Peters et al., J Appl Bioinform Comput Biol 2012, 1:1
http://dx.doi.org/10.4172/jabcb.1000101

Journal of Applied Bioinformatics & Computational Biology

Research Article

A SCITECHNOL JOURNAL

Speeding Up Large-Scale Next Generation Sequencing Data Analysis with *pBWA*

Darren Peters[1], Xuemei Luo[2], Ke Qiu[1] and Ping Liang[2]*

Abstract

Newly available DNA sequencing technologies can generate billions of DNA sequences in a single machine run, making it feasible to obtain an individual's entire genome sequences very quickly and at a very affordable cost. These personal genome sequences can be used to identify genetic variations associated with variable traits, but they must first be aligned to a reference genome sequence using computer algorithms that make use of approximate string matching methods. The process of aligning a very large number of short sequence reads is computationally expensive and has led to the development of many sequence mapping programs. While new software, such as *BWA*, *SOAP2*, and *Bowtie,* efficiently align large numbers of short DNA sequences, we proposed that parallel computing would provide additional speedup in data processing to accommodate the rapidly increasing sequencing throughput. For this purpose, we developed *pBWA*, an efficient parallel version of *BWA,* based on the *Open MPI* library. *pBWA* retains the multi-threading capability provided by *BWA* while adding efficient parallelization for its core alignment functions. We have shown that *pBWA's* wall-time speedup is bounded only by the size of the parallel system. *pBWA* can run in both parallel and multi threaded environments simultaneously, allowing it to be tailored to run on parallel systems of all configurations and sizes. By taking the advantage of high performance computing cluster, the use of *pBWA* can cut down the computing wall-time for the reference alignment step from weeks to hours for extremely large DNA sequence data. We expect that the availability of *pBWA* should facilitate the analysis of large-scale genome sequencing data generated by the new generations of sequencing technologies. The source code and detailed user manual of *pBWA* are freely available at http://sourceforge.net/projects/pbwa.

Keywords: Next-Generation sequencing; Burrows-Wheeler Transform (BWT); Parallel computing; Cluster computing; Short sequence alignment

Abbreviations

BWT: Burrows-Wheeler Transform; SNP: Single Nucleotide Polymorphism; SV: Structure Variants; NGS: Next Generation Sequencing; SHARCNET: Shared Hierarchical Academic Research Computing Network

Introduction

Genetic variations in forms of single nucleotide polymorphisms (SNPs) and structural variants (SVs) are known to exist very commonly among individuals and populations [1]. These variations influence how individuals differ not only in their physical appearance, but also in their risk of disease and their response to therapeutic treatments [2-4]. The advent of next generation sequencing (NGS) technologies has made the survey of genetic variations at the genome level feasible by permitting sequencing of the genomes to be very fast and cheap [5-8]. Determination of genetic variants from personal genome data represents a challenging task due to the short length and the lack of order for the generated sequences. In the first step of the data analysis, computer algorithms performing sequence alignment are used to determine the locations of these short sequences within the reference genome. These algorithms must balance speed and accuracy, as improvements to one almost always comes at the cost of the other. Until recently, most sequence alignment software has been inadequate to address the massive amount of data generated by NGS and there has been great need for better software that can effectively handle this increase in production. Newer generations of software, such as *MAQ* [9], *BWA* [10], *SOAP* [11], *Bowtie* [12],and *mrFast* [13,14] were developed to efficiently align large amounts of short DNA sequences generated by the earlier generation of NGS platforms that are short in length with the number of reads in millions [15,16]. However, NGS platforms have been evolving very rapidly, pushing the sequencing capacity at a dramatic speed. For example, newer platforms, such as Ion Proton [17], Helicos [18], and PacBio [19], as well as the newer version of Illumina HiSeq platform, are now able to offer a throughput of billions of sequence reads daily. Such new levels of sequencing capacity call for further speedup in the sequence alignment step. To address this challenge, we have to inevitably go with high performance computing since speedup via algorithmic improvement may be limited. The speedup on HPC (High Performance Computing) can be achieved via either input data splitting or parallel computing. In comparison with the input data splitting, in which extremely large datasets are first split into a large number of smaller sets, with each processed individually by distributing them over large computer clusters followed by concatenating their results via shell scripts, we reason that parallel computing should offer more efficient workflow and more convenient use of HPC. This motivated us to design and implement *pBWA*, a parallel version of the short sequence alignment tool, BWA.

