Interdiscip Sci Comput Life Sci (2011) 3: 57–63
DOI: 10.1007/s12539-011-0062-0

A Comparative Analysis of Parallel Computing Approaches for Genome Assembly

Munib AHMED[1]*, Ishfaq AHMAD[2], Samee Ullah KHAN[3]
[1](Department of Computer Science and Engineering, University of Texas at Arlington, TX 76019, USA)
[2](Department of Computer Science and Engineering, University of Texas at Arlington, TX 76019, USA)
[3](Department of Electrical and Computer Engineering, North Dakota State University, ND 58108, USA)

Received 23 January 2010 / Revised 8 October 2010 / Accepted 12 November 2010

Abstract: Over the last two decades, we have witnessed a tremendous growth of sequenced genomic data. However, the algorithms and computational power required to expeditiously process, classify, and analyze genomic data has lagged considerably. In bioinformatics, one of the most challenging and computationally intensive processes, which may take up to weeks of compute time, is the assembly of large size genomes. Several computationally feasible sequential assemblers have been devised and implemented to assist in the process. A few algorithms also have been parallelized to speed up the assembly process. However, very little has been done to thoroughly analyze such parallel algorithms using the specific metrics of parallel computing paradigm. It is essential to investigate parallel assembly algorithms to ascertain their scalability and efficiency. The genomic data varies considerably in size that ranges from a few thousand units of data to several billions. Moreover, the degree of repetition in the data also exhibits high variance from one set to another. Therefore, we must establish an association between the nature, size, and degree of repetition in the genomic data and the best parallel assembly algorithm. The paper includes a comparative analysis of some of the most widely used approaches to assemble genomes using the parallel computing paradigm.

Key words: sequencing, genome assembly, parallel computing, isoefficiency.

1 Introduction

The field of bioinformatics has seen phenomenal advances since the early 1950s when the structural composition of DNA was laid out by (Watson and Crick, 1953). Several hundred gigabytes of genomic data have been sequenced, deciphered, and stored. However, the limit imposed by the sequencing technology that allows reading only a few hundreds of DNA bases at a time, makes the full genome assembly process very difficult. The objective is to assemble the DNA structure from a set of millions of overlapping, repeating, and somewhat inaccurate fragments. These fragments are (usually) denoted by strings of characters from an alphabet set $\partial = \{\text{A, T, C, G}\}$ representing four bases Adenine, Thymine, Cytosine, and Guanine, respectively. In this paper, we will first go over the shotgun sequencing technique and identify the tasks and the data they produce before the data is fed to the assembly programs. We will then explain the need for employing parallel computing and the criteria used to compare different parallel algorithms. Following that, we will define a frame of reference that outlines a generic sequence of stages of the assembly process. Next, the two commonly used approaches to genome assembly, *Overlap Layout Consensus* and *Euler Superpath*, and their parallel implementations will be presented, which will be followed by their scalability analysis and discussion of an experiment using the Overlap Layout Consensus approach and its results. Finally, we will discuss the future trends in this area of research.

1.1 Shotgun sequencing

The whole genome shotgun sequencing (WGSS) is a process of breaking up a DNA molecule into smaller pieces so it can be read. The WGSS process typically results in millions fragments, consisting of 10^3 or fewer bases, referred to as "reads". These reads entail very little detail on how to join them back together to create the complete DNA blueprint of the specie being studied. The reads are usually obtained by embedding a piece of the DNA sequence called "clone" into a special host molecule, generally termed as a "vector", which can be replicated in the laboratory to produce multiple copies of the clone. The clone is then separated from the vector extracting reads preferably in opposite directions

*Corresponding author.
E-mail: munib.ahmed@mavs.uta.edu

noting the orientation and the distance between the two reads. This distance is called a "mate-pair distance" and provides an important piece of information used in the later stages of assembly. The extracted pieces are decoded in the laboratory using special processes, e.g. gel electrophoresis (Southern, 1975) (see Fig. 1). The entire process is all but perfect and yields many poor quality reads, further adding to the difficulty in assembling these fragments. There are special programs, e.g. Phred (Ewing *et al.*, 1998), which evaluate the reads and assign a quality score to each base to quantify its accuracy in that position. The end regions of reads are usually of relatively poor quality. From a computer science perspective, the process of sequencing a DNA involves various steps many of which can be performed with special multi-phase programs called "assemblers". A typical assembler: (a) reads a set of input reads along with the corresponding base quality scores, (b) detects the overlaps among reads, and (c) aligns the pairs with the most suitable overlap. The process is repeated until no more overlaps are possible, yielding a set of aligned sequences called "contigs". These contigs are then oriented and arranged using multiple sequence alignment techniques yielding a "consensus" sequence. Finally, the relative position of each element in the array is validated using two important pieces of information, *i.e.* orientation and the mate-pair distance recorded during extraction of the reads earlier in the process. We will discuss the process in more detail in the following sections.

