Highly efficient parallel approach to the next-generation DNA sequencing

Jacek Blazewicz[1,2], Bartosz Bosak[3], Piotr Gawron[1], Marta Kasprzak[1,2], Krzysztof Kurowski[3], Tomasz Piontek[3], and Aleksandra Swiercz[1,2⋆]

[1] Institute of Computing Science, Poznan University of Technology, Poznan, Poland
[2] Institute of Bioorganic Chemistry, Polish Academy of Sciences, Poznan, Poland
[3] Poznan Supercomputing and Networking Center, Poznan, Poland

Abstract. Due to the rapid development of the technology, next-generation sequencers can produce huge amount of short DNA fragments covering a genomic sequence of an organism in short time. There is a need for the time-efficient algorithms which could assembly these fragments together and reconstruct the examined DNA sequence. Previously proposed algorithm for de novo assembly, SR-ASM, produced results of high quality, but required a lot of time for computations. The proposed hybrid parallel programming strategy allows one to use the two-level hierarchy: computations in threads (on a single node with many cores) and computations on different nodes in a cluster. The tests carried out on real data of *Prochloroccocus marinus* coming from Roche sequencer showed, that the algorithm was speeded up 20 times in comparison to the sequential approach with the maintenance of the high accuracy and beating results of other algorithms.

Keywords: hybrid programming paradigm, *de novo* assembly

1 Introduction

A molecule of *deoxyribonucleic acid* (*DNA*) can be found in every cell of a living organism. It carries genetic information necessary for its functioning. Despite the fact that DNA is similar for individuals from the same species, each individual has a unique DNA sequence. These differences in DNA cause different appearance of each individual but may also cause susceptibility to some diseases. Thanks to the knowledge of specific differences in DNA, a proper therapy can be applied to an ill patient. Thus, its proper decoding has a crucial meaning in all the aspects of the health issues.

DNA sequence is in the form of a double helix, where two strands are closely connected by hydrogen bonds according to specific rules. Each strand is composed of small molecules, called nucleotides, which differ in nitrogenous bases. Four different bases can be distinguished: adenine, cytosine, guanine and thymine, and are abbreviated respectively as A, C, G, T. Reading a DNA sequence means reading the order of nucleotides (letters A, C, G, and T). It is

⋆ corresponding author: aswiercz@cs.put.poznan.pl

sufficient to read only one strand, because the other one is complementary to it, which means that if there is 'A' in one strand, then there is 'T' at the same position in the other strand. Similarly, 'C' is always opposite to 'G'. The DNA assembly is a part of the process of reading a DNA sequence. In the DNA assembly problem short fragments of DNA (of length up to a few hundreds of nucleotides) are combined together in order to construct a longer sequence, e.g. a sequence of a whole chromosome of length ca 108 nucleotides. The set of short DNA fragments is the output of the preceding process: DNA sequencing. Originally, in DNA sequencing phase gel-based methods were used [15, 16], which produced long fragments but with a lot of effort and time. A few years ago several techniques of *next-generation sequencing* were developed, which read huge amount of DNA fragments in parallel. Among next-generation sequencing systems the most known are *Roche* (454) *sequencer* [14], *Illumina sequencer* [1] and *SOLiD Applied Biosystems sequencer* [8]. Huge amounts of data force the implementation of time-efficient algorithms which optimize the memory usage.

DNA assembly problem is strongly NP-hard, because even its simplified version, shortest common superstring problem is strongly NP-hard [9](cf. also [10]). In DNA assembly problem the aim is to reconstruct a sequence from shorter DNA fragments. Fragments may contain errors like insertions, deletions or mismatches, and may come from both strands of a DNA helix.

Several heuristic algorithms were developed, both for assembling DNA fragments coming from gel-based methods, e.g. [11, 13, 18], and for shorter fragments obtained with next-generation sequencers, e.g. [12, 19]. Some methods are specialized for the special type of the sequencer, e.g. *Newbler assembler* is designed only for Roche sequencer [14]. The method developed previously by the authors of the current paper [2] proved to give as its output long reconstructed sequences of high quality, outperforming other available methods. The crucial point of the algorithm was long computation time, even in the case of parallel version [5].