While parallel computing should in theory provide speedup to any of the above-mentioned NGS aligners, and all aligners have their own advantages and disadvantages [20], we had to inevitably decide on one aligner for a parallel implementation. The most important criterion we required was that the software needed to be open source, as we would be modifying the source code to create a parallel implementation. This led to the immediate dismissal of SOAP2 as a candidate, as the source code for SOAP2 is not publicly available. Another important criteria we considered was that the NGS aligner we chose should align input reads to an indexed genome sequence, since in this case, genome indexing only needs to be done once per genome, while input read indexing needs to be done for each dataset and the size of the reference genome is usually much smaller than the sequences to be aligned. This led us to exclude MAQ and mrFast as candidates for parallelization as they index the input

***Corresponding author:** Dr. Ping Liang, PhD, Department of Biological Sciences, Brock University, St. Catharines, Ontario, L2S 3A1, Canada, Tel: 905-688-5550; Fax: 905-658-1855; E-mail: pliang@brocku.ca

Received: October 10, 2012 **Accepted:** November 23, 2012 **Published:** November 28, 2012

reads instead of aligning to an indexed reference sequence. Between the two remaining candidates (Bowtie and BWA) we chose *BWA* for parallelization for its excellent overall performance and compatibility with different NGS platforms and downstream analysis, as well as its popularity in the NGS community [20,21]. Another large factor in choosing *BWA* over Bowtie was that *BWA* only multi-threads one of the two alignment steps, meaning it would be better served by a parallel implementation than Bowtie, which multi-threads its entire process. *BWA* is the successor to MAQ [9], developed by Li et al. [10]. Its aim was to improve MAQ by allowing gapped alignment while also giving a significant speed increase for large genomes such as the human genome. It was written in the C programming language, and is available as a command-line tool that can be run on a standard desktop computer due to its moderate memory requirement (approximately 3GB for the human genome). While having multiple functions, *BWA* is mainly used for short read alignment using a reference sequence indexed by the Burrows-Wheeler transform (BWT) [22].

Parallel applications are programs that can run on any number of processors simultaneously. To facilitate this, they must make use of a message-passing interface (MPI) library, which provides functions facilitating the passing of data (or "messages") from one processor to another [23,24]. We developed *pBWA*, a parallel version of BWA, using the MPI library. Like any other parallel applications built on an MPI library, *pBWA* differs from *BWA* running under multi-threading in that parallel processes do not share variables or memory, thus each parallel process must go through the same variable initialization and file I/O. This is the weakness and strength of *pBWA* or any other parallelized applications. It is a weakness in that the amount of RAM required to run *pBWA* is linearly scalable to the number of processors running *pBWA*. It is a strength in that unlike *BWA*, which can only be run on one processor with a multi-threading option available only for the *aln* command (+ *t-1* threads, where *BWA* is running on a t-core processor), *pBWA* can be run on as many processors as available for both *aln* and samse/sampe commands. *pBWA* can also make use of multi-threading, making more efficient use of RAM while providing the parallel functionality. This is where *pBWA* can achieve great reductions in elapsed wall-time when aligning massive sequencing datasets.

Materials and Methods

BWA

Short read alignment using BWA is broken down into three main components, each with its own *BWA* command. The first component is executed with the *index* command. This component takes the reference genome and indexes it by applying the Burrows-Wheeler transform and additionally generating a few other auxiliary data structures. This step needs to be done only once for a given reference genome and it can be used for all later mapping tasks for the same genome and shared among different computers by simply copying the files. Therefore, there is essentially no need to parallelize this step. The second component is executed with the *aln* command, which takes a set of short DNA reads and calculates their suffix array intervals based on their relation to the BWT. The last component, executed by commands *samse/sampe* (for either single or paired-end reads), takes each short read and generates its chromosomal coordinates based on the previously calculated suffix array intervals. These coordinates are output in the SAM file format [25]. The last two steps consume most of the time required for aligning the sequences to a reference genome by *BWA*, thus they are the targets for parallelization.

