Exploring Parallelism in Short Sequence Mapping Using Burrows-Wheeler Transform

Doruk Bozdağ*, Ayat Hatem*† and Umit V. Catalyurek*†
*Department of Biomedical Informatics
†Department of Electrical and Computer Engineering
The Ohio State University, Columbus, OH 43210
Email: {bozdagd,dayat,umit}@bmi.osu.edu

Abstract

Next-generation high throughput sequencing instruments are capable of generating hundreds of millions of reads in a single run. Mapping those reads to a reference genome is an extremely compute-intensive process that takes more than a day on a modern computer even when the accuracy of the results is traded off to speed up the execution. In this work, we explore various data distribution strategies for parallel execution of three state-of-the-art mapping tools, namely Bowtie, BWA and SOAP2, that are based on the Burrows-Wheeler Transformation. We report on the performance of these strategies and show that the best strategy depends on the input scenario as well as the relative efficiency of the tools in the indexing and matching steps of the mapping process. The parallelization strategies investigated in this paper are general and can easily be applied to different mapping algorithms. With the availability of parallel execution methods, it will be possible to carry out more intensive computations that cannot be accomplished in a reasonable time using sequential tools, including mapping with larger mismatch tolerance.

I. Introduction

Next-generation high throughput sequencing instruments, including Roche's 454, Illumina's Solexa and Applied Biosystem's SOLiD are capable of sequencing hundreds of millions of short sequences (reads) in a single run. Accurate and efficient mapping of these massive amounts of sequence data to a reference genome is the most time consuming step in many application workflows such as whole-genome resequencing, targeted sequencing, DNA methylation and ChIP sequencing. To address this challenging task many short sequence mapping tools have been proposed (Bowtie [1], BWA [2], SOAP [3], MapReads [4], MAQ [5], RMAP [6], SHRiMP [7], ZOOM [8], mrFAST [9], SOCS [10], PASS [11]). These tools are designed to take advantage of the fact that the sequences are short and therefore they allow only a limited number of mismatches and short gaps. Due to the variety of mapping techniques used, each of these tools offers a different trade-off between speed and quality of the results. Still, even by using the fastest tool and allowing loss of some quality, mapping hundreds of millions of reads to a mammalian genome takes about a day when executed on a single computer [1], [12]. Given that the goal is to sequence the entire human genome in 15 minutes by the year 2013 [13], parallel mapping strategies are needed to keep up with the pace of the technology.

In this paper, we consider the three newest and fastest sequence mapping tools, Bowtie [1], BWA [2] and SOAP2 [3], and explore the parallelism in these programs by evaluating effects of different data distribution strategies to speed them up without compromising the quality of the results. All of these tools are based on Burrows-Wheeler Transform (BWT) and follow a two-step mapping approach. In the first step, a BWT-based index is built for the reference genome. Then, in the second step reads are mapped to the genome by searching for the matching locations in the index structure. The accuracy of the mapping depends on several factors such as sequencing and reading errors or existence of SNPs and repetitive regions in DNA. Even though accounting for these factors improves the quality of the mapping, it also increases the computational cost dramatically. Therefore, similar to other sequence mapping programs, Bowtie, BWA and SOAP2 provide several options to compromise quality in the following ways to limit the computation time:

- Limiting the number of allowed mismatches,
- Ignoring indels or limiting their count and length,
- Ignoring/under-utilizing base quality score information,
- Limiting the number of reported matching locations,
- Ignoring information about errors particular to each sequencing technology.

In this respect, parallel processing is inevitable to keep the run-time low while achieving higher quality. Therefore, in this work we explore different data distribution strategies to parallelize Bowtie, BWA and SOAP2 programs and show that the best strategy depends on the input scenario as well as the relative efficiency of the tools in the two steps of the mapping process.

This work was supported in parts by the NIH/NCI grant R01CA141090; the DOE grant DE-FC02-06ER2775; by the NSF grants CNS-0643969, OCI-0904809, OCI-0904802 and CNS-0403342; and an allocation of computing time from the Ohio Supercomputer Center.

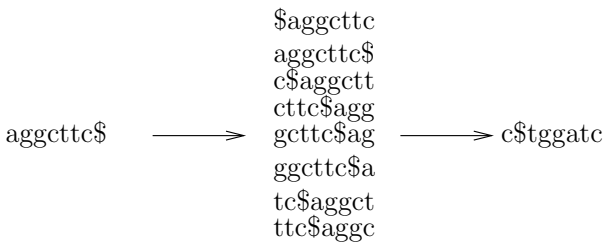


Figure 1. Construction of the BWT of the text **T**=aggcttc.