The aim of the current paper was to develop an algorithm, which will maintain high accuracy of the previous approach but being much faster. The proposed hybrid strategy allows to use the two level hierarchy: computations in threads (on a single node with many cores) and computations on different nodes in a cluster. The applied parallel mechanism reduces computational time giving at the same time a high quality of the results.

A construction of the paper is as follows. Section 2 describes the algorithm for DNA assembly while Section 3 – the parallel version of the algorithm. In Section 4 results of the computational experiment are presented. The last section concludes the paper.

2 Short Reads ASseMbly (SR-ASM) algorithm

SR-ASM algorithm [2] is based on operations on an overlap graph. Here, we modify slightly a concept of DNA graphs introduced in [4]. The vertices of the overlap graph correspond to DNA fragments which are connected by arcs if there exists a feasible overlap between vertices. Next, in this graph a path(s) is searched

for, which passes a maximum number of vertices. At the end, the sequence(s) is reconstructed from the path(s). Each step of the algorithm is described in details below.

Construction of the overlap graph. Every DNA fragment corresponds to one vertex in the graph. Additionally, for each vertex its complementary counterpart is created, which corresponds to the fragment coming from the other strand of the DNA helix, which is reverse and complementary to the original fragment. Next, the overlaps between pairs of fragments are calculated with the Smith-Waterman (SW) algorithm [17]. The algorithm aligns two fragments returning an error rate (the number of mismatches in the alignment) and a shift between fragments. The SW algorithm takes $O(kl)$ time, where k and l are the lengths of the fragments. The number of alignments, which need to be calculated, is $O(n^2)$, where n is the number of fragments. The most preferable would be to determine the alignment between each pair of the fragments (every one taken twice, as the original and the reverse and complementary version), but then the number of fragments goes up to a million. Most of the fragments do not overlap with each other in a satisfying manner, thus, determining the alignment is not necessary. The algorithm selects in a fast and intelligent way the pairs of fragments which possibly give high score of the alignment. In our algorithm, a heuristic was introduced, which searches for common substrings in pairs of fragments. Based on this, it selects pairs of promising fragments, i.e. the fragments which would overlap with high probability.

For each pair of the promising fragments the alignment is determined according to the SW algorithm. The score of the alignment gives an information about the number of errors (mismatches, insertions and deletions), but also about the shift between two fragments. If the shift and the error rate for two fragments is feasible, an arc is added to the graph which connects corresponding vertices.

The construction of the graph is composed of many independent operations, very suitable for distributing among parallel processes.

Searching for the path in the graph. In the overlap graph a path is searched for, which contains most of the vertices. (Here, the approach is very similar to the one constructing a DNA sequence in the SBH approach [3, 6]) Optimally, it will be a Hamiltonian path. Unfortunately, due to errors present in the instance it is usually not possible. Another reason is that the heuristic used for construction of the overlap graph, selects just few pairs of fragments for further comparison, and many connections between vertices are not considered at all. In that case, the graph is quite sparse and might not be connected. Following this fact, many shorter paths are returned. Some vertices may be not used in any of the path, because they may represent contaminated fragments.

As the output of this step we obtain for each path an ordered set of vertices, for which respective fragments are tightly overlapping.

Printing the consensus sequence. In this step, the consensus sequences are constructed from the paths. Consecutive fragments (vertices) from each path are aligned together. The fragments which cover a certain position (letter) in the consensus sequence vote and the majority settle the proper letter (A, C, G or T). The consensus sequence of a path is called a *contig* (contiguous consensus sequence)

Again the problem of multiple alignment of the sequences is NP-hard, thus heuristic algorithm was applied here.

3 Parallelization of the algorithm

The parallelism can be introduced to the aforementioned algorithm, especially in the first step. However, one should remember that the process of building the graph is composed of a few parts. Some parts cannot be parallelized at all and some to a satisfactory extent. Although the comparison of fragments with each other was efficiently performed by different threads (on different nodes), the stage of selecting fragments to be compared was implemented sequentially, due to high demand on the memory and relatively short computation time.

The second part of the algorithm (searching for paths in the graph) was also parallelized. This part is more complex and CPU-time consuming than the previous one. Searching for paths in a graph is, in general, very difficult to perform in a parallel way. Nevertheless, our method has partially solved this problem by running the same algorithm (on many nodes, in many threads) starting from different initial points. In each iteration every fragment is taken as the initial solution. The longest path is chosen and the arcs from the path are deleted from the graph.