Parallel programming

We explain in the following sections the issues and strategies we used to parallelize *BWA*. It should be noted here that each stage (*aln,* and *samse/sampe*) of *pBWA* must be executed with the same number of processors for each run of the program, however the number of threads per processor can vary. This is to keep the output files matched up with the input sequences across different stages of the analysis.

Index distribution

Due to the fact that parallel processes do not share variables and RAM, much of the initialization process performed by *BWA* had to be modified to facilitate a parallel implementation. Before *BWA* can begin to perform sequence alignment, it must read in index files generated by running the *index* command. For large genomes, such as the human genome, these index files are extremely large, and this process can lag considerably when *p* processors are simultaneously trying to read from the same large file. *pBWA* handles this process by designating one processor as the *master* processor. Only the master processor reads the index files into RAM. After this is complete, the master processor performs a broadcast of the index to all other processors using an MPI function. This broadcast is facilitated in a binary tree structure, resulting in an execution time of *O (log p)*, where *p* is equal to the number of processors executing *pBWA*. This data distribution approach is known as the master-slave paradigm [26].

Sequence distribution

After the index has been read, *BWA* typically begins the alignment by reading DNA sequences from the beginning of a *FASTQ* file. *pBWA* achieves most of its speedup by distributing sequences across processors, such that each processor only performs alignment for *N/p* sequences, where *N* is the number of reads in the *FASTQ* file. In order for this sequence distribution to take place, the master processor must first scan the *FASTQ* file, creating an index of the input sequences on the fly and sending processor *i* its current file position after *(i*N/p)* reads have been scanned. Processor *i* then jumps to its designated file location and performs alignment for *N/p* sequences. An alternate strategy of sequence distribution would be to have all *p* processors reading from the *FASTQ* file in rounds. This prevents the time wasted while the master processor is indexing the sequence input file, but can lead to uneven sequence distribution as processors that are ready earlier may snatch up all of the sequences for alignment before other processors get a chance to claim any. For this reason, we decided to take the first option. Since the sequence distribution process requires random file access, *pBWA* unfortunately does not support compressed *FASTQ* files as input files. At this step, for paired reads (paired end or paired mate reads), *pBWA* provides an option, which is not available in *BWA*, to allow two *FASTQ* input files to be supplied on the same *aln* command execution. In this case, the program automatically generates index files by adding "_1" and "_2" to the sai index files for the first and second input *FASTQ* files, respectively. This option provides additional speedup by saving the time of reading and distributing the index files for the 2nd input file.

Improved threading

While implementing *pBWA*, we made an improvement to the use of multi-threading for its *aln* command. Multi-threading in *BWA* version 0.5.9 is facilitated by a sequence of variable locking and sequence assignments, both of which adversely affect the efficiency of the multi-threading. When executed with 24 threads, *BWA* showed only a 12-fold speedup in wall-time execution. In *pBWA* multi-threading is based only on a loop counter. When executed with 24 threads, *pBWA* shows a 16-fold speedup in wall-time execution, representing a 33% increase in efficiency over the original *BWA*. This improvement has now been incorporated in the newer version of *BWA*.

Final alignment output files and *samse/sampe*

Since parallel applications do not share file pointers, variables, and RAM, each process executing *pBWA* receives its own output file for all alignment functions within *BWA*. After the final alignment files have been output, they can easily been concatenated back together using a simple shell command (*cat*). The auxiliary files generated by *pBWA* can then be removed to clean up the execution folder. A run-time option [-M] is provided for *samse/sampe* to output a single output file for all processors at a small penalty to performance on certain systems. This single output file is generated using parallel file I/O, which is facilitated via non-blocking write operations. If processor *i* has completed its alignments prior to processor *i*-1, processor *i* will make the non-blocking write call and move forward to continue its next batch of alignments. Once processor *i* receives the correct file position from processor *i*-1, the non-blocking write will complete.

We provide at *pBWA* project for Source Forge (http://pbwa.sf.net) the detailed user instructions and examples of running *pBWA* for systems on the Shared Hierarchical Academic Research Computing Network (SHARCNET). We recommend users to read the information before adopting *pBWA* for a different HPC system, which may use a different job submission system than SHARCNET.