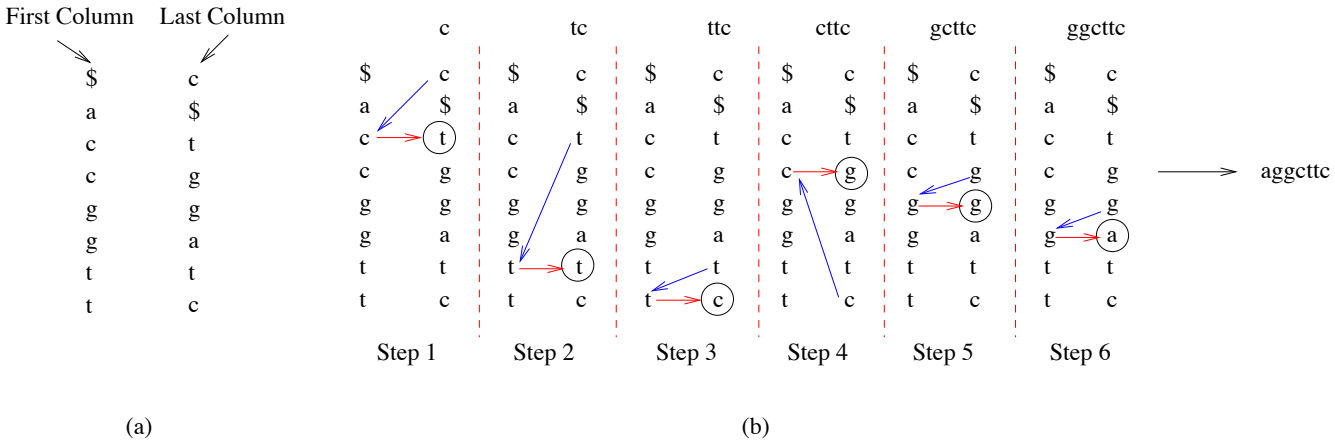


Figure 2. Regeneration of the original text **T**=aggcttc from the BWT of **T**.

II. Burrows-Wheeler Transform

Burrows-Wheeler Transform [14] of a given text **T** is a reversible permutation of the characters in **T**. BWT was primarily developed for data compression, however, it is widely used by data indexing techniques due to its efficiency. In particular, BWT allows searching through a given data block with a relatively small memory footprint. The basic idea is the construction of a sorted matrix **M** for all possible cyclic rotations of the text **T** in two steps. First, a character, denoted by $, is added to the end of **T**, where $ does not belong to the alphabet of **T** and is smaller than any other character in the alphabet. Next, the matrix **M** of all cyclic rotations of **T**$ is constructed. Once the rows of **M** are sorted, the last column of **M** becomes the output text of the BWT. Figure 1 shows an example for constructing the BWT of the text **T**=aggcttc. In general, the BWT process is reversible, i.e., the original text can be regenerated from the output text. In fact, the reverse process is based on the Last First (LF) mapping property of BWT. Using LF mapping, if a character X is ranked i^{th} among the rows ending with the same character in the last column, then it will have the same rank i among the same characters in the first column. Figure 2 shows how to use the LF mapping property to regenerate **T**=aggcttc from the BWT of **T**. First, the first column is generated from the output text by ordering the BWT text as shown in Figure 1. Then, starting with character c, we detect its location in the first column and retrieve the previous character which is t. After that, we detect the location of character t in the first column and find the corresponding previous character which is t. This process continues until the $ character is found which indicates the end of the string. Therefore, the original text can be regenerated. Accordingly, Ferragina and Manizini [15] provided an exact matching algorithm on BWT text based on the LF mapping property.

One of the key applications of BWT is sequence alignment in bioinformatics. In fact, several recent short sequence mapping tools, including Bowtie [1], BWA [2], and SOAP2 [3] were developed based on BWT. In BWA, a BWT based index is built using the method introduced by Lam et al. [16]. This method tries to mimic the top down traversal of a suffix tree using BWT. In addition, it uses the search algorithm of Ferragina and Manzini to find the exact matches through the index. The authors of BWA have also introduced a modified algorithm that allows for inexact matching. Bowtie, on the other hand, uses an index called FM which is again based on the BWT. The FM index uses the Ferragina and Manzini exact matching algorithm to search through the index. Furthermore, Bowtie provides

