2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing

Accelerating Bowtie2 with a lock-less concurrency approach and memory affinity

Claudia Misale
Computer Science Dept. – University of Torino, Italy
Email: misale@di.unito.it

***Abstract*—The implementation of DNA alignment tools for Bioinformatics lead to face different problems that dip into performances. A single alignment takes an amount of time that is not predictable and there are different factors that can affect performances, for instance the length of sequences can determine the computational grain of the task and mismatches or insertion/deletion (indels) increase time needed to complete an alignment. Moreover, an alignment is a strong memory-bound problem because of the irregular memory access patterns and limitations in memory-bandwidth. Over the years, many alignment tools were implemented. A concrete example is Bowtie2, one of the fastest (concurrent, Pthread-based) and state of the art not GPU-based alignment tool. Bowtie2 exploits concurrency by instantiating a pool of threads, which have access to a global input dataset, share the reference genome and have access to different objects for collecting alignment results. In this paper a modified implementation of Bowtie2 is presented, in which the concurrency structure has been changed. The proposed implementation exploits the task-farm skeleton pattern implemented as a Master-Worker. The Master-Worker pattern permits to delegate only to the Master thread dataset reading and to make private to each Worker data structures that are shared in the original version. Only the reference genome is left shared. As a further optimisation, the Master and each Worker were pinned on cores and the reference genome was allocated interleaved among memory nodes. The proposed implementation is able to gain up to 10 speedup points over the original implementation.**

I. Introduction

The diffusion of the Next Generation Sequencing (NGS) has increased the amount of data obtainable by genomic experiments. From a DNA sample, a NGS run is able to produce millions of short sequences, called *Reads*, which should be mapped onto a reference genome. Each Read is a sequence of nucleotides, which contains also the information about the *quality* of the sequencing process done to determine the reliability of the nucleotide called during sequencing. *Bowtie2*, is one of the fastest alignment tool [1] [2]. It is suffix-array and genome compression based and indexes the genome by using the *Burrows-Wheeler transform* (BWT) [3] and the *FM Index* [4]. The genome is loaded read-only in memory and it is shared among threads. In the multithreaded execution, a pool of threads is created. Those threads get *Reads* from a shared dataset, populate shared data structure with the alignment result and, finally, the main thread writes the alignment result into the output file. All accesses to shared data and objects are protected by mutexes (either Pthreads locks or spin-locks). The proposed implementation aims to:

- give access to the dataset only to the master thread;
- add to each Worker a private instance of all classes that are shared in the original version (both for the alignment algorithm and for results);
- avoid Workers and Master threads' migration by pinning them to cores;
- provide memory affinity for each thread's private data;
- allocate the genome with an interleaved policy among memory nodes.

The proposed design exploits the high-level *farm* pattern of FastFlow, a structured parallel programming framework targeting shared memory multi-core architectures [5], implemented on top of nonblocking multi-threading and lock-less (CAS-free) queues, and providing the programmer with high-level mechanism to tune task scheduling to achieve both load-balancing and memory affinity. With FastFlow it was possible to implement the Master-Worker pattern with few source code modifications, while threads pinning and genome interleaving were realised by using *libnuma*. This paper is organised as follows. Related works are discussed in Section II, Section III provides a brief introduction of the Bowtie2 alignment tool and Section IV presents the Stream Parallelism programming model and describes the FastFlow programming framework. In Section V the FastFlow porting of Bowtie2 is presented. Section VI reports performances evaluations and Section VII reports datasets analysis. Section VIII concludes the paper.

II. Related Works

Over years many algorithms for sequence alignment have been proposed and different tools were implemented, these already entire exploit multithreading. All these tools have different characteristics both in alignment algorithms and in the genome indexing.

A. Alignment Tools

The first step done before an alignment is to create and load the reference genome. The used techniques are hash tables and Burrows-Wheeler Transform [3]. The hash-based technique builds a hash table for subsequences of both genome and Reads. Keys are created by hashing subsequences and values are lists of positions in which subsequences can be found. The Burrows-Wheeler Transform (BWT) [3] is a string permutation algorithm used in data compression tools as bzip2. Ferragina and Manzini have enhanced it with the implementation of the FM-index [4] [6], an opportunistic data structure for text compression that permits fast substring queries. The name stands for Full-text index in Minute space. It is used to find rapidly the number of occurrences of a string pattern within the compressed text created, as well as the position in which the string is located. Hash-based tools, such as SOAP [7],

1066-6192/14 $31.00 © 2014 IEEE
DOI 10.1109/PDP.2014.50

SHRiMP [8] and mrFAST [9] are particularly suitable for short sequences alignment. SOAP is designed for efficient gapped and ungapped alignment of short Reads generated by Illumina-Solexa sequencing technology and it is able to execute single-end or pair-end alignment, small RNA discovery and mRNA tag sequence mapping. Shrimp and the last version Shrimp2 [10], are a short Reads mapper that support both colour-space and letter-space sequences. MrFAST maps short Reads emphasising the discovery of structural variation and segmental duplications. It is possible to map both single-end and paired-end Reads and can support up to 4+4 base-pair indels.

Burrows-Wheeler Transform (BWT) based tools, such as Bowtie [12] and Bowtie2 [1], BWA [13] and SOAP2 [14], use the FM index in order to create a suffix array on sequences compressed by the BWT algorithm. The combination of these two algorithms permits the creation of a compressed genome that can be fully loaded in memory. This technique has the limitation of a lower sensitivity in alignment with respect to hash-based indexing and of a reduction of maximum allowed mismatches (for instance, Bowtie2 allows only up to one mismatch), but has the advantage to make the alignment faster. Burrows-Wheeler Aligner (BWA) aligns relatively short Reads. It consists of three algorithms: BWA-backtrack, BWA-SW and BWA-MEM. The first is designed for Reads up to 100bp, while the other two for longer sequences. BWA-MEM is used typically for high quality Reads, having better performance than BWA-backtrack for 70-100bp Reads. SOAP2 is the improved version of SOAP, that both reduces computer memory usage and increases alignment speed. In this version, the genome indexing technique was changed from the hash-based one to the Burrows Wheeler Transformation compression algorithm. Bowtie2 is presented in Section III.