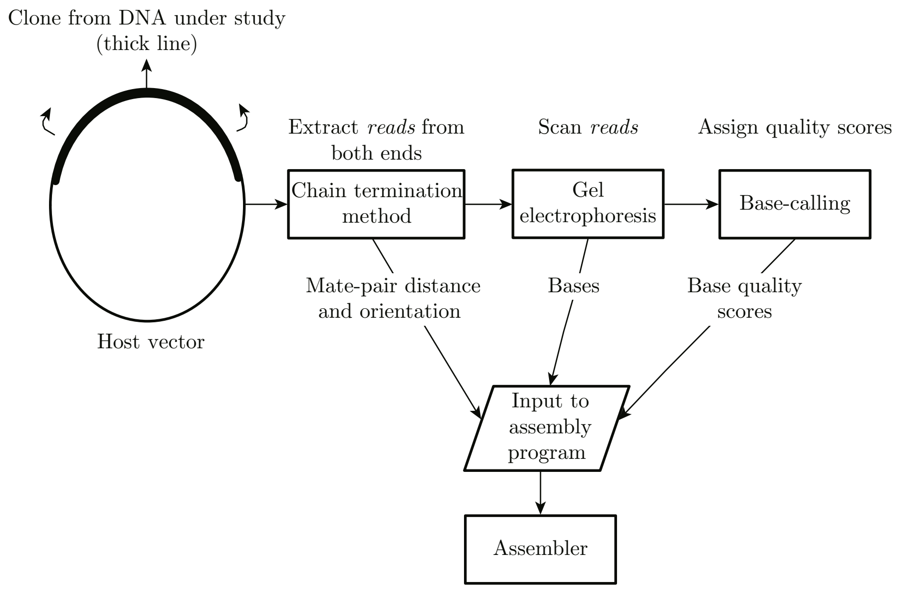

Fig. 1 Preprocessing that yields the input to the software based assembly

1.2 Role of parallel computing

To compensate for the possible reading errors, multiple copies of the DNA strands are sequenced at the same time to allow for multiple overlaps. This is called "coverage" or "redundancy factor", and is typically in a range of 5 to 12. The coverage or redundancy factor can be derived as $(N{*}l)/s$, where N, l, and s are the number of reads, average length of a read, and size of the DNA, respectively. Although the coverage improves the chances of getting better overlaps and hence more accurate results, it increases the size of the data by many-fold and renders the process computationally intensive. This makes the sequencing an excellent candidate for parallel or distributed computing where multiple processors can work on intelligently divided subsets of the data concurrently. Each processor may share and utilize dedicated resources, e.g. memory. The processors may at times communicate intermediate results among each other thereby reducing the overall processing time. As with any parallel program, the usual yardstick to measure the performance of the algorithm is "speedup". A speedup is defined as the ratio of sequential program execution time to the parallel program execution time. It should be noted that a parallel implementation (usually) incurs an overhead because of the communication among processors. A communication to computation ratio, denoted as C2C ratio, provides a measurable characteristic of a parallel program. Moreover, because the parallel program employs multiple processors, it is important to evaluate the utilization of the extra resources, such as processors and memory, to justify the cost associated with using such a high performance computing environment. A key measure providing such a cost benefit analysis is called "efficiency" and is calculated as the ratio of speedup to the number of processors used in the underlying parallel computation. Finally, a very critical factor for the success of a parallel algorithm is "scalability", i.e. the capability to exhibit a consistent performance when the data size grows provided that an increase in the number of processors is also permitted. In other words, there needs to be a balanced relationship between the number of available processors and the size of the data required to efficiently utilize the available resources.

1.2.1 Parameters of analysis and measurement

In this section, we discuss the metrics that will be used to compare the two parallel algorithms for genome assembly. As explained earlier, speedup, efficiency, and scalability are usually the three key metrics to measure the viability of a parallel algorithm. Grama *et al.* (1993) presented another metric termed "Isoefficiency" to relate the number of processors to the input data while maintaining efficiency, which allows measuring the scalability of an algorithm. That is, how much data should be increased to maintain efficiency within an acceptable range in response to an increased number of processors? The research provided some useful relations that we can utilize to perform our analysis. Firstly, the overhead time can be derived in terms of sequential and parallel times as:

$$T_0 = pT_p - T_1 \tag{1}$$

where T_p is the parallel time taken by p processors to process the input data and T_1 is the sequential time (using a single processor). Secondly, the input data, denoted by W is directly proportional to the overhead time T_0. That is, the overhead time increases with the number of processors because the input data must be increased proportionally to maintain the efficiency. Considering efficiency to be constant and denoted by e,

$$W = eT_0 \tag{2}$$

It can be observed that a smaller isoefficiency function means that the input data size can be increased in small increments for efficiently using a growing number of processors resulting in a highly scalable system whereas a larger isoefficiency function indicates a poorly scalable system which requires large increments of data for maintaining the same efficiency when more processors are added to the system.