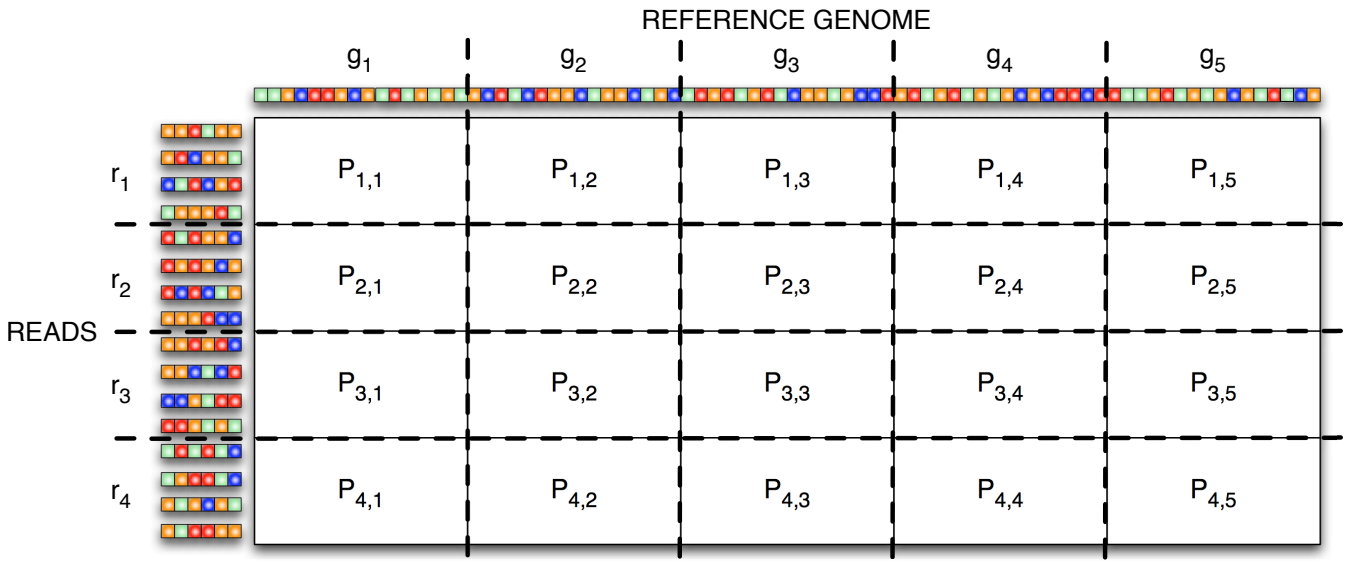


Figure 3. Example of an $NR \times NG = 4 \times 5$ data partitioning configuration. Each compute node $P_{i,j}$ is assigned the corresponding reads group r_i and genome part g_j of the 2D grid.

an excessive backtracking search method to find inexact matches through the reference genome. In SOAP2, the reads are split into $k+1$ parts while searching for inexact matches with up to k mismatches. Then, exact match is sought in at least one of the parts.

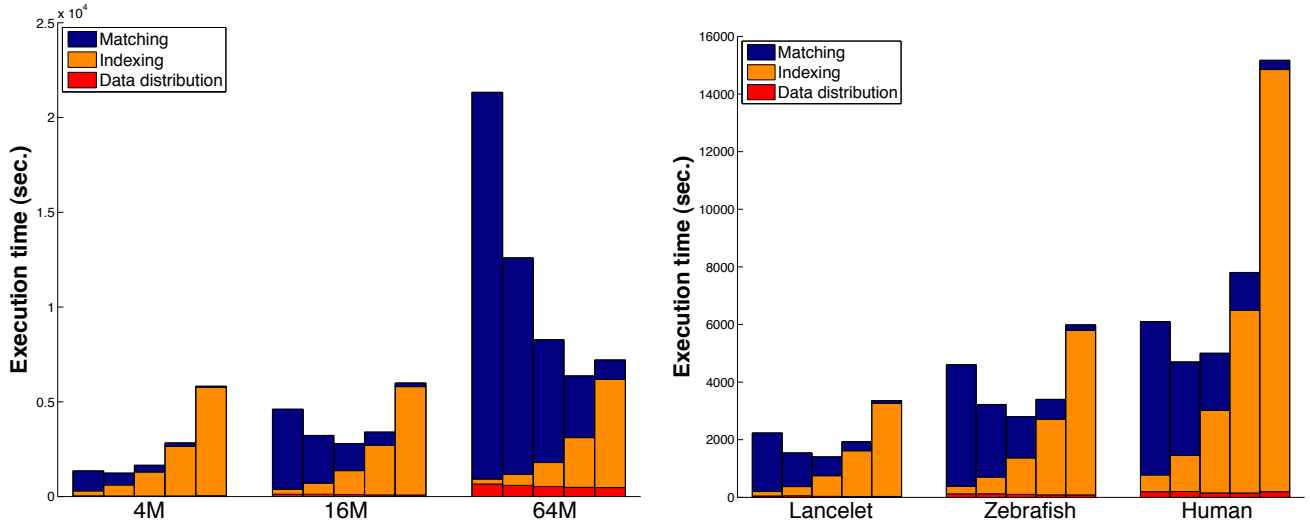
III. Parallelization Strategies

In this work, we focus on the parallelization of BWA, Bowtie and SOAP2 tools on a cluster of distributed memory computers. Both the indexing and the matching steps of these algorithms can be run in parallel by appropriate distribution of the genome and the short sequence data.

A common technique to parallelize the matching step is to use multiple threads to simultaneously execute independent portions of work on a multi-core computer. Basically, different blocks of reads are assigned to each thread. This approach is comparable to partitioning the reads and assigning each part to a different node in a compute cluster. However, in multi-threading the amount of parallelization is limited by the number of cores, and the efficiency decreases as the number of threads increases.

Bowtie, BWA and SOAP2 all offer multi-threading support to parallelize the matching step, however, due to the difficulty of parallel indexing, no such support exists for the indexing step. In this work, in addition to read partitioning to speed up the matching step, we also consider genome partitioning to parallelize the indexing step of the mapping process. By partitioning the genome and assigning only a portion of it to each node, the indexing step is expected to get faster due to smaller input size. Furthermore, the memory footprint is reduced, which can be critical in non-dedicated systems or in systems with limited memory especially when indexing large genomes. Note that, this effect cannot be achieved by multi-threading. Genome partitioning may also help improve the performance of the matching step, since the search time in a BWT-based approach also depends on the size of the index. Yet another benefit of cluster computing is the larger amount of aggregate cache, which is useful for both steps.