B. Tools Parallelisation and Optimisations

Alignment tools, generally, exploit parallelism via multi-threading. For instance, Bowtie, in both versions (Bowtie1 and Bowtie2) implements multithreading with Posix Threads, while the parallel version of BWA, pBWA [15], was developed by using the OpenMPI C library and can be executed exploiting both parallelism and multithreading. Shrimp and Shrimp2 alignment tools load the genome in memory in order to exploit multi-threading for mapping Reads in parallel. In [16] was presented distributed version of Shrimp, implemented upon the MapReduce programming model. Alignment tools that exploit GPUs were also presented. An example is SOAP3 [17], the GPU-based version of SOAP2. It can find all alignments with up to 3 mismatches and can be up to tens of times faster with respect to its previous version SOAP2. SOAP3 is at least 7.5 to 20 times faster than BWA and Bowtie, respectively. In addition, BarraCUDA [18] and CUSHAW [19] are short Reads alignment tools that exploit GPUs. Optimisations were proposed also for the Burrows-Wheeler transformation: this algorithm has been proposed with a novel locality-aware design for exact string matching in [20], in which this novel implementation can reduce LLC misses by 30%, TLB misses by 20% and resulting in up to 2.6- fold speedup with respect to the original BWA implementation.

III. The Bowtie2 Alignment Tool

Bowtie and Bowtie2 are alignment tool intensively used in biology, bioinformatics and medical research. While the first is specialised for short Reads (up to 25-50 nucleotides), Bowtie2 can align reads of very different length (no upper limit on read length) and support gapped alignment with gap penalties. The human genome indexing in Bowtie requires a small amount of memory to be fully loaded (about 2.3 GB), while other tools (both hash-table based and suffix-array based) asks for more memory: for instance, mrFAST (hash-table based) uses about 3.8 GB of memory and BWA (suffix-array based) requires about 3.2 GB of memory [22]. Different pipelines use Bowtie2 for bioinformatics, such as Myrna for a cloud-scale RNA-sequencing differential expression analysis [23], ChimeraScan for identifying chimeric transcriptions [24], commercial products such as Geospiza GeneSifter and CLC Genomics Server. Bowtie is also available as package in some Linux distribution, for example Debian and Fedora. This tool is used by researchers because it supports gapped, local, and paired-end alignment modes and, unlike different alignment tools (such as Shrimp), it is fast and can be used as the first step in pipelines for comparative genomics.

A. Bowtie2 Implementation

In shared memory multiprocessor architectures, Bowtie2 implements parallelism by using PThreads library. Bowtie2 can be executed specifying the desired number of threads which use synchronisation in fetching Reads, populating structures for alignment results, and performing global operations. All threads share the memory image of the index, so the memory footprint does not increase when multiple threads are used [12]. The alignment starts after the initialization of all the global structures, objects, variables and files related to the indexed genome. Referring to Figure 1, threads also share different data structures for collecting alignment result and other statistics (sh-mem output data structures). Each threads workflow is characterised by first setting up per-thread pointers to shared global data structures, creating private per-thread structures, and then cycling the following three steps:

1) Take a Read (or a pair of) from the input file (global to all threads);
2) Align the Read against the genome loaded into the index file (shared by threads);
3) Populate global structures related to the alignment output and general output statistics.

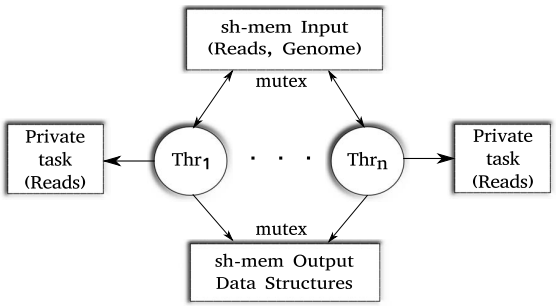

Fig. 1: Bowtie2 structure

All these steps are highly based on synchronisation because of the strong presence of shared data. In particular, mutexes were removed from

- Data structures and routines used during an alignment for saving Reads' information for the final result;
- Merging metrics objects and counters shared by multiple threads;
- Reporting a matrix of results into the output file;
- Tally memory allocation and releasing;
- Populating the object related to collecting the list of lines of output;
- Fetching Reads from file.

After collecting lines of output, the main thread flushes the alignment result to the (eventually) specified file and displays alignment statistics to the standard output.

IV. Stream Parallelism Programming Models

The stream parallelism programming model supports the parallel execution of a stream of tasks through a series of sequential or parallel stages. Each stage of a stream program (called kernel or filter) can be represented as a node of a graph of independent stages, in which nodes communicate over data channels. Each stage of such a graph works on one or more tasks coming from the input stream by applying some kind of computation and writing output tasks to the output stream. Parallelism is achieved by running each stage of the graph simultaneously on subsequent or independent data elements. Stream programs can be represented as a graph of concurrent activities and can be programmed using a low-level shared memory or message passing programming framework. In order to reduce programming effort and raising the level of abstraction, different parallel programming frameworks has been implemented. Within the skeletal approach [25] (or pattern-based parallel programming) can be captured most common parallel programming paradigms (such as MapReduce, ForAll, Divide & Conquer, etc.) and programmers can be provided with high-level constructs equipped with well-defined functional and extra-functional semantics [26]. The pipeline skeleton and the farm skeleton are two of the most used: with a pipeline, parallelism is achieved by running each stage simultaneously on a stream of data while the farm skeleton consists in running multiple independent stages in parallel, all working on different tasks of the input stream. The farm skeleton is typically described as a three-stage pipeline. The first stage consists of the Emitter, which dispatches stream items (or tasks) to a set of Workers(the second stage), that will apply transformations to stream items. The last stage consists of the Collector, which gather all results from Workers into a single stream. These logical stages are considered by a consolidated literature as the basic building blocks of stream programming [27]. The loop skeleton (also known as feedback) can generate cycles in a stream graph. This skeleton is used typically together with the farm skeleton to model recursive and Divide&Conquer computations.
The FastFlow implementation of the farm pattern is used to implement the proposed version of Bowtie2.