Results and Discussions

pBWA was tested for assessing its speedup and efficiency on a variety of datasets and computing clusters of varying sizes with varying running parameters, including different combinations of parallelism and multi-threading. The first computing cluster used to test *pBWA* was the *requin* cluster on SHARCNET. This cluster has 768 computation nodes running at 2.6 GHz, each with 8 GB memory and 2 cores. Since *requin* lacks a large number of cores per node, we used another cluster which has a larger number of cores to compare between pure multi-threading and parallelization and test the optimal combination of parallelization and multi-threading. In this case, we tested *pBWA* on the *orca* cluster on the SHARCNET, which has 320 computation nodes running at 2.2 GHz, each with 32 GB memory and 24 cores. Tests were run with datasets of 5, 25, 50, and 100 million 36 bp Illumina paired end reads, allowing two mismatches. An additional test was run on *orca* with a dataset of mouse deep whole genome sequence with approximately 350 million 50 bp SOLiD paired mate reads allowing three mismatches. Each test was repeated three times and the average time was used for comparison. This is to avoid occasional unusual behavior of parallelization we experienced on one cluster due to cluster instability, in which the failure of one processor could hold up the completion of the entire process. The Illumina reads are human sequences using UCSC hg19/NCBI GRCh37 as the reference genome, while the ABI SOLiD reads are mouse sequences using the UCSC mm9/NCBI build 37 as the reference genome. Both sets of sequences were paired reads, meaning that the *aln* command is run with each pair of files, followed by the *sampe* command to pair the resulting suffix array intervals. Because there is no multi-threading support for *samse/sampe*, this step in each parallel instance was run as single-threaded as in *BWA*. The time used in each of the two steps and the two steps combined (running *aln* and *sampe)* were recorded for each test option. The speedup for each step was calculated in relation to running the processes as a single thread on a single core and was calculated for each step individually and for all steps combined. Tests were also performed to combine multi-threading and parallelization. For these tests, the number of processors, alongside the number of threads *each processor* spawns, was indicated. The total number of executed threads is equal to the number of processors multiplied by the number of threads per processor. The results of a few representative test runs are provided in Tables 1-4 and Figure 1.

Comparison between multi-threading and parallelization

As a way of measuring the efficiency of *pBWA*, we compared the wall-time used between one process with 24 threads and 24 processors each with one thread, thus the same of number of total threads. The jobs were run on the orca cluster, which allows a maximal 24 threads per node. As we can see from Table 1 and Figure 1, at all data sizes examined, running *pBWA* in pure parallelization provides speedup close to multi-threading for the *aln* step. However, when the *sampe* step is included, parallelization achieves 5-6 times more speedup than multi-threading. This is expected, since no multi-threading is available for sampe. This is where *pBWA* offers a significant advantage over *BWA* when running parallelization.

For paired-end reads, the use of option to supply two *FASTQ*

Table 1: Wall-time and speedup for *pBWA* with 100 million paired 36 bp reads on orca.

	1 T[1]	24 T @ 1 P[2] /speedup	24 P @ 1 T /speedup	48 P @ 1 T /speedup	96 P @ 1 T /speedup	240 P @ 1 T /speedup
aln 1[3]	420 m	26 m/16.2	33.5 m/12.5	17.5 m/24	9.3 m/45.2	5.3 m/79.2
aln 2	533 m	28 m/19.0	34 m/15.7	16.8 m/31.7	9.3 m/57.3	5.8 m/91.2
sampe	688 m	688 m/1	53.3 m/12.9	32 m/21.5	22.5 m/30.6	17.5 m/39.3
Totals	**1641 m**	**742 m/2.2**	**120 m/13.7**	**66 m/24.7**	**40.8 m/40.2**	**28.8 m/57.0**
Efficiency[4]	**1**	**0.09**	**0.57**	**0.52**	**0.42**	**0.24**

[1]number of threads; [2]number of processors; [3]time used for running *aln* for each of the pair read file; [4]efficiency calculated based on the combined time in minutes (m) or seconds (s) of *aln* and *sampe* commands, and is calculated as speedup divided by the number of threads or processors or the combined total threads.

Table 2: *pBWA* executed with 5 million paired 36bp reads on requin.