Partitioning the reads can be very useful for cases where a very large number of reads needs to be mapped to a small genome. On the other hand, partitioning the genome will be more efficient if a relatively small number of reads should be mapped to a large genome. Between these extreme cases better data partitioning schemes can be utilized. Here, we explore a more general two-dimensional (2D) data partitioning scheme, where the nodes of the compute cluster are viewed as a 2D grid (note that the 2D grid analogy is solely used for demonstrating the data partitioning scheme; the actual topology of the compute cluster is not required to be a grid). Let the number of rows in this grid correspond to the number of read partitions (N_R), and the number of columns correspond to the number of genome partitions (N_G), where $N_R \times N_G$ is equal to the number of nodes in the system. Under this partitioning, parallel execution works as follows. First, each reads group is multicast to the processors in the corresponding row, and each genome part is multicast to the processors in the corresponding column of the 2D grid (see Figure 3). Then, each processor independently runs the mapping algorithm to map the assigned group of reads to the assigned portion of the genome. At the end of the execution, a read may map to multiple locations. This information is



(a) Zebrafish (b) $16M$ reads

Figure 4. Bowtie execution time results on 16 nodes for mapping (a) $4M$, $16M$ and $64M$ reads to Zebrafish genome; (b) $16M$ reads to different sized reference genomes (lancelet: 0.9Gbp, zebrafish: 1.5Gbp, human: 3.1Gbp). Each group of five bars correspond to the following $N_R \times N_G$ configurations from left to right: 1×16, 2×8, 4×4, 8×2, and 16×1.

combined across the row processors and returned to the user. By choosing appropriate values for N_R and N_G based on the input data sizes and the relative performance of indexing and matching phases of the mapping tool being used, the overall performance can be improved.

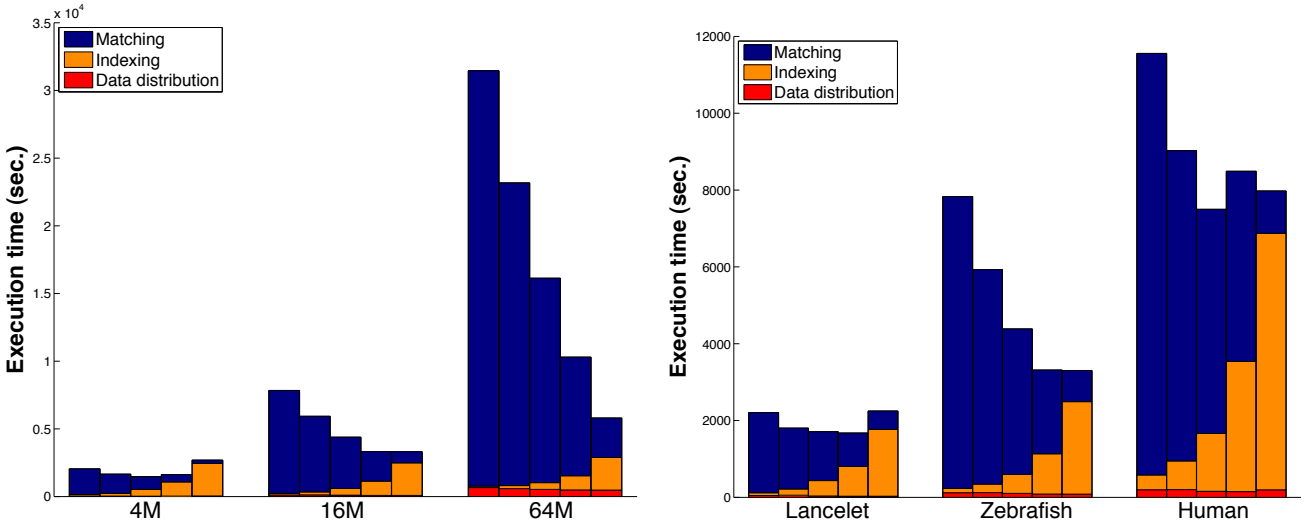
IV. Experimental Results

In this section, we present results from our experiments on different parallelization strategies for Bowtie, BWA and SOAP2 tools on a cluster of distributed memory computers. We used version 0.10.1 of Bowtie, version 0.5.0 of BWA, and version 2.20 of SOAP2 with their default options unless stated otherwise. For all of these tools the default value for the maximum number of mismatches allowed in the seed of a read was 2. We used 28 as the seed length for BWA and SOAP2, which is the default value for Bowtie. The experiments were conducted on a 32-node dual 2.4 GHz Opteron cluster with 8 GB of memory per node. The nodes are interconnected with switched gigabit Ethernet. In all of the tests reported in this section, we used two threads per node in the matching step of all three tools.

In our tests we used three reference genomes, human, zebrafish and lancelet, with sizes approximately 3.1Gbp, 1.5Gbp and 0.9Gbp, respectively [1]. As for the reads, we used synthetic datasets for scalability experiments generated by the *wgsim* tool as well as a real dataset. *wgsim* tool is a part of *SAMtools* package[2], and it generates sample reads for a given reference genome. In particular, for each genome in our testbed, we generated 70bp reads each with 0.09% SNP mutation rate, 0.01% indel mutation rate and 2% uniform sequencing base error rate. The real reads data were generated by a single run of a SOLiD system where total RNA was isolated from human brain tissue, converted to double stranded DNA templates suitable for sequencing using the Ambion's Legend technology. The library was clonally amplified by ePCR and run on Applied Biosystem's SOLiD instrument [17]. The final reads file has approximately 130M reads of length 50bp.