A. The FastFlow Library

FastFlow is a parallel programming environment targeting shared memory multi-core. It is implemented in C++ on top of Pthreads library and provides developers with task farms and pipelines parallel patterns [5]. FastFlow is based on four principles: layered design for supporting local optimizations and incremental design; efficiency in base mechanisms; stream parallelism support for implementing stream parallelism application, data parallel application or Divide&Conquer applications; skeleton/pattern based programming model [28]. As shown in Figure 2, it is designed as a stack of layers in order to abstract the level of parallelism, starting from cores level up to high level programming constructs. It supports parallel programming on cache-coherent shared memory multi-core and many-core architectures [29]. The core of FastFlow provides an efficient implementation of lock-free (CAS-free) Single-Producer-Single-Consumer (SPSC) FIFO queues. The next level extends SPSC queues to Multiple-Producer-Multiple-Consumer (MPMC) synchronizations implemented using only SPSC queues and arbiter threads, providing lock-free and wait-free data-flow graphs. Cache invalidations in core-to-core synchronizations are significantly reduced thanks to the lock-free implementations, which makes FastFlow to have higher speedup for fine-grained computations respect to other programming tools such as POSIX, Cilk, OpenMP, and Intel TBB [30]. On the top layer, programmers are provided with a set of patterns implemented as C++ templates: farm, farm-with-feedback (i.e. Divide&Conquer) and pipeline patterns, which can be arbitrarily combined. The FastFlow farm is used to implement the Master-Worker pattern in Bowtie2, which consists of an Emitter, a set of Workers and no Collector.

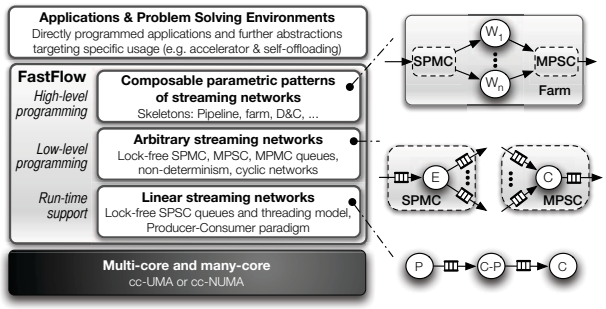

Fig. 2: FastFlow Layered Design

V. Bowtie2-FF Implementations

In the alternative version of Bowtie2 (*Bowtie2-FF*, from now on), the original code was changed in order to exhibit a high-level design using the FastFlow farm pattern. The farm pattern has specialised to behave as a Master-Worker, where reading data from input file and collecting alignment results are mapped onto the Master (Emitter E), whereas the alignment is mapped onto parallel Workers (W_i). As by-product, Bowtie2-FF exhibits no mutexes since data dependencies are managed by the FastFlow run time support (via Emitter thread) using a lock-less approach. Task balancing is automatically achieved thanks to FastFlow farm on-demand memory-affine scheduling

policy. Two variants of Bowtie2-FF are proposed. In the first, Emitter and Workers are pinned on cores and data structures used are allocated in the memory node in which each thread is pinned, thus providing locality for private data. The second variant provides threads pinning and genome allocation with an interleaved policy among memory nodes. Therefore, there are three variants of Bowtie2-FF:

1) Bowtie2-FF (Bt2FF): Master-Worker with workload dynamically partitioned among Workers by using an on-demand scheduling policy;
2) Bowtie2-FF with thread pinning (Bt2FF-pin): Master-Worker with threads pinning on cores and memory affinity for private data;
3) Bowtie2-FF with thread pinning and genome interleaving (Bt2FF-pin+int): Master-Worker with threads pinning on cores, memory affinity for private data and interleaved allocation policy among memory nodes for shared data (genome).

A. Master-Worker

In the original version of Bowtie2, each Worker thread takes needed data directly from the filesystem. For this reason, it is necessary to lock a counter variable that identifies the current Read and let each thread increment it avoiding data races and, consequently, it is necessary to avoid conflicts in accessing the input file. Furthermore, data structures related to Reads and alignment's results are allocated once at the beginning of the execution and the used memory is recycled at each iteration, thus avoiding new allocations. In the very first version of Bowtie2-FF, the Emitter was allocating each of the aforementioned structures every time a new Read was read from file and, at the end of each single alignment, each Worker destroyed these. This implementation was increasing the execution time because of the high number of allocation and deallocation. There would be the same effect by implementing the farm skeleton with both Emitter and Collector. In order to avoid changing the original allocation policy with memory recycling, each Worker was provided with a backward channel towards the Emitter, while Emitter and Collector were collapsed in the Emitter. In this way, data structures can be allocated once by the Emitter and recycled at each iteration. At the end of the computation, the Emitter itself will deallocate used memory, acting as the Collector. The implemented farm acts as follows: the first step consists of the instantiation of a finite number of tasks by the Emitter (`ff_task`) and of the initialisation of all task's members. Fields of `ff_task` involved in this first step are related to the current Read taken from the dataset (Read value, Read id). Once these are populated, the `ff_task` is sent to the first available Worker. Instead, if the current task was previously instantiated and has been just sent back to the Emitter by a Worker, statistics and results are merged, a new Read is assigned (if available) and it is sent back to the first available Worker.