This approach has two major disadvantages. First, each thread on every node requires full information about the graph, which dramatically increases requirements of memory and limits size of problem instances that can be processed. The second problem involves the implementation of searching method. Our solution offered the possibility of modifying the structure of the graph by adding some arcs during the phase of finding paths. If the algorithm finds out that there is no way to continue the process of extending the path from the last vertex, but there are some premises to continue the path, coming from the preceding vertices, additional alignments are calculated and if feasible – respective arcs are added. Possible extensions to the overlap graph are presented in Figure 1.

Adding arcs to the graph implies possibility of desynchronizing information about the graph among different processes and could be the reason of receiving different results on different computing environments (various number of processes or threads, different computer architectures). The results obtained from the desynchronized data may be of lower quality. To avoid these situations, additional functionality was implemented - the *synchronization*. After every iteration of the algorithm, information about modifications in the graph is gathered from all processes and, if there are any differences, the graphs from each node are synchronized. In such a case, the iteration is calculated from the beginning.

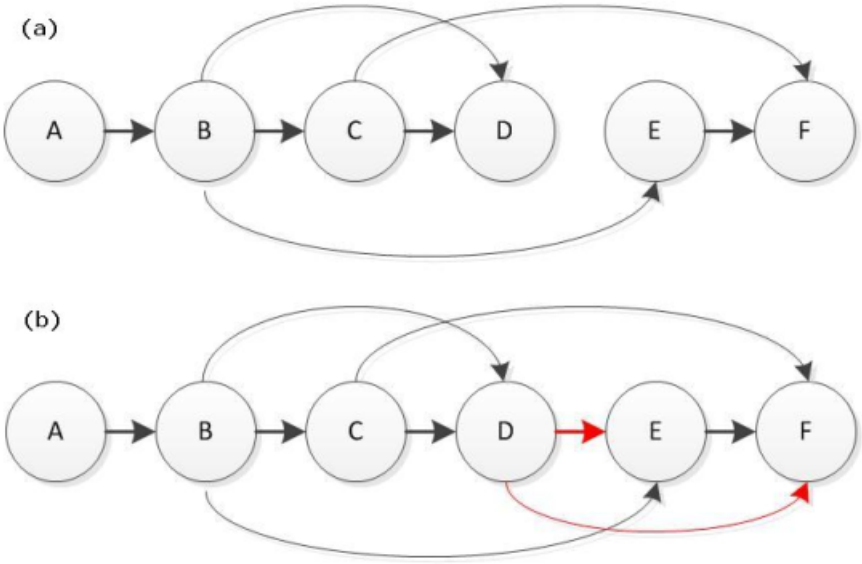

Fig. 1. An example of adding arcs in the overlap graph in the phase of searching for paths. (a) In the graph a path was found (A,B,C,D). The path cannot be extended because there is no arc outgoing from D, but some arcs from the predecessors of D were detected. (b) Scores of alignments between D and E, and D and F are calculated, and if they are feasible, the respective arcs are added to the graph. These arcs were not inserted in the first phase of the algorithm because the fragments (vertices) were not selected as 'promising pairs' (due to errors in fragments).

Synchronization ensures that the results obtained are the same as in the sequential version of the algorithm. At the same time, the synchronization leads to overhead time (single iteration may be computed a few times) and the user can decide to turn it off.

The third part of the algorithm (printing the consensus sequence) takes relatively little time and no attempt was made to parallelize it.

The implementation of parallelism in the program is based on a hybrid strategy introducing two level hierarchy: the communication between nodes in one cluster is realized by MPI library implementing the message passing paradigm, while the computations on a single node with many cores are parallelized utilizing POSIX threads. When the program is run on a single node, the first thread becomes a dedicated supervisor responsible for synchronization and distribution of the data as well as for doing sequential computations. The other available threads become worker units executing parallel computations. Similarly, in the case of using MPI, the first MPI process is designed to be a master, while the other MPI processes (slaves) do parallel computations.