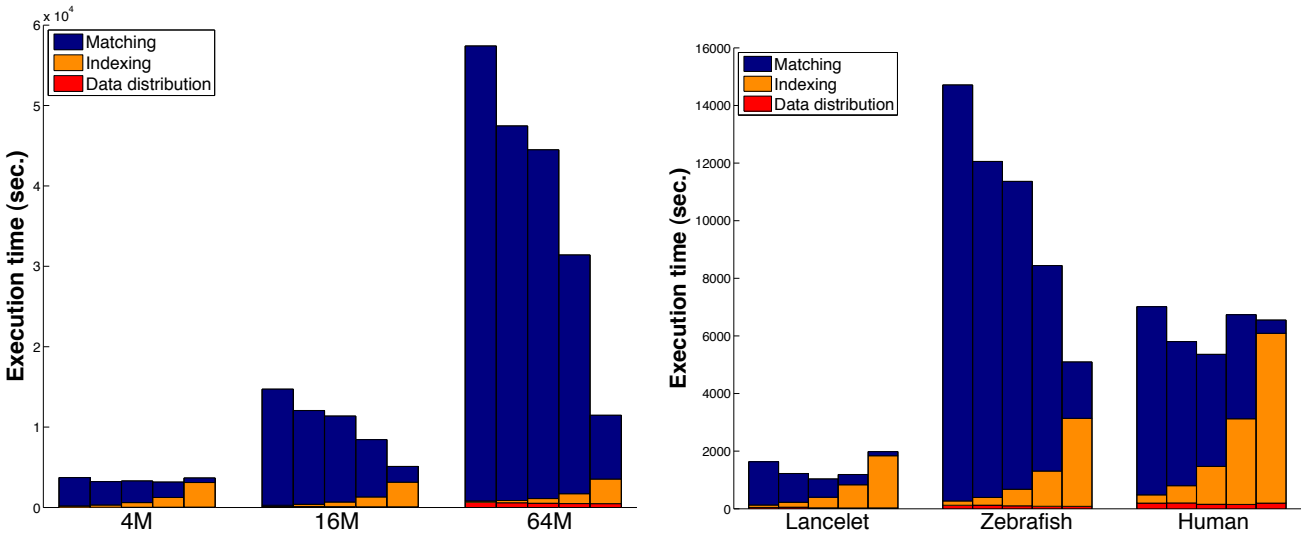
In our experiments on synthetic datasets, we used 16 compute nodes and tested the performance of the tools using the following read and genome distribution configurations: $N_R \times N_G = 1 \times 16, 2 \times 8, 4 \times 4, 8 \times 2$, and 16×1. For each configuration we measured the time to distribute reads and genome data to nodes as well as indexing and matching times of the mapping tools. In Figure 4(a), parallel execution time for Bowtie is given for mapping $4M$, $16M$ and $64M$ reads to zebrafish genome (M = million). Corresponding results for BWA and SOAP2 are given in Figures 5(a) and 6(a), respectively. The detailed timing results corresponding to those figures are also displayed in Tables I–III. From these charts, one can see that as the number of reads increases, matching time becomes more dominant relative to the indexing time. Therefore, using configurations with larger N_R becomes more efficient. However, the ratio of indexing time to matching time is also important in selecting the best configuration. As the results indicate, the ratio of indexing time to matching time for Bowtie is larger than that for BWA. Hence, Bowtie is likely to benefit more from larger N_G values relative to BWA. For instance, while mapping 16M reads to the zebrafish genome,

[1] http://genome.ucsc.edu
[2] http://samtools.sourceforge.net



(a) Zebrafish (b) $16M$ reads

Figure 5. BWA execution time results on 16 nodes for mapping (a) $4M$, $16M$ and $64M$ reads to zebrafish genome; (b) $16M$ reads to different sized reference genomes (lancelet: 0.9Gbp, zebrafish: 1.5Gbp, human: 3.1Gbp). Each group of five bars correspond to the following $N_R \times N_G$ configurations from left to right: 1×16, 2×8, 4×4, 8×2, and 16×1.



(a) Zebrafish (b) $16M$ reads

Figure 6. SOAP2 execution time results on 16 nodes for mapping (a) $4M$, $16M$ and $64M$ reads to zebrafish genome; (b) $16M$ reads to different sized reference genomes (lancelet: 0.9Gbp, zebrafish: 1.5Gbp, human: 3.1Gbp). Each group of five bars correspond to the following $N_R \times N_G$ configurations from left to right: 1×16, 2×8, 4×4, 8×2, and 16×1.

the best configuration for Bowtie was $N_R \times N_G = 4 \times 4$, whereas the best configuration for BWA and SOAP2 was $N_R \times N_G = 16 \times 1$. Note that these results also give insight about application scenarios in which the same index can be used multiple times for mapping different sets of reads to the same reference genome. If the total number of reads that will be mapped to the same reference genome is large, larger N_R values should be preferred.