1.3 Frame of reference

For the purpose of comparative analysis of parallel algorithms presented in this paper, we will assume the underlying sequencing process to be based on WGSS as explained before. The two most widely used assembly approaches are "Overlap-layout-consensus" (Batzoglou *et al.*, 2002, Huang and Madan, 1999) and "Euler Superpath" (Pevzner *et al.*, 2001).

The aforementioned approaches tend to reduce the problem into graphs such that a consensus sequence can be inferred by traversing either all of the nodes or all of the edges exactly once, resulting in a Hamiltonian or an Euler path, respectively. The subsequent text provides an in-depth discussion of the two approaches and their parallel implementations. It should be noted that a few other approaches, e.g. Genetic algorithms (Zhao *et al.*, 2008), have also been devised but we will leave those out of this discussion. Moreover, we may also use the terms "approach" and "algorithm" interchangeably. The laboratory procedures are outside of the scope of this study.

It is important to understand that even though the objective of each of the algorithms (under study) is the same, the logical flow and the number of stages in each of the algorithms are quite different. Therefore, it is not a straightforward one-to-one comparison of common functionalities. As a result, before we can compare two such approaches with respect to their parallel implementations, it seems logical to lay out a frame of reference using common standards and comparison criteria.

There are multiple variations of both approaches. However, for the purpose of our analysis, we will describe a more generic version that has all of the ingredients of the corresponding approach. We assume a common data set used as input with a total of n input reads, each of an average length of l, with a base coverage of c. For simplicity, we will assume that the reverse complements of the reads also have been added to the mix and are included in n.

Almost all of the assemblers assume that the reads have been extracted and the corresponding base quality scores, mostly optional, have been calculated as prerequisites. Following is a list of high level tasks that are generally performed by an assembler. Fig. 2 below illustrates the process flow.

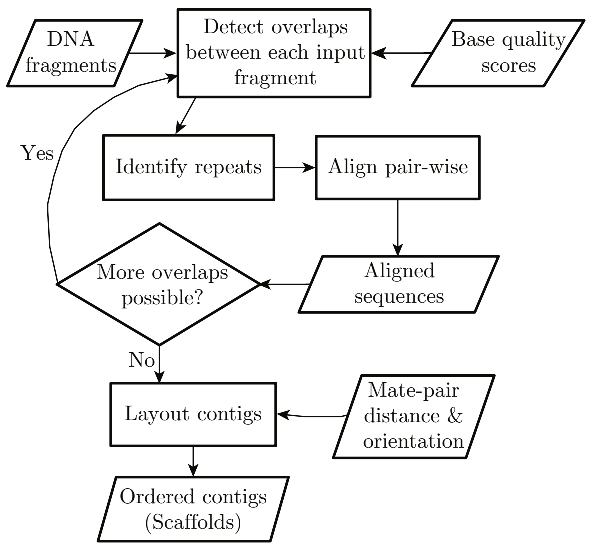

Fig. 2 A generic process flow of assembly tasks

1. Iterate through the input reads and corresponding quality scores.

2. Detect overlaps by comparing each read r_i with $r_j \forall i,\ j$: $0 < i < n,\ i < j \leqslant n$. A comparison is two way, *i.e.* suffix of r_i is compared with prefix of r_j and also suffix of r_j is compared with prefix of r_i.
3. Identify repeats consisting of either full or partial reads.
4. Align the most promising overlapping pairs resulting in aligned sequences.
5. Reiterate the above steps until no more overlaps are feasible, producing a set of long and contiguous segments called contigs that are disconnected from each other.
6. Using the orientation and mate-pair distance information lay out the contigs in order.

In the subsequent text, we will discuss each approach with both sequential and parallel version of the corresponding implementation.

2 Overlap-layout-consensus

This is the most commonly used approach and a large number of assemblers have been implemented based on one or more variations of the overlap-layout-consensus. This approach, as evident from the name, comprises of three high level stages. In the first stage, repeated regions and overlaps among the reads are detected. The repeats are either filtered out or masked by replacing each base value within repeat region with a special character using special purpose programs, such as RepeatMasker (Smit *et al.*, 1996). The process is reiterated until no further overlaps are detected. The outcome of this process is a set of contiguous segments, contigs that are finally arranged using the orientation and mate-pair distance information obtained during the earlier stages of genomic assembly.