B. Task Scheduling with FF–Queues

As shown in Figure 3, links between the Emitter (E) and each Worker (W_i) and vice-versa, are implemented using lock-free FIFO queues [31]. Backward connections ($W_i \rightarrow$E) accomplish to two main duties: 1) carry results from W_i to E, and 2) recycle exhaust tasks allocated memory in order to avoid any code change in the original one. Despite FastFlow supports both bound and unbound queues, all the queues are bounded to a fixed number of elements since the scheduling policy enforces the absence of deadlocks. Having fixed the size of queues, the Emitter is allowed to assign a finite number of tasks per Worker obtaining the following advantages:

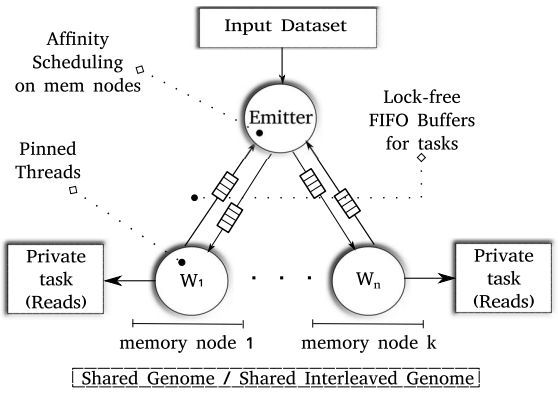

Fig. 3: Bowtie2-FF structure

- A Worker always has tasks ready in its queue; hence it never happens that a Worker is waiting for a task from the Emitter;
- Using bounded queues, the workload is dynamically partitioned among Workers in order to avoid the possibility of assigning long duration alignment tasks to few threads;
- The Emitter instantiates a finite number of tasks. Each task coming from Workers is recycled and populated with new data for a new alignment. The amount of memory used is bounded to *sizeof(task)*$\times$ *#Workers* $\times$ *QueueSize*;

That is, number of tasks created within an execution is not depending on datasets dimension or on alignments duration, it only depends on the number of Workers and on the size of queues connecting Workers and Emitter. Tasks number is not changing because it is fixed on *#Workers* $\times$ *QueueSize* elements. This way, memory allocated by the Emitter for an alignment job is equal to *sizeof(task)*$\times$ *#Workers* $\times$ *QueueSize*. Despite the number of tasks created is the same within each Bowtie2-FF version, the way the Emitter instantiates and schedules each task is different depending on the FastFlow version of Bowtie2. For each Worker, the Emitter in Bt2FF version follows these steps:

1) Creates a new task and initialises it with data needed for the alignment (i.e. Read value, Read id);
2) Task is pushed into the Worker's queue with a Round-Robin scheduling policy. Worker aligns the sequence against the genome and pushes the task with obtained results into the queue in its backward connection towards the Emitter;
3) The Emitter saves alignment results, initialises the received task with a new Read value and pushes it into the first free queue slot selected in a Round-Robin fashion among Workers;

It can be noticed that the proposed structure can be used also to realise a distributed version of Bowtie2-FF. The FastFlow library provides nodes with an extra communication channel which connects the edge-node of the graph with one or more edge nodes of other FastFlow application graphs running on the same or on a different host. A possible applicability for a distributed version could be executing multiple instances of Bowtie2-FF against one genome, thus aligning more datasets, or aligning one dataset against different genomes. In this case it would be necessary to add one additional thread for farms orchestration. By distributing workers on multiple hosts could only slow down the application because of high overhead due to communications between Emitter and Workers. Figure 4 shows a first comparison between Bowtie2 and the proposed Bt2FF. Besides locks removal, the execution time decreases of few seconds. To improve performances, implemented optimi-

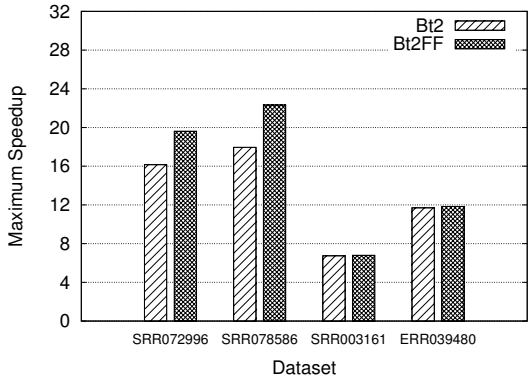

Fig. 4: First comparison between Bowtie2 and Bowtie2-FF

sations consist of: i) pinning threads to cores in order to avoid threads migration and mitigating the possibility cache misses due to accessing to different caches and ii) providing memory affinity for private data or accessed data. Details are presented in the next section.

C. Memory Access Optimisations

To ensure data locality, tasks are provided with three new information: 1) the id of the core in which destination Worker is running; 2) the id of the memory node that contains the core and 3) destination Worker, that is, the id of the Worker running on that core. For each Worker, now the Emitter behaves as follows:

1) It determines on which core the Worker is pinned and the relative memory node;
2) It creates a new task, which is initialised with the id of the destination Worker and the memory node id;
3) It allocates each task's private data needed for the alignment (i.e. Read value, Read id, data structure for results collection) and the task itself into the selected memory node;

This is valid for the allocation of new tasks. Considering that tasks are recycled in every iteration, the Emitter initialises received tasks with a new Read value and pushes them into the queue of the Worker identified by the relative field of the task. In this way, it is guaranteed for each task to be always sent back to the same Worker. For each Worker, allocating memory on the node in which they are executing is the way to get the best memory latency for their local data structures. Workers running on different nodes instead access the shared genome, which is allocated by the main thread on its current node, using the default allocation policy. In order to improve access to the genome (and avoiding its replication), it has been allocated with an interleaved policy by using the NUMA policy library, that is, allocating memory pages in a Round-Robin fashion into all nodes on the system. This way it is possible to avoid hot spot on the node in which genome is allocated. The interleaved allocation function is slower than the `malloc` allocation function, but it permits to have advantages in performances because of spreading the memory load across memory nodes.