In a typical scenario of running the SR-ASM program on a cluster, the process being an MPI master divides the tasks and distributes them among other MPI processes located on different nodes. Next, the MPI processes (dedicated supervisor threads) being a node-level coordinators, receive the data, once again divide the tasks and distribute them among the worker threads. After computations, the worker threads send results back to supervisor threads on theirs nodes and finally these threads response to the MPI master that gathers all the results. The MPI communication is here both synchronous and asynchronous. The MPI master communicates with supervisor threads (MPI slaves) fully synchronously. It sends tasks and waits for results. When the results are obtained from one of the MPI slave processes, the master stores the data, generates next set of tasks and sends it to this process for further computations. On the other hand, on a single computing node level, worker threads communicate with the supervisor thread in an asynchronous way. Thanks to this approach, the computations and communication may be done simultaneously in order to improve efficiency of the whole process. Moreover, the implemented communication scheme utilizes queues to ensure that the next task to compute will be immediately available for processing after finishing the currently executed one. The performed tests confirm that all these mechanisms significantly improve an overall program scalability and performance.

4 Computational tests

The scalability and performance tests were done on the Zeus cluster being a part of the Polish National Grid Initiative - PL-Grid Project. The cluster consists of several types of Intel-based nodes, which offer total computational power about 105 TF. The results presented in the paper are the outcome of running SR-ASM program on nodes with the following parameters:

Processor: Intel Xeon L5650 2666MHz (2 x 6 cores)

Memory: 24GB per node
Network Interface: Ethernet 1Gb/s
Operating System: Scientific Linux 5

The test bed used in the computational experiment consists of raw data produced at Joint Genome Institute. The data cover the whole genome of Prochlorococcus marinus bacteria of length 1.84 nucleotides [7]. The output of the sequencer contains over 300 000 DNA fragments, each approximately 100 nucleotides long. The sequencer also provides rates of confidence for every nucleotide.

As compared with its standard sequential version [2] the parallel SR-ASM improves the speed of the computations considerably. In fact, while the sequential version solved the analyzed instance of the problem in 255 min. (the first version of the algorithm published in [2] produced the results in 80 hrs., but since then it has been improved), its parallel version, even in the simplest case of the 1 node and 12 cores, needed 27 min only. What's more the parallel version is well scalable (to some extent).

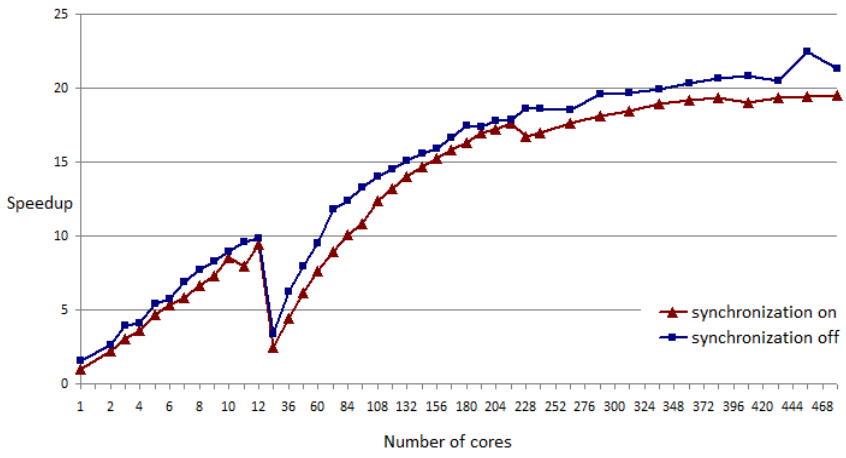

Fig. 2. Speedup of the parallel version of SR-ASM: version with synchronization turned on and synchronization turned off. Computations were performed on a computer cluster of 12-core nodes.