To assess the impact of varying genome size, we designed a second set of experiments, where $16M$ reads generated by *wgsim* tool were mapped to the corresponding three reference genomes. The results for these tests are given in Figures 4(b), 5(b) and 6(b) for Bowtie, BWA and SOAP2 respectively. From these results one can observe that, for all of the tools, the indexing time is slightly more than doubled either when N_G is halved or when a two times larger

Table I

DETAILED TIMING RESULTS CORRESPONDING TO FIGURE 4 FOR EXECUTING BOWTIE IN PARALLEL USING 16 NODES. SEQUENTIAL EXECUTION TIME FOR EACH EXPERIMENT IS INCLUDED IN PARENTHESIS.

Zebrafish genome															
	4M reads (Index=7197, Match=951, Total=8148)					16M reads (Index=7197, Match=3878, Total=11075)					64M reads (Index=7197, Match=16438, Total=23635)				
	1x16	2x8	4x4	8x2	16x1	1x16	2x8	4x4	8x2	16x1	1x16	2x8	4x4	8x2	16x1
Data dist.	41	29	28	30	47	122	123	104	86	83	661	590	531	487	475
Indexing	259	580	1263	2624	5718	259	580	1263	2624	5718	259	580	1263	2624	5718
Matching	1051	627	356	176	50	4222	2519	1430	689	187	20404	11422	6478	3252	1010
Total	1351	**1236**	1648	2830	5815	4603	3222	**2797**	3398	5988	21324	12592	8272	**6362**	7204

16M reads															
	Lancelet genome (Index=3297, Match=2337, Total=5634)					Zebrafish genome (Index=7197, Match=3878, Total=11075)					Human genome (Index=20775, Match=3961, Total=24736)				
	1x16	2x8	4x4	8x2	16x1	1x16	2x8	4x4	8x2	16x1	1x16	2x8	4x4	8x2	16x1
Data dist.	46	55	34	29	30	122	123	104	86	83	198	202	156	150	196
Indexing	155	318	716	1588	3230	259	580	1263	2624	5718	580	1253	2856	6338	14653
Matching	2031	1168	654	315	98	4222	2519	1430	689	187	5314	3244	1988	1316	322
Total	2233	1541	**1405**	1932	3358	4603	3222	**2797**	3398	5988	6092	**4698**	5000	7805	15170

Table II

DETAILED TIMING RESULTS CORRESPONDING TO FIGURE 5 FOR EXECUTING BWA IN PARALLEL USING 16 NODES. SEQUENTIAL EXECUTION TIME FOR EACH EXPERIMENT IS INCLUDED IN PARENTHESIS.

Zebrafish genome															
	4M reads (Index=2297, Match=3071, Total=5368)					16M reads (Index=2297, Match=10938, Total=13235)					64M reads (Index=2297, Match=49676, Total=51973)				
	1x16	2x8	4x4	8x2	16x1	1x16	2x8	4x4	8x2	16x1	1x16	2x8	4x4	8x2	16x1
Data dist.	41	29	28	30	47	122	123	104	86	83	661	590	531	487	475
Indexing	109	222	494	1049	2408	109	222	494	1049	2408	109	222	494	1049	2408
Matching	1896	1402	939	546	231	7600	5583	3787	2179	811	30675	22353	15099	8758	2917
Total	2046	1652	**1462**	1626	2685	7831	5928	4384	3314	**3302**	31446	23165	16124	10294	**5800**

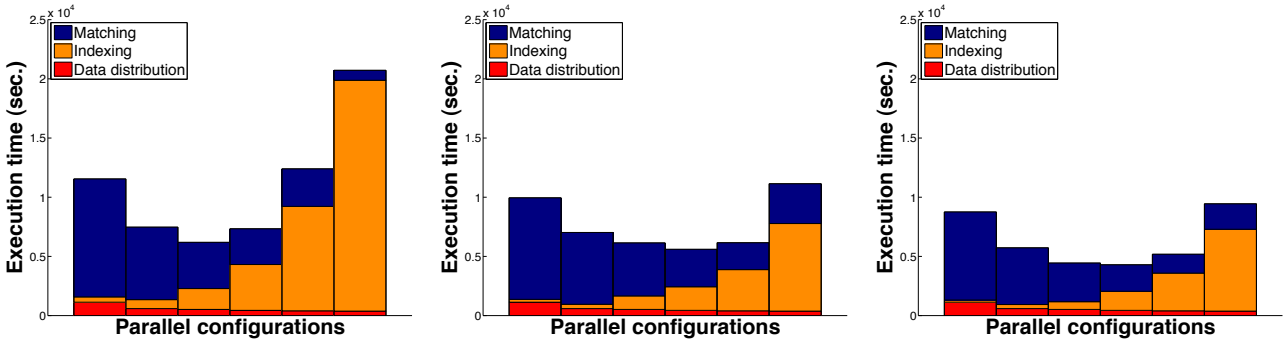
16M reads															
	Lancelet genome (Index=1613, Match=6785, Total=8398)					Zebrafish genome (Index=2297, Match=10938, Total=13235)					Human genome (Index=7096, Match=15650, Total=22746)				
	1x16	2x8	4x4	8x2	16x1	1x16	2x8	4x4	8x2	16x1	1x16	2x8	4x4	8x2	16x1
Data dist.	46	55	34	29	30	122	123	104	86	83	198	202	156	150	196
Indexing	74	163	401	777	1740	109	222	494	1049	2408	384	745	1506	3389	6680
Matching	2086	1584	1274	867	478	7600	5583	3787	2179	811	10973	8077	5834	4948	1100
Total	2206	1802	1709	**1673**	2248	7831	5928	4384	3314	**3302**	11555	9024	**7496**	8487	7975

Table III

DETAILED TIMING RESULTS CORRESPONDING TO FIGURE 6 FOR EXECUTING SOAP2 IN PARALLEL USING 16 NODES. SEQUENTIAL EXECUTION TIME FOR EACH EXPERIMENT IS INCLUDED IN PARENTHESIS.