	1 T	2 T/speedup	50 P/speedup	100 P /speedup
aln 1	1294 s	703 s/1.8	133 s/9.7	121 s/10.7
aln 2	1253 s	713 s/1.8	130 s/9.6	114 s/11.0
sampe	2513 s	2513 s/1	337 s/7.5	226 s/11.1
Totals	**84.5 m**	**65.5 m/1.3**	**10 m/8.5**	**7.5 m/11.3**
Efficiency	**1**	**0.6**	**0.17**	**0.11**

Table 3: *pBWA* executed with 5 million paired 36bp reads on orca.

	1 T	24 P/speedup	48 P/speedup	96 P/speedup	240 P/speedup
aln 1	1675 s	99 s/16.9	63 s/25.9	50 s/33.5	78 s/21.5
aln 2	1542 s	93 s/16.6	66 s/23.4	44 s/35.0	69 s/22.3
sampe	2128 s	307 s/6.9	199 s/10.7	145 s/14.7	123 s/17.3
Totals	**89 m**	**8.5 m/10.5**	**5.5 m/16.2**	**4 m/22.3**	**4.5 m/19.8**
Efficiency	**1**	**0.44**	**0.34**	**0.23**	**0.08**

Table 4: *pBWA* executed with ~350 million paired 50 bp reads on orca.

	1 T	24 P/speedup	48 P/speedup	96 P/speedup	240 P/speedup	48 P @ 5 T (240 T)/speedup	240 P @ 12 T (2880 T)/ speedup
aln 1	7611 m	606 m, 12.6	294 m, 25.9	140 m, 54.4	62 m, 122.8	66 m; 115.3	13 m, 585.4
aln 2	6950 m	495 m, 14.0	253 m, 27.5	124 m, 56.0	55 m, 126.4	59 m, 117.8	12 m, 579.2
sampe	520 m	67 m, 7.7	34 m, 15.3	24 m, 21.7	16 m, 32.5	34 m, 15.3	16 m, 32.5
Totals	**15081 m**	**1168 m/12.9**	**581m/26.0**	**288 m/52.4**	**132 m/114.3**	**159m/94.8**	**41 m/367.8**
Efficiency	**1**	**0.53**	**0.54**	**0.55**	**0.47**	**0.40**	**0.13**

input files at once achieves approximately another 10% wall-time reduction (based on a test with ~150 million 100 bp paired end reads) by eliminating the reading and distribution of index files for the 2nd input file (data not shown). Therefore, the use of this option is recommended for aligning paired-end reads. For the same reason, merging multiple FASTQ subsets for the same sample into one set (e.g. a human whole genome deep sequence data generated using the Illumina HiSeq 2000 will usually have multiple pairs of FASTQ files, each from one sequencing channel) before running *pBWA* would provide further wall-time reduction. By doing so, it also reduces the number of files to handle for downstream steps and eliminates the need of merging the multiple SAM/BAM files for the same sample, a process that is necessary for most types of sequence analyses and requires the use of special tools, such as Sam tool's merge command [25] or Picard's Merge Sam Files tool (http://sourceforge.net/projects/picard/).

Speedup using increasing numbers of processors

Comparisons between tests on different numbers of processors can be drawn from any of the tables individually. It can be seen that in general, as the number of processors increases, the speedup increases. For larger datasets, the speedup shows a close to linear relationship with the increase of processor after 24 processors (Tables 1, 4 and Figure 1). For example, for the *aln* step, the doubling of processors from 24 to 48 for 350 million reads achieved slightly more than doubled speedup (speedup is 25.9 and 54.4, respectively), and the same is true for speedup from 48 processors to 96 processors and from 96 to 240 processors (Table 4). This is true for the *sampe* step as well. In comparison, the speedup using 24 processors only achieved a speedup between 13 and 14, rather than 24 for the *aln* step. This is likely due to the initial significant overhead cost of parallelization, which does not seem to have a visible increase with the increase of processors. The speedup from 1 processor to 24 processors for the *sampe* step is lower than for the *aln* step, and is less than linear for further increase of processors (Tables 1-4), suggesting a larger initial overhead cost at this step. For smaller datasets, speedup can actually decrease after an increase in processors over a certain point due to the miniscule amount of time actually spent performing sequence alignment in comparison to the amount of time required to initialize each processor. For example, for the 5 million reads, running with 240 processors ended up taking more time for the *aln* step than with 96 processors (Table 3). For this reason, the maximal level of parallelization to use is determined by the size of dataset to ensure a minimal number of sequence reads per processor (e.g., 100,000 reads/processor or more; see more detailed discussion in later sections).