Speedup of the whole algorithm with enabled and disabled synchronization is shown in Figure 2. One can easily see from this chart that the speedup for 24 cores is much lower than for 12 cores. This dramatic fall off is a result of changing the type of parallelism. For the first 12 cores, only one machine was used to compute results, while bigger number of cores was achieved by running SR-ASM on a cluster consisting of many 12-core nodes, what resulted in a higher communication overhead. Speedup of the version with synchronization turned

algorithm (time[s])	1st contig			2nd contig			3rd contig		
	length	coverage	quality	length	coverage	quality	length	coverage	quality
SR-ASM,s_on (781)	105951	5.66%	99.71	81990	4.38%	99.76	59714	3.19%	99.68
SR-ASM,s_off (714)	82209	4.39%	99.27	73071	3.90%	98.95	60475	3.23%	88.32
at JGI (-)	86108	4.60%	96.85	74315	3.97%	96.94	73941	3.95%	97
NEWBLER (-)	84798	4.53%	99.29	73192	3.91%	99.38	72818	3.89%	99.45
PHRAP (2067)	88729	4.74%	94.56	35379	1.89%	81.66	35192	1.88%	98.56
CAP3 (12376)	7862	0.42%	99.85	7113	0.38%	99.69	5990	0.32%	99.91
VELVET (328)	3182	0.17%	99.9	2808	0.15%	99.96	2621	0.14%	99.9

Table 1. The results of the computational experiment performed on the dataset of short fragments coming from the experiment of sequencing bacteria *Prochlorococcus marinus*. The length of the genome is 2Mbp. The number of fragments in the dataset is ca. $3 * 10^5$

off varies between 30% (for the small number of cores) and 5% (for the big number of cores) as compared to the version with synchronization turned on. It is worth stressing that the parallel version of the algorithm with the synchronization turned on, maintains the high quality level for constructed sequences, already known for the sequential SR-ASM. As compared with other assembling algorithms: Newbler [14], Velvet [19], Phrap [11], Cap3 [12], the basic quality measures are favorable for SR-ASM. These measures include:

a) the *length* of the contig (for the three largest contigs),
b) the *coverage*, i.e. the percentage of the total length of the reconstructed sequence (genome), which is covered by the contig,
c) the *quality*, i.e. the similarity of a contig to a sequence of the genome; it was calculated by the Smith-Waterman algorithm.

The coverage and the similarity to a sequence of the reference genome can be calculated by comparing the obtained results with the reference genome. The reference genome of Prochlorococcus marinus is known and can be downloaded from Genbank (National Center for Biotechnology Information – NCBI).

The comparison of the results obtained for the considered instance are given in Table 1. Each algorithm is evaluated with the time of computations and the measures (mentioned above) for three largest contigs. Two versions of our algorithm are compared: with the synchronization turned on and off. Parallel SR-ASM with the synchronization on gives the same results in the quality terms as the sequential algorithm. When the synchronization is off, the algorithm does not update the information about new arcs in the graph (in the part of searching

for paths). This may lead to nondeterministic results. Time of computations is shorter than for SR-ASM with synchronization on, but the contigs are shorter, and often with lower quality. Thus, updating the information about the graph among the nodes seems to be very important. The results of other methods presented in Table 1 are obtained by assemblers from the Joint Genome Institute with an additional step of experts' finishing ('at JGI'), NEWBLER (assembler available with the Roche sequencer), PHRAP and CAP3 (well known publicly available assemblers), and VELVET (specialized algorithm for assembling short fragments). The executing time for NEWBLER and JGI were not available. All the methods except SR-ASM were tested sequentially. CAP3 and VELVET resulted in a great amount of short contigs but of very high quality. Thus, these methods are not likely to be used for de novo sequencing, but rather for resequencing, when the reference genome is known and good quality contigs are aligned to it to check for the difference between genomes of individuals. VELVET was the fastest method among all, while CAP3 the slowest. PHRAP obtained contigs shorter and of worse quality than the NEWBLER and JGI methods.

None of the algorithms found one contig covering the whole genome. This is usually the case, because the fragments are not uniformly distributed among the genome sequence. So, they result in disjoint contigs. Further order of the contigs can be determined e.g. on the basis of paired end fragments or by aligning contigs to a reference genome. If the reference genome is not known (this is the case of de novo sequencing) one may try to align contigs to a genome closely related to the examined organism.

5 Conclusions

The parallelization of SR-ASM algorithm satisfactorily speeded up the computations, especially in the case of multi-threading (almost linear speedup can be viewed for up to 12 cores in Fig. 2). The possible improvement of the algorithm could be realized on machines with more cores and shared memory to decrease the time needed for the communication and synchronization.

The results presented in Table 1 and in Figure 2 show, that although turning off synchronization may speed up the method (5-30% in comparison with the synchronization turned on), the results have worse quality. For a greater number of clusters the speedup goes down. This is a result of changing time distribution between the computation, communication and synchronization.