VI. Performances Analysis

Because the default allocation policy is the one used in Bowtie2, it has been tested also using an interleaved allocation policy for the whole application, in order to compare the original application with test conditions as similar as possible. Interleaving is provided by the `numactl --interleave=all` command. Therefore, performances were measured on five versions of the software:

1) Bt2FF: the FastFlow version without threads pinning and without genome interleaving;
2) Bt2FF-pin: the FastFlow version with threads pinning and without genome interleaving;
3) Bt2FF-pin+int: the FastFlow version with both threads pinning and genome interleaving;
4) Bt2: the Bowtie2 original implementation;
5) Bt2-int: the Bowtie2 original implementation but with memory allocation policy interleaved among nodes.

All versions were tested on datasets with different characteristics. Two of them (ERR039480 and SRR003161) are referring to [1] and Unpublished dataset refers to a synthetic dataset generated in laboratory. Table I summarises datasets. Bowtie2-FF was tested on an Intel workstation with 4 eight-

TABLE I: Datasets

Dataset	Type	Read Length	# of Reads
SRR534301	Paired-End RNA-Seq	101	108749331
lane2_CTL_qseq	ChIP-Seq	36	53673423
SRR568427	ChIP-Seq	36	53594954
SRR502198	ChIP-Seq	36	25675656
SRR003161	Genomic Synthetic	47 – 4931	1376701
ERR039480	Genomic Synthetic	4 – 2716	36201289
SRR072996	ChIP-Seq	20	60673318
SRR078586	Genomic Hybrid Selection	8 – 68	3101013
SRR578211	Genomic	49	59654080
Unpublished	Genomic Synthetic	100	900006
SRR027963	Hi-C Paired-End	76	18145940
SRR576421	Genomic Dnase-Seq	50	212165270

core E7-4820 Nehalem (64 HyperThreads) @2.0GHz with 18MB L3 cache and 64 GBytes of main memory with Linux x86_64. Each processor uses HyperThreading with 2 contexts per core. Each Bowtie version has been executed with 1 up to 32 threads. Speedup has been calculated with respect to the sequential execution of Bowtie2. In Figure 5, it can

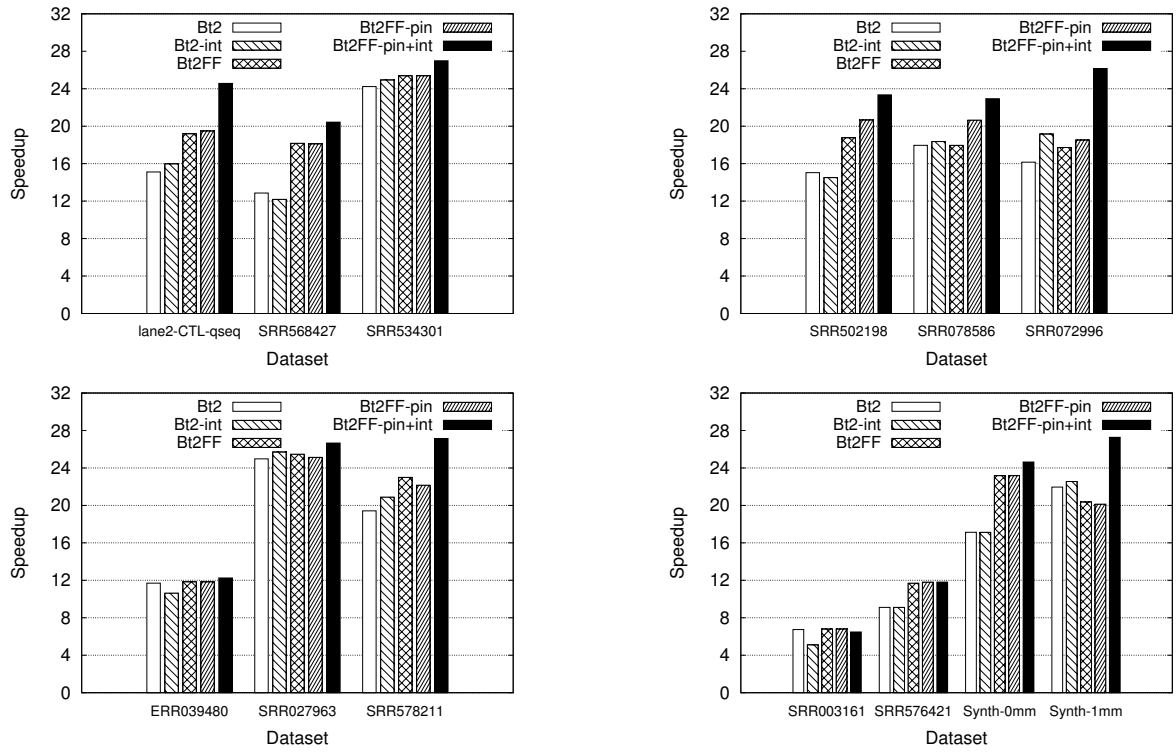

Fig. 5: Performances of all different versions of Bowtie2

be observed that the version with pinning and interleaving performs better in the most cases. Analysing performances about threads migration, number of used CPUs and executed operation on a test dataset with linux `perf` tool, it is possible to notice how Bowtie2-FF with threads pinning and genome interleaving (Bt2FF-pin+int) performs better than Bowtie2. In Table II were analysed performances with `perf stat -d` on dataset SRR578211, on a subset of 5 million of Reads of length 49. Perf stat was used to monitor the execution of the original Bowtie2 version with `interleave==all` policy (Bt2-int) and the FastFlow version with threads pinning, memory affinity and interleaving of shared data (Bt2FF-pin+int) running with 32 threads (31 Workers and 1 Emitter). By this estimation,