Zebrafish genome															
	4M reads (Index=2657, Match=7221, Total=9878)					16M reads (Index=2657, Match=28522, Total=31179)					64M reads (Index=2657, Match=119499, Total=122156)				
	1x16	2x8	4x4	8x2	16x1	1x16	2x8	4x4	8x2	16x1	1x16	2x8	4x4	8x2	16x1
Data dist.	41	29	28	30	47	122	123	104	86	83	661	590	531	487	475
Indexing	150	272	574	1222	3056	150	272	574	1222	3056	150	272	574	1222	3056
Matching	3520	2908	2694	1897	561	14437	11659	10686	7133	1956	56590	46596	43382	29698	7931
Total	3711	3210	3297	**3150**	3664	14709	12054	11364	8441	**5095**	57401	47459	44488	31406	**11462**

16M reads															
	Lancelet genome (Index=1633, Match=1903, Total=3536)					Zebrafish genome (Index=2657, Match=28522, Total=31179)					Human genome (Index=6591, Match=5107, Total=11698)				
	1x16	2x8	4x4	8x2	16x1	1x16	2x8	4x4	8x2	16x1	1x16	2x8	4x4	8x2	16x1
Data dist.	46	55	34	29	30	122	123	104	86	83	198	202	156	150	196
Indexing	83	172	367	802	1811	150	272	574	1222	3056	286	600	1315	2975	5895
Matching	1504	995	631	357	142	14437	11659	10686	7133	1956	6530	5000	3887	3611	458
Total	1633	1222	**1032**	1188	1983	14709	12054	11364	8441	**5095**	7013	5802	**5359**	6736	6549



(a) Bowtie (b) BWA (c) SOAP2

Figure 7. Execution time results on 32 nodes for mapping $130M$ reads from our real data set to human genome for (a) Bowtie, (b) BWA and (c) SOAP2 programs. Each group of six bars correspond to the following $N_R \times N_G$ configurations from left to right: 1×32, 2×16, 4×8, 8×4, 16×2, and 32×1.

Table IV
DETAILED EXECUTION TIME RESULTS ON 32 NODES FOR MAPPING 130M READS FROM OUR REAL DATA SET TO HUMAN GENOME USING BOWTIE, BWA AND SOAP2 PROGRAMS. THE RESULTS IN THIS TABLE CORRESPOND TO THE ONES IN FIGURE 7.

	Bowtie				BWA				SOAP2			
Configuration	Data dist.	Index	Match	Total	Data dist.	Index	Match	Total	Data dist.	Index	Match	Total
1x32	1131	447	9961	11538	1131	243	8572	9944	1131	184	7434	8748
2x16	592	757	6118	7466	592	364	6065	7020	592	344	4784	5719
4x8	519	1766	3901	**6186**	519	1142	4473	6134	519	653	3262	4434
8x4	439	3874	3018	7331	439	1994	3164	**5597**	439	1615	2232	**4286**
16x2	405	8820	3165	12390	405	3476	2270	6151	405	3183	1590	5178
32x1	377	19500	838	20715	377	7398	3357	11132	377	6915	2148	9440

reference genome is used. This indicates that partitioning the genome may help speeding up the indexing phase more than expected. On the other hand, when $N_R \leq 8$, the matching time is less than doubled when N_R is halved. This can be explained by the fact that the matching time not only depends on the number of read sequences assigned to a node but also on the index size. The index size at each node increases with increasing N_R (since N_G is decreased), which results in an increase in the matching time. As a result, one can conclude that parallelization of the indexing step may also help reducing the execution time due to reduced index size. The only special case for this observation occurs for the 16×1 configuration. In this case, the matching time is much lower than what can be predicted from the other configurations. Since $N_G = 1$ in this configuration, all of the processors use the same reference genome. In the other configurations, where $N_G > 1$, the matching time is larger than expected due to an additional imbalance caused by genome partitioning. Although the size of the genome segments assigned to the nodes are equal, some segments get more hits than the others resulting in this imbalance.

In the case of zebrafish experiments using SOAP2, the matching time was found to be significantly larger than expected. After additional tests we concluded that this is due to the combination of the fact that the zebrafish genome had many fewer ambiguous characters than the human and the lancelet genomes (0.2% as opposed to 7% and 10%) and the way that SOAP2 handle ambiguous characters during matching.

Tables I–III, also include sequential execution times, in parenthesis, for each scenario we ran. As seen in the tables, parallel execution always leads to speedup but it is far from ideal linear speedup. On 16 nodes, speedup ranges from 2.18 to 10.65. Please also note that, the matching time for almost all of the configurations, including 1x16, is better than the sequential matching time. Therefore, even if the indexing is done once for the different genomes, the parallelization helps in executing the experiments faster. Obviously, if the same reference genome will be used multiple times to match against multiple reads dataset, one should prefer partitioning reads (e.g., 16x1 configuration in our tests), to achieve best, almost linear, speedup.