Comparing the speedup and efficiency among the equivalent columns in Tables 1, 3 and 4 and Figure 1 can draw comparisons between tests on different data sizes. It is shown in our results that as the size of the dataset increases, the speedup and efficiency increases when using the same number of processors. This is due to the fact that as the dataset size increases, more time is spent on performing sequence alignment in relation to time spent initializing each processor and communicating between them. We notice that efficiency does not seem to plateau across large increases in dataset

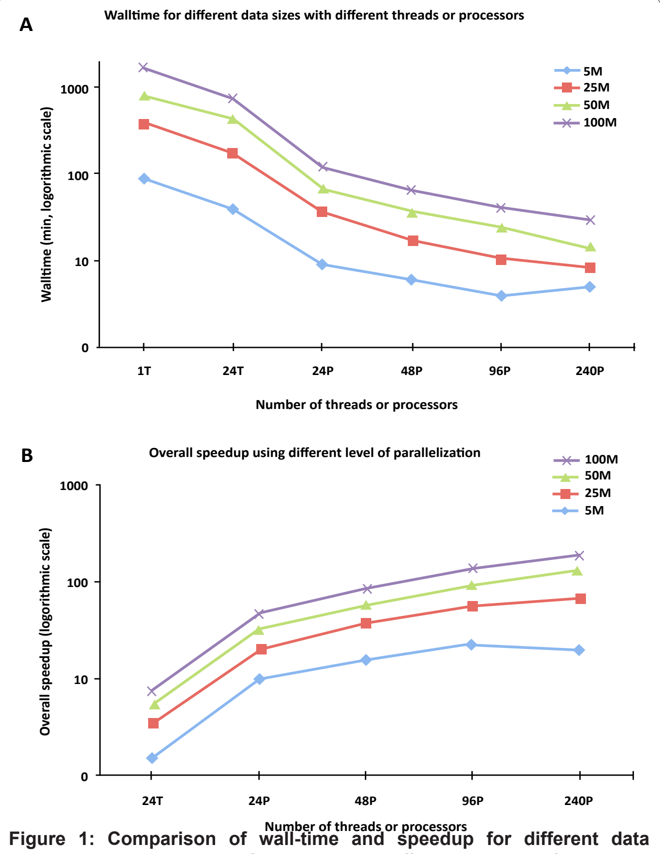

Figure 1: Comparison of wall-time and speedup for different data sizes. The comparison was for *pBWA* using different numbers of threads or processors on the orca system of SHARCNET Panel A: wall-time comparison. Panel B: speedup comparison. In both above cases, the comparison is based on the run with 1 processor at 1 thread. “24T” is for runs at 1 processor with 24 threads. All groups with “P” are runs using different number of processors (indicated by the number before “P”), each running at 1 thread.

size (from 5 million to ~350 million), furthering the appeal of *pBWA* for extremely large datasets.

Combination of parallelization and multi-threading

We also compared the efficiency of running pure parallelization with combined use of parallelization and multi-threading. For example, the 350 million reads dataset was run with 240 processors in pure parallelization (i.e. 1 threading per processor) and using 48 processors, each running 5 threads, therefore both for a total of 240 threads (the 240P and 48P/5T columns in Table 4). It is a bit of surprising for us to notice that the pure parallelization actually showed slightly better speedup than the combined option for the *aln* step. With this dataset, we also checked the speedup using the close to the maximal capacity of the orca cluster by using 240 of its processors each running 12 threads for a total of 2880 threads (Table 4). In this case, although at cost of lower hardware efficiency, a much better speedup was still obtained. Specifically, the task of aligning the 350 million paired reads (or 700 million total reads) would take 1 processor running 1 thread 15081 minutes (~10 days), and would take one processor running at its full capacity for multi-threading (24) 1509 minutes (~1 day), and it can be completed in 41 minutes if using less than half of the cluster’s capacity, i.e. using 240 processors each running 12 threads. From 10 days to less than 1 hr represents a very significant speedup, which makes analysis of extremely large datasets practically feasible by taking the advantage of high performance computing hardware.