TABLE II: Analysis with Perf Stat

Metric	Bt2FF-pin+int	Bt2 interleaved
CPUs utilised	30.408	28.655
Context-switches	34816	199592
CPU-migrations	53	901
IPC	1.01	0.75
Stalled cycles per insn	0.58	0.93
Stalled-cycles-frontend	58.59%	69.67%
Stalled-cycles-backend	38.53%	53.19%
Branches-misses	5.08%	5.20%
L1-dcache-misses (of all L1-dcache hits)	4.07%	3.92%
LLC-load-misses (of all LL-cache hits)	41.62%	46.14%
Execution time (s)	35	55

we can notice that the FastFlow version can exploit all CPUs thanks to threads pinning, while Bowtie2, also executed with 31 Worker threads, uses approximately 28 CPUs. The resulting speedup with respect to the original Bowtie2 execution is 24.83 for the Bt2FF-pin+int version and 15.8 for the Bt2-int version. The number of threads migration by pinning Worker threads in Bowtie2-FF version is 17 times lesser than Bowtie2. The main thread is not pinned and then those migrations could be imputable to the main thread.

The front-end is a group of the pipeline stages responsible for fetch and decode phases, by which providing a stream of work to the back-end, which works on micro-operations. Intel Nehalem workstation can deliver up to 4 instruction/cycles because of its 4 decoders and a CPU cycle is stalled when the pipeline does not advance during it. Here we can notice that the FastFlow version, despite of few code modifications, is able to exploit the workstation better than the original version decreasing by about 10% front-end stalled cycles and by about 15% back-end stalled cycles. We can also notice that IPC in Bowtie2-FF is higher than in Bowtie2.

Analysing cache misses both in L1-dcache and in LL-cache (L3 in this workstation, shared among cores) by executing Bowtie2 on different datasets there are always different results. For instance, tests on a real world dataset and a synthetic one are reported in table III. Bowtie2-FF with threads pinning and genome interleaving performs better with the real dataset while, with the synthetic datasets, it has the same performances as Bowtie2. Although small differences in percentages and the brief memory analysis, cache-misses can be considered an important factor that affects performances.

TABLE III: Analysis with Perf Stat

Metric	Bt2-FF Pin+Int	Bt2 interleaved
	Real Dataset	
L1-dcache-misses (of all L1-dcache hits)	4.55%	4.59%
LLC-load-misses (of all LL-cache hits)	46.61%	52.99%
Execution time (s)	57.18	77.82
	Synthetic Dataset	
L1-dcache-misses	10.54% of all L1-dcache hits	12.27%
LLC-load-misses	36.94% of all LL-cache hits	36.45%
Execution time (s)	99.31	108.79

VII. Datasets Analysis

Because of the strong differences in performances among tests (consider for instance datasets SRR568427 and SRR576421 in Figure 5), the attention moved on the datasets analysis in order to understand if there is some particular feature that affects global performances. To analyse in deep all datasets, the *FastQC* tool was used. Despite the number of different statistics reported by FastQC (i.e. number of sequences, per-base/per-sequence GC content, overrepresented sequences, Kmer contents), Reads length appears to be a factor that can affect performances. In Figure 6 it is reported the comparison between maximum speedup values of Bt2FF-pin+int and the original implementation. Each point of the graph represents the obtained speedup with a dataset with Reads length defined on the x axis. Each point of the graph refers to, respectively, SRR072996, lane2_CTL_qseq, SRR578211, SRR078586, SRR027963 and SRR534301. Synthetic datasets were not considered (see table I), because are not representative of real world experiments, and dataset SRR576421. The last one has very low performances (maximum speedup of Bt2 is 9.10, maximum speedup of Bt2FF-pin+int is 11.81) and it was not represented because it can be considered a biased dataset. The protocol used to create this dataset (Dnase-Seq), creates a high number of repeated sub-sequences (Kmer) over Read length, which are localised after the 20th nucleotide. Reads created by this protocol are long 20: the other values should be trimmed because are nucleotides specifically recognized by the enzyme used in the experiment. It should be investigated if these residues are the reason of low performances. Datasets SRR502198 and SRR568427 were not reported because Reads length is 36 and they were overlapping the second dataset plotted in the graph (referred to lane2_CTL_qseq) which has Reads length 36, too. Maximum speedup obtained by Bowtie2 and Bt2FF-pin+int in these two datasets are, respectively, 15.03 and 23.34 for SRR502198 and 12.86 and 20.42 for SRR568427. It can be noticed that Bt2FF-pin+int can always gain speedup points with respect to Bowtie2. In particular, datasets with Reads of length less than 70 nucleotides make the FastFlow version of Bowtie2 to gain up to 10 points of speedup over the original version. On last two datasets, Bt2FF-pin+int is faster than the original version but gains at most about 4 speedup points. Even though this analysis can not be considered fully demonstrative of the correlation between Reads length and tool performances, it has been shown that on datasets with short/medium length, Bowtie2-FF with threads pinning and genome interleaving performs better than the original version of the tool. It is confirmed by the FastFlow library because it is designed to perform better with fine grain tasks. Datasets SRR003161 and ERR039480 are those with Reads with very different length (see table I) that are obtained, respectively, with Roche 454 GS FLX Titanium and Ion Torrent PGM technology, which generate long Reads (up to 1000bp the former, up to 200 the latter). Such a length is generally used in metagenomics analysis. A further experiment has been done using a genomic dataset obtained by a 454 GS FLX Titanium sequencing (85361 Reads, average 600bp). Speedup expectations has been confirmed: Bowtie2 and Bowtie2-FF with threads pinning and genome interleaving have same performances, reaching a maximum speedup of (respectively) 6.61 and 6.86 both executed with 12 workers. Alignment tests with Roche 454 real and synthetic datasets can be found in [32], where Bowtie2 has been compared to other tools, reporting a poor speedup as well (maximum speedup of 7). All these results confirm that, besides this alignment algorithm is able to align very long Reads, in terms of performances is not well suited for metagenomics.