It can be seen, that the parallel version of SR-ASM maintained high quality of the results as for the sequential version, while the time was greatly speeded up. Thus, the proposed solution can be of practical interest for those who would like to sequence whole genomes, based on the massive data coming from next-generation sequencers.

Acknowledgements We are very grateful to the anonymous referees for their helpful comments on how to improve the paper.

This work is partially funded by NCN grant (DEC-2011/01/B/ST6/07021), Polish NGI, and PL-Grid Project (POIG.02.03.00-00-007/08-00)

References

1. Bennett,S.: Solexa Ltd. Pharmacogenomics 5, pp. 433–438 (2004)
2. Blazewicz, J., Figlerowicz, M., Gawron, P., Kasprzak, M., Kirton, E., Platt, D., Swiercz, A., Szajkowski, L.: Whole genome assembly from 454 sequencing output via modified DNA graph concept. Computat. Biol. Chem. 33, pp. 224–230 (2009)
3. Blazewicz, J., Formanowicz, P., Kasprzak, M., Markiewicz, W.T., Weglarz, J.: DNA sequencing with positive and negative errors. J. Comput. Biol. 6, pp. 113–123 (1999)
4. Blazewicz, J., Hertz, A., Kobler, D., de Werra, D.: On some properties of DNA graphs. Discrete Appl. Math. 98, pp. 1–19 (1999)
5. Blazewicz, J., Kasprzak, M., Swiercz, A., Figlerowicz, M., Gawron, P., Platt, D., Szajkowski, L.: Parallel implementation of the novel approach to genome assembly. In: Proceedings of SNPD 2008, R. Lee, P. Muenchaisri, W. Dosch (eds.) pp. 732–737. IEEE Computer Society, Los Alamitos (2008)
6. Blazewicz, J., Oguz, C., Swiercz, A., Weglarz, J: DNA sequencing by hybridization via genetic search. Oper. Res. 54, pp. 1185–1192 (2006)
7. Chen, F., Alessi, J., Kirton, E., Singan, V., Richardson, P.: Comparison of 454 sequencing platform with traditional Sanger sequencing: a case study with de novo sequencing of Prochlorococcus marinus NATL2A genome. In: Plant and Animal Genomes Conference. http://www.jgi.doe.gov/science/posters/chenPAG2006.pdf (2006)
8. Fu, Y., Peckham, H.E., McLaughlin, S.F., Rhodes, M.D., Malek, J.A., McKernan, K.J., Blanchard, A.P.: SOLiD sequencing and Z-Base encoding. In: The Biology of Genomes Meeting. Cold Spring Harbour Laboratory (2008)
9. Gallant, J., Maier, D., Storer J.: On finding minimal length superstrings. J. Comput. Sys. Sci. 20, pp. 50–58 (1980)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, San Francisco (1979)
11. Green, P.: Documentation for PHRAP. Genome Center, University of Washington, Seattle (1996)
12. Huang, X., Madan, A.: CAP3: a DNA sequence assembly program. Genome Res. 9, pp. 868–877 (1999)
13. Kececioglu, J.D., Myers, E.W.: Combinatorial algorithms for DNA sequence assembly. Algorithmica 13, pp. 7–51 (1995)
14. Margulies, M., Egholm, M., Altman, W.E., Attiya, S., Bader, J.S.: Genome sequencing in microfabricated high-density picolitre reactors. Nature 437, pp. 376–380 (2005)
15. Maxam, A.M., Gilbert, W.: A new method for sequencing DNA. Proc. Natl. Acad. Sci. USA 74, pp. 560–564 (1977)
16. Sanger, F., Nicklen, S., Coulson, A.R.: DNA sequencing with chain-terminating inhibitors. Proc. Natl. Acad. Sci. U.S.A. 74, pp. 5463–5467.(1977)
17. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. J. Mol. Biol. 147, pp. 195–197 (1981)
18. Waterman, M.S.: Introduction to Computational Biology. Maps, Sequences and Genomes, Chapman & Hall, London (1995)
19. Zerbino, D.R., Birney, E.: Velvet: algorithms for de novo short reads assembly using de Bruijn graphs. Genome Res. 8, pp. 821–829 (2008)