Comparison of pBWA runs on different clusters

Comparisons between test runs on different clusters can be made from Tables 2 and 3. Each of these tables is for running *pBWA* for the same dataset and command parameters. It can be seen that while the *requin* cluster (1 thread data in Table 2) appears to execute more quickly with sequential *BWA* than the *orca* cluster (1 thread data in Table 3), the *orca* cluster is more efficient at executing *pBWA* based on the speedup and efficiency at equivalent number of threads or processors (48P on orca vs. 50P on requin and 96P on orca vs. 100P on requin in Tables 2 and 3). This can be attributed to the small amount of RAM possessed by each *requin* node. It suggests that each parallel cluster has its own optimal parameter set for *pBWA,* i.e. clusters with less RAM will benefit from combining multi-threading with parallelism, while clusters with more RAM will benefit from going purely parallel, as this maximizes the parallelism for the *samse/sampe* step.

Combining the results from variable numbers of processors and variable sequence sizes, there seems to be an optimal use of parallelization based on the amount of sequences. The speedup seems to stop and even deteriorate when the amount of sequences is below 50,000 per processor (Table 3). Furthermore, at very large amounts of sequences, parallelization provides slightly better speedup and efficiency for the *aln* step than multi-threading for the same total number of threads and much better overall speedup and efficiency, which is due to lack of multi-threading for *sampe* (240P vs. 48P@5T in Table 4). Nevertheless, with an extra large amount of sequences for a cluster with a relatively small number of multi-core nodes that do not have sufficient RAM for running a number of processors at a number equal to that of cores on the node, it would certainly help to achieve a better speedup by combining parallelization with multi-threading to maximize the use of all available cores. Overall, significant improvement in speed and efficiency can be obtained using parallelization once the amount of sequences is over 1 million, and the larger the sequence amounts, the better the improvement.

Conclusion and Future Development

We have developed *pBWA,* an efficient parallel implementation of *BWA*, based on the Open MPI library. This presents the first fully BWT-based parallelized open source short sequence alignment tool as of this writing. *pBWA* shows excellent results with speedup to be bounded only by the size of the parallel system, and it can be run on clusters of all shapes and sizes due to the ability to combine multi-threading and parallelization. The ability of processing both FASTQ input files for paired datasets at once at the *aln* step and the practice of merging multiple FASTQ files for the same sample provides further improvement in efficiency. With even just a moderate level of performance computing cluster, the use of *pBWA* can cut down the computing wall-time for the reference alignment step from weeks to hours for extremely large DNA sequence datasets, which is becoming a norm due to the ever increasing capacity of the NGS technologies. Therefore, the availability of *pBWA* should facilitate the analysis of large-scale next generation DNA sequencing data, such as personal genomes. The source code and detailed user manual are freely available at http://sourceforge.net/projects/pbwa with the intent to merge with the *bwa* source forge project in the future for long-term maintenance.

doi:http://dx.doi.org/10.4172/jabcb.1000101

We are aware of the release of newer versions of *BWA* (latest version is 0.6.2), which seems to involve significant changes including how the indexing of the reference genome is done. While the newer generation of *BWA* awaits to be fully tested by users for its stability and performance, we are currently working on a parallel version for the latest version of *BWA*, and it will be posted on the same source forge project site once completed.

Acknowledgements

This work is in part supported by grants from the Canada Research Chair program and Canadian Foundation of Innovation (to PL), and from Natural Sciences and Engineering Research Council (NSERC) (to PL and KQ), and was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET: www.sharcnet.ca) and Compute/Calcul Canada. Special thanks to Robert Egan for modifications made to *pBWA* that improve stability and provide the single output file option.

References

1. 1000 Genomes Project Consortium (2010) A map of human genome variation from population-scale sequencing. Nature 467: 1061-1073.
2. Barøy T, Misceo D, Frengen E (2008) [Structural variation in the human genome contributes to variation of traits]. Tidsskr Nor Laegeforen 128: 1951-1955.
3. Abecasis G, Tam PK, Bustamante CD, Ostrander EA, Scherer SW, et al. (2007) Human genome variation 2006: Emerging views on structural variation and large-scale SNP analysis. Nat Genet 39: 153-155.
4. Mardis ER (2008) The impact of next-generation sequencing technology on genetics. Trends Genet 24: 133-141.
5. Yang MQ, Athey BD, Arabnia HR, Sung AH, Liu Q, et al. (2009) High-throughput next-generation sequencing technologies foster new cutting-edge computing techniques in bioinformatics. BMC Genomics 10: I1.
6. Pospisilova S, Tichy B, Mayer J (2009) [Human genome sequencing--next generation technology or will the routine sequencing of human genome be possible?]. Cas Lek Cesk 148: 296-302.
7. Metzker ML (2010) Sequencing technologies - the next generation. Nat Rev Genet 11: 31-46.
8. Wheeler DA, Srinivasan M, Egholm M, Shen Y, Chen L, et al. (2008) The complete genome of an individual by massively parallel DNA sequencing. Nature 452: 872-876.
9. Li H, Ruan J, Durbin R (2008) Mapping short DNA sequencing reads and calling variants using mapping quality scores. Genome Res 18: 1851-1858.
10. Li H, Durbin R (2009) Fast and accurate short read alignment with burrows-wheeler transform. Bioinformatics 25: 1754-1760.
11. Li R, Li Y, Kristiansen K, Wang J (2008) SOAP: Short oligonucleotide alignment program. Bioinformatics 24: 713-714.
12. Langmead B, Trapnell C, Pop M, Salzberg SL (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biol 10: R25.
13. Alkan C, Kidd JM, Marques-Bonet T, Aksay G, Antonacci F, et al. (2009) Personalized copy number and segmental duplication maps using next-generation sequencing. Nat Genet 41: 1061-1067.
14. Hach F, Hormozdiari F, Alkan C, Hormozdiari F, Birol I, et al. (2010) mrsFAST: A cache-oblivious algorithm for short-read mapping. Nat Methods 7: 576-577.
15. Ruffalo M, LaFramboise T, Koyuturk M (2011) Comparative analysis of algorithms for next-generation sequencing read alignment. Bioinformatics 27: 2790-2796.
16. Li H, Homer N (2010) A survey of sequence alignment algorithms for next-generation sequencing. Brief Bioinform 11: 473-483.
17. DeFrancesco L (2012) Life technologies promises $1,000 genome. Nat Biotechnol 30: 126.
18. Korlach J, Bjornson KP, Chaudhuri BP, Cicero RL, Flusberg BA, et al. (2009) Real-time DNA sequencing from single polymerase molecules. Science 323: 133-138.
19. Harris TD, Buzby PR, Babcock H, Beer E, Bowers J, et al. (2008) Single-molecule DNA sequencing of a viral genome. Science 320: 106-109.
20. Ruffalo M, LaFramboise T, Koyuturk M (2011) Comparative analysis of algorithms for next-generation sequencing read alignment. Bioinformatics 27: 2790-2796.
21. Lam HY, Pan C, Clark MJ, Lacroute P, Chen R, et al. (2012) Detecting and annotating genetic variations using the HugeSeq pipeline. Nat Biotechnol 30: 226-229.
22. Burrows M, Wheeler DJ (1994) A block-sorting lossless data compression algorithm. Digital Equipment Corporation, Palo Alto, CA.
23. Graham RL, Shipman GM, Barrett BW, Castain RH, Bosilca G, et al. (2006) Open MPI: A High-Performance, Heterogeneous MPI. IEEE International Conference on Cluster Computing.
24. Wilkinson B, Allen M (2005) Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers. (2ndedn), Pretence Hall, Upper Saddle River, New Jersey, USA.
25. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, et al. (2009) The sequence alignment/map format and SAMtools. Bioinformatics 25: 2078-2079.
26. Sahni S, Vairaktarakis G (1996) The master-slave paradigm in parallel computer and industrial settings. J Glob Optimizat 9: 357-377.

Author Affiliations

Top

[1]Department of Computer Science, Brock University, St. Catharines, Ontario, Canada
[2]Department of Biological Sciences, Brock University, St. Catharines, Ontario, Canada