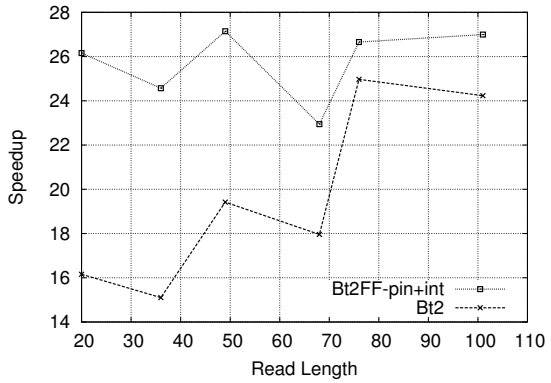

Fig. 6: Speedup trend with respect to Reads length.

VIII. Conclusion

In this paper, a modified version of the Bowtie2 alignment tool has been presented, in which the concurrency structure has been modified into a Master-Worker. The proposed implementation exploits the task-farm skeleton pattern implemented as a Master-Worker, which permits to delegate only to the Master thread dataset reading and to make private to each Worker data structures that are shared in the original version. Only the reference genome is left shared. As a further optimisation, the Master and each Worker has been pinned on cores in order to avoid threads migration and, consequently, threads were provided with memory affinity to get the best memory latency for their local data structures. The genome, accessed by Workers running on different nodes, has been allocated with an interleaved policy in order to avoid hot spot on the node in which genome is allocated. The modified version with both threads pinning and genome interleaving performs better than the original version, especially with datasets with short Reads, gaining up to 10 speedup points. This is possible thanks to the FastFlow library, designed to perform better with fine grain

tasks. Considering that many pipelines for comparative genomics use Bowtie2 as preliminary step, the proposed version can be used without changing or adding any input parameter. With some dataset, the proposed version performs the same as the original or few seconds better (ERR039480, SRR003161 and SRR576421). The first two of those datasets are synthetic and are composed by Reads of very different length (see Table I), while the last is a real world dataset with Reads of fixed size, but it is biased because of the high presence of kmers. It should be investigated, as future work, if datasets features are a reason of performances lowering. Another important factor that should be investigated is how workers access shared data and, in general, utilise tools for performance analysis in order to understand how memory is accessed by threads and trying to improve it.

ACKNOWLEDGMENT

This work has been supported by the European Union Framework 7 grant IST-2011-288570 ParaPhrase: Parallel Patterns for Adaptive Heterogeneous Multicore Systems, http://www.paraphrase-ict.eu.

REFERENCES

[1] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with Bowtie 2," *Nat Meth*, vol. 9, no. 4, pp. 357–359, apr 2012.

[2] Y. Chen, J. Hong, W. Cui, J. Zaneveld, W. Wang, R. Gibbs, Y. Xiao, and R. Chen, "Cgap-align: A high performance dna short read alignment tool," *PLoS ONE*, vol. 8, no. 4, 04 2013.

[3] M. Burrows, D. J. Wheeler, M. Burrows, and D. J. Wheeler, "A block-sorting lossless data compression algorithm," Tech. Rep., 1994.

[4] P. Ferragina and G. Manzini, "Opportunistic data structures with applications," in *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, ser. FOCS '00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 390–.

[5] M. Aldinucci, M. Danelutto, P. Kilpatrick, and M. Torquati, "Fastflow: high-level and efficient streaming on multi-core," in *Programming Multi-core and Many-core Computing Systems*, ser. Parallel and Distributed Computing, S. Pllana and F. Xhafa, Eds. Wiley, 2013, ch. 13.

[6] P. Ferragina and G. Manzini, "An experimental study of an opportunistic index," in *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA '01. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2001, pp. 269–278.

[7] R. Li, Y. Li, K. Kristiansen, and J. Wang, "Soap: short oligonucleotide alignment program," *Bioinformatics*, vol. 24, no. 5, pp. 713–714, 2008.

[8] S. M. Rumble, P. Lacroute, A. V. Dalca, M. Fiume, A. Sidow, and M. Brudno, "Shrimp: Accurate mapping of short color-space reads," *PLoS Comput Biol*, vol. 5, 05 2009.

[9] C. Alkan, J. M. Kidd, T. Marques-Bonet, G. Aksay, F. Antonacci, F. Hormozdiari, J. O. Kitzman, C. Baker, M. Malig, O. Mutlu, S. C. Sahinalp, R. A. Gibbs, and E. E. Eichler, "Personalized copy number and segmental duplication maps using next-generation sequencing." *Nature genetics*, vol. 41, no. 10, pp. 1061–1067, Oct. 2009.

[10] M. David, M. Dzamba, D. Lister, L. Ilie, and M. Brudno, "Shrimp2: Sensitive yet practical short read mapping," *Bioinformatics*, vol. 27, no. 7, pp. 1011–1012, Apr. 2011.

[11] C. Alkan, J. M. Kidd, T. Marques-Bonet, G. Aksay, F. Antonacci, F. Hormozdiari, J. O. Kitzman, C. Baker, M. Malig, O. Mutlu, S. C. Sahinalp, R. A. Gibbs, and E. E. Eichler, "Personalized copy number and segmental duplication maps using next-generation sequencing." *Nature genetics*, vol. 41, no. 10, pp. 1061–1067, Oct. 2009.

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

[13] H. Li and R. Durbin, "Fast and accurate short read alignment with burrowswheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

[14] R. Li, C. Yu, Y. Li, T.-W. Lam, S.-M. Yiu, K. Kristiansen, and J. Wang, "Soap2: an improved ultrafast tool for short read alignment," *Bioinformatics*, vol. 25, no. 15, pp. 1966–1967, 2009.

[15] D. Peters, K. Qiu, and P. Liang, "Faster short dna sequence alignment with parallel bwa," in *American Institute of Physics Conference Series*, ser. American Institute of Physics Conference Series, I. Kotsireas, R. Melnik, and B. West, Eds., vol. 1368, nov 2011, pp. 131–134.

[16] R. AlSaad, Q. Malluhi, and M. Abouelhoda, "Efficient parallel implementation of the shrimp sequence alignment tool using mapreduce," in *Qatar Foundation Annual Research Forum Proceedings*, 2012.

[17] C.-M. Liu, T. Wong, E. Wu, R. Luo, S.-M. Yiu, Y. Li, B. Wang, C. Yu, X. Chu, K. Zhao, R. Li, and T.-W. Lam, *Bioinformatics*, vol. 28, no. 6, pp. 878–879, Mar. 2012.

[18] P. Klus, S. Lam, D. Lyberg, M. Cheung, G. Pullan, I. McFarlane, G. Yeo, and B. Lam, "Barracuda - a fast short read sequence aligner using graphics processing units," *BMC Research Notes*, vol. 5, no. 1, p. 27, 2012.

[19] Y. Liu, B. Schmidt, and D. L. Maskell, "Cushaw: a cuda compatible short read aligner to large genomes based on the burrows-wheeler transform," *Bioinformatics*, 2012.

[20] J. Zhang, H. Lin, P. Balaji, and W. chun Feng, "Optimizing burrows-wheeler transform-based sequence alignment on multicore architectures," in *CCGRID*, 2013, pp. 377–384.

[21] H. Li and R. Durbin, "Fast and accurate long-read alignment with burrowswheeler transform," *Bioinformatics*, vol. 26, no. 5, pp. 589–595, 2010.

[22] H. Xin, D. Lee, F. Hormozdiari, S. Yedkar, O. Mutlu, and C. Alkan, "Accelerating read mapping with fasthash," *BMC Genomics*, vol. 14, no. Suppl 1, p. S13, 2013.

[23] B. Langmead, K. Hansen, and J. Leek, "Cloud-scale rna-sequencing differential expression analysis with myrna," *Genome Biology*, vol. 11, no. 8, p. R83, 2010.

[24] M. K. Iyer, A. M. Chinnaiyan, and C. A. Maher, "Chimerascan: a tool for identifying chimeric transcription in sequencing data," *Bioinformatics*, vol. 27, no. 20, pp. 2903–2904, 2011.

[25] M. Cole, *Algorithmic Skeletons: Structured Management of Parallel Computation*. MIT Press & Pitman, 1989.

[26] M. Aldinucci and M. Danelutto, "Skeleton based parallel programming: functional and parallel semantic in a single shot," *Computer Languages, Systems and Structures*, vol. 33, no. 3-4, pp. 179–192, oct 2007.

[27] M. Aldinucci, M. Coppo, F. Damiani, M. Drocco, M. Torquati, and A. Troina, "On designing multicore-aware simulators for biological systems," in *Proc. of Intl. Euromicro PDP 2011: Parallel Distributed and network-based Processing*, Y. Cotronis, M. Danelutto, and G. A. Papadopoulos, Eds. Ayia Napa, Cyprus: IEEE, feb 2011, pp. 318–325.

[28] M. Aldinucci, M. Danelutto, and M. Torquati, "Fastflow tutorial," Università di Pisa, Dipartimento di Informatica, Italy, Tech. Rep. TR-12-04, mar 2012.

[29] M. Aldinucci, M. Danelutto, P. Kilpatrick, M. Meneghin, and M. Torquati, "Accelerating code on multi-cores with fastflow," in *Proc. of 17th Intl. Euro-Par 2011 Parallel Processing*, ser. LNCS, E. Jeannot, R. Namyst, and J. Roman, Eds., vol. 6853. Bordeaux, France: Springer, aug 2011, pp. 170–181.

[30] M. Aldinucci, M. Meneghin, and M. Torquati, "Efficient Smith-Waterman on multi-core with fastflow," in *Proc. of Intl. Euromicro PDP 2010: Parallel Distributed and network-based Processing*, M. Danelutto, T. Gross, and J. Bourgeois, Eds. Pisa, Italy: IEEE, feb 2010, pp. 195–199.

[31] M. Aldinucci, M. Danelutto, P. Kilpatrick, M. Meneghin, and M. Torquati, "An efficient unbounded lock-free queue for multi-core systems," in *Proc. of 18th Intl. Euro-Par 2012 Parallel Processing*, ser. LNCS, vol. 7484. Rhodes Island, Greece: Springer, aug 2012, pp. 662–673.

[32] S. Marco-Sola, M. Sammeth, R. Guigo, and P. Ribeca, "The GEM mapper: fast, accurate and versatile alignment by filtration," *Nat Meth*, vol. 9, no. 12, pp. 1185–1188, Dec. 2012.