In our experiments on the real reads data, we used the following configurations using 32 nodes to evaluate the parallel performance of the tools: $N_R \times N_G = 1 \times 32, 2 \times 16, 4 \times 8, 8 \times 4, 16 \times 2$, and 32×1. The results of these experiments are illustrated in Figure 7 and detailed in Table IV. The outcome of these experiments were in line with those from our synthetic datasets. The best configuration for Bowtie for this particular test case was $N_R \times N_G = 4 \times 8$, whereas it was $N_R \times N_G = 8 \times 4$ for BWA and SOAP2.

V. Conclusion and Future Work

In this work, we extensively analyzed different data distribution strategies for mapping massive amounts of reads to a reference genome in parallel. We considered partitioning both the reads and the genome data on different number of parts for mapping on a compute cluster arranged as a 2D grid. The parallelization strategies are tested on three new and fast mapping algorithms: Bowtie, BWA and SOAP2, all of which are based on the Burrows-Wheeler Transform. The results revealed that the best data distribution strategy depends on the input scenario as well as the relative efficiency of the tools in the indexing and matching steps of the mapping process. Furthermore, the size of the genome part assigned to each node turned out to have a significant impact on the performance of the tools in the matching phase.

The parallelization techniques investigated in this paper can easily be applied to different mapping algorithms to speed up their execution time and to reduce memory requirement without compromising the accuracy of the results. With the availability of parallel execution methods, it will be possible to carry out more intensive computations that cannot be accomplished in a reasonable time using sequential tools, including mapping with larger mismatch tolerance. In the future, we plan to investigate analytical cost models for parallel execution of the mapping tools considered in this paper in order to automatically predict the best data partitioning strategy for given data sizes and number of available nodes.

References

[1] 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, pp. R25+, 2009. [Online]. Available: http://dx.doi.org/10.1186/gb-2009-10-3-r25

[2] H. Li and R. Durbin, "Fast and accurate short read alignment with burrows-wheeler transform." *Bioinformatics (Oxford, England)*, vol. 25, no. 14, pp. 1754–1760, July 2009. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btp324

[3] 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 (Oxford, England)*, vol. 25, no. 15, pp. 1966–1967, August 2009. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btp336

[4] *mapreads: SOLiD System Color Space Mapping Tool*, Applied Biosystems, http://solidsoftwaretools.com/gf/project/mapreads/.

[5] H. Li, J. Ruan, and R. Durbin, "Mapping short dna sequencing reads and calling variants using mapping quality scores." *Genome research*, vol. 18, no. 11, pp. 1851–1858, November 2008. [Online]. Available: http://dx.doi.org/10.1101/gr.078212.108

[6] A. D. Smith, Z. Xuan, and M. Q. Zhang, "Using quality scores and longer reads improves accuracy of solexa read mapping." *BMC bioinformatics*, vol. 9, no. 1, pp. 128+, February 2008. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-9-128

[7] 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, no. 5, pp. e1 000 386+, May 2009. [Online]. Available: http://dx.doi.org/10.1371/journal.pcbi.1000386

[8] H. Lin, Z. Zhang, M. Q. Zhang, B. Ma, and M. Li, "ZOOM! zillions of oligos mapped." *Bioinformatics (Oxford, England)*, vol. 24, no. 21, pp. 2431–2437, November 2008. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btn416

[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, October 2009. [Online]. Available: http://dx.doi.org/10.1038/ng.437

[10] B. D. Ondov, A. Varadarajan, K. D. Passalacqua, and N. H. Bergman, "Efficient mapping of applied biosystems solid sequence data to a reference genome for functional genomic applications," *Bioinformatics*, vol. 24, no. 23, pp. 2776–2777, December 2008. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btn512

[11] D. Campagna, A. Albiero, A. Bilardi, E. Caniato, C. Forcato, S. Manavski, N. Vitulo, and G. Valle, "PASS: a program to align short sequences," *Bioinformatics (Oxford, England)*, vol. 25, no. 7, pp. 967–968, April 2009. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btp087

[12] D. Bozdag, C. C. Barbacioru, and U. Catalyurek, "Parallel short sequence mapping for high throughput genome sequencing," in *Proc. of the International Parallel and Distributed Processing Symposium*, May 2009.

[13] K. Davies, "Pacific Biosciences preparing the 15-minute genome by 2013," *Bio IT World*, Feb. 2008.

[14] M. Burrows and D. J. Wheeler, "A block sorting lossless data compression algorithm," *Technical report 124 Palo Alto, CA: Digital Equipment Corporation*, 1994.

[15] P. Ferragina and G. Manzini, "Opportunistic data structures with applications," in *Proc. of the 41st Annual Symposium on Foundations of Computer Science*, 2000.

[16] T. Lam, W. Sung, S. Tam, C. Wong, and S. Yiu, "Compresses indexing and local alignment of dna," *Bioinformatics*, vol. 24, no. 6, pp. 791–797, 2008.

[17] "Applied Biosystems SOLiD system," http://marketing.appliedbiosystems.com/mk/get/SOLID_KNOWLEDGE_LANDING.