.59
	1	59,305	98.32	97.30
	2	58,679	97.93	96.52

TABLE 9: Number of splice junctions present in each dataset and percentage of splice junctions detected by HPG Aligner SA and STAR.

variant of SA in order to rapidly map those reads with no indels, as the reliability of current NGS technology ensures that these cases constitute a large fraction of the total. After this initial stage, mapping failures are expected to be mostly due to reads with more than 1 indel or, alternatively, reads that span over two or more exons. To tackle both scenarios, we proceed by dividing the reads into a collection of short seeds (of variable length), which are next mapped yielding a collection of CALs in the reference genome. This information is finally passed to the accurate SWA that, under these conditions, turns most of the previous failures into successful mappings at a low cost.

The experiments on an Intel-based server with a large number of cores reveal the parallel efficiency of the HPG Aligner SA work-flow-based algorithm, which outperforms other state-of-the-art aligners from the points of view of both execution time and sensitivity.

ACKNOWLEDGMENTS

This work has been supported by the Bull-CIPF Chair for Computational Genomics.

The researchers from the Jaume I University were supported by project TIN2011-23283 and FEDER.

REFERENCES

[1] M. Garber, M. G. Grabherr, M. Guttman, and C. Trapnell, "Computational methods for transcriptome annotation and quantification using RNA-seq," *Nature methods*, vol. 8, pp. 469–477, 2011.

[2] B. Langmead, C. Trapnell, M. Pop, and S. L Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome biology*, vol. 10, no. 3, p. R25, 2009.

[3] H. Li and N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing," *Brief Bioinform*, vol. 11, pp. 473–483, 2010.

[4] D. Adjeroh, T. C. Bell, and A. Mukherjee, *The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern matching*. Springer, 2008.

[5] C. Trapnell, L. Pachter, and S. L. Salzberg, "TopHat: discovering splice junctions with RNA-seq," *Bioinformatics*, vol. 25, no. 9, pp. 1105–1111, 2009.

[6] D. Kim, G. Pertea, C. Trapnell, H. Pimentel, R. Kelley, and S. L. Salzberg, "TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions," *Genome Biology*, vol. 14, no. R36, 2013.

[7] L. Ben and S. Steven L., "Fast gapped-read alignment with Bowtie 2," *Nature methods*, vol. 9, no. 4, pp. 357–359, 2012.

[8] T. D. Wu and S. Nacu, "Fast and SNP-tolerant detection of complex variants and splicing in short reads," *Bioinformatics*, vol. 26, no. 7, pp. 873–881, 2010.

[9] K. Wang, D. Singh, Z. Zeng, S. J. Coleman, Y. Huang, G. L. Savich, X. He, P. Mieczkowski, S. A. Grimm, C. M. Perou, J. N. MacLeod, D. Y. Chiang, J. F. Prins, and J. Liu, "MapSplice: Accurate mapping of RNA-seq reads for splice junction discovery," *Nucleic Acids Research*, vol. 38, no. 18, p. e178, 2010.

[10] K. F. Au, H. Jiang, Y. Xing, and W. H. Wong, "Detection of splice junctions from paired-end RNA-seq by SpliceMap," *Nucleic Acids Research*, vol. 38, no. 4570–4578, 2010.

[11] G. R. Grant, M. H. Farkas, A. D. Pizarro, N. F. Lahens, J. Schug, B. P. Brunk, C. J. Stoeckert, J. B. Hogenesch, and E. A. Pierce, "Comparative analysis of RNA-seq alignment algorithms and the RNA-seq unifed mapper (rum)," *Bioinformatics*, vol. 27, no. 18, pp. 2518–2528, 2011.

[12] T. Bonfert, G. Csaba, R. Zimmer, and C. C. Friedel, "A context-based apporach to identify the most likely mapping for RNA-seq experiments," *Bioinformatics*, vol. 13, no. Suppl. 6, p. S9, 2012.

[13] Y. Liu and B. Schmidt, "Long read alignment based on maximal exact match seeds," *Bioinformatics*, vol. 28, no. 18, pp. i318–i324, 2012.

[14] A. Dobin, C. A. Davis, F. Schlesinger, J. Drenkow, C. Zaleski, S. Jha, P. Batut, M. Chaisson, and T. R. Gingeras, "STAR: ultrafast universal RNA-seq aligner," *Bioinformatics*, vol. 29, no. 1, pp. 15–21, 2013.

[15] H. Martínez, J. Tárraga, I. Medina, S. Barrachina, M. Castillo, J. Dopazo, and E. S. Quintana-Ortí, "A dynamic pipeline for RNA sequencing on multicore processors." in *EuroMPI*, 2013, pp. 235–240.

[16] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, and . G. P. D. Subgroup, "The sequence alignment/map format and SAM tools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.
[17] I. Medina, J. Tarraga, H. Martínez, S. Barrachina, M. Castillo, J. Paschall, J. Salavert-Torres, I. Blanquer, V. Hernández, E. Quintana-Ortí, and J. Dopazo, "Highly sensitive and ultrafast read mapping for RNA-seq analysis," *Bioinformatics*, under revision 2014.
[18] U. Manber and E. W. Myers, "Suffix arrays: A new method for on-line string searches," *SIAM J. Comput.*, vol. 22, no. 5, pp. 935–948, 1993.
[19] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J. Mol. Biol.*, vol. 147, pp. 195–197, 1981.
[20] P. J. A. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice, "The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants," *Nucleic Acids Research*, vol. 38, no. 6, pp. 1767–1771, 2010.
[21] SAMtools, "BAM/SAM API documentation," http://samtools.sourceforge.net/.
[22] UCSC Genome Bioinformatics, "BED format," http://genome.ucsc.edu/FAQ/FAQformat.html#format1.
[23] *The SAM Format Specification (v1.4-r985)*, September 2011.
[24] A. Wozniak, "Using video-oriented instructions to speed up sequence comparison," *Computer Applications Biosci*, vol. 13, p. 145:150, 1997.
[25] T. Rognes and E. Seeberg, "Six-fold speed-up of smith-waterman sequence database searches using parallel processing on common microprocessors," *Bioinformatics*, vol. 16, pp. 699–706, 2000.
[26] B. Alpern, L. Carter, and K. Gatlin, "Microparallelism and high performance protein matching," *Proceedings of the 1995 ACM/IEEE Supercomputing Conference*, 1995.
[27] T. Rognes, "Faster smith-waterman database searches with inter-sequence simd parallelisation," *BMC Bioinformatics*, vol. 12, p. 221, 2011.
[28] G. R. Grant, M. H. Farkas, A. D. Pizarro, N. F. Lahens, J. Schug, B. P. Brunk, C. J. S. Jr., J. B. Hogenesch, and E. A. Pierce, "Comparative analysis of rna-seq alignment algorithms and the rna-seq unified mapper (rum)." *Bioinformatics*, vol. 27, no. 18, pp. 2518–2528, 2011.

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

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 Prince Felipe Research Center, 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 Prince Felipe Research Center, 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.

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 Jaume I University, Castellón, Spain, in 2003. He is an Associate Professor in the Computer Science and Engineering Department at the Jaume I University, 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 Computer Science and Engineering Department at the Jaume I University, 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 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 Prince Felipe Research Center, 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 Jaume I University, 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".