We will now discuss the key tasks in both sequential and parallel implementations of this approach with a focus on deriving time complexity that will be utilized in measuring the isoefficiency. A few parallel versions of genomic sequencers have been published, e.g. (Huang *et al.*, 2003), but we will look at a more generic implementation. Following are the various stages of such a generic algorithm employing graph theory.

2.1 Finding overlaps

Each input sequence S_i is broken into a set K_i of smaller subsequences of length k, called k-mers, such that $K_i = \{S_i[x][x+k-1]\}\forall x,\ 1 \leqslant x < |S_i|-k+1$. All k-mers along with their origin information are stored in a sorted array. An overlap between a suffix of one and a prefix of another sequence is noted and can be readily observed as all such overlapping k-mers can be found contiguous in the sorted array. Unique overlaps are further verified using a local alignment algorithm (Smith and Waterman, 1981) or one of its several variants. For n sequences yielding m k-mers per sequence, a sequential algorithm to generate and sort k-mers and then align promising sequences takes asymptotically $\mathrm{O}(nm + nm\log(nm) + \alpha n)$ time, where α is a product of genome coverage and the square of average length of a sequence to account for potential alignments for each of n sequences. A parallel implementation on p processors of the same must asymptotically take $\mathrm{O}(pT_s + nT_t + nm/p + (nm/p)\log(nm/p) + \alpha n/p)$, where T_s is the start time and T_t is the transfer time of a sequence between two processors. It must be noted that T_s and T_t are infinitesimally smaller compared to the actual processing times.

2.2 Building layout

In most cases the layout is a directed graph G with sequences laid out as vertices and the overlaps between two sequences as the edges. Once a directed graph has been constructed, the redundant edges are removed using transitive property. That is, an edge $u \to v$ between u and v is removed if two other edges $u \to z$ and $z \to v$ are found and the sum of non-overlapping prefix lengths of u and z in $u \to z$ and $z \to v$, respectively, is equal to the non-overlapping prefix of u in $u \to v$. The reduced graph is to be traversed to find a Hamiltonian path, which is an NP-Complete problem. However, in practice, it is unlikely to find such a path from the perspective of both data and computation but the effort using heuristic algorithms results in subpaths that are recorded as contigs. In terms of time complexity, assuming a heuristic algorithm is employed, a scan of nm entries constructs a basic graph and then removing the redundant arcs can be done in asymptotically $\mathrm{O}(n^2m^2)$ time whereas a parallel implementation can achieve the same in asymptotically $\mathrm{O}(n^2m^2/p + \log p)$ time, when ignoring latency during merge operations among processors.

2.3 Finding consensus

Using the contigs produced above, a multiple alignment is performed to derive a consensus sequence. Recently, algorithms have been presented that would require asymptotically $\mathrm{O}(nm\log n)$ time to align n sequences assuming $m \ll n$ (Edgar, 2004). However, a parallel version would be able to achieve the same in asymptotically $\mathrm{O}(nm\log n)/p$ time.

3 Euler path method

The Euler path based approach was originally proposed in (Pevzner *et al.*, 2001). This approach reduces the genome assembly problem into an Eulerian path problem that lends itself to a polynomial computational complexity compared to the NP-completeness of the Hamiltonian path seen in the overlap-layout-consensus approach. The central idea of this algorithm is based on sequencing by hybridization where smaller reads are obtained leveraging a faster and cheaper sequencing process. It must be noted that the same concept could also be applied to the relatively larger sized reads obtained through Sanger sequencing (Sanger *et al.*, 1977). For example, a k-mer can be split into l-mers such that suffix of l_i overlaps with the prefix of l_{i+1} for a length of $(l-1)\forall i$, l: $0 < i \leqslant k-l+1$, $l < k$. Using all such l-mers, a directed deBruijn graph (Ralston, 1982) is constructed with suffix and prefix as vertices and the l-mer as the edge connecting suffix to the prefix. Ideally, a path traversing all such edges, called Eulerian path, would include all l-mers yielding a fully reconstructed genome. However, practically, a set of paths is obtained representing multiple contigs due to the presence of repeats and inaccurate data. The algorithm also attempts to identify and remove repeats by finding directed paths where the indegree of the starting vertex and the outdegree of the ending vertex are both greater than 1 whereas all intermediate vertices have both indegree and outdegree equal to 1. Using the relation and position of all of the l-mers in such path, within the original reads, the algorithm distinguishes the repeats from overlaps.

A parallel implementation of the algorithm has been reported in (Shi and Zhou, 2005). We will now discuss the key tasks in both sequential and a generic version of parallel implementation of this approach with a focus on deriving time complexity that will be utilized in measuring the isoefficiency. Following are various stages of such a generic algorithm employing graph theory based on ideas presented by (Pevzner *et al.*, 2005).

3.1 Generating *k*-mers

Considering each k-mer generation as a single operation, the time taken to generate all k-mers is proportional to the product of n and m. The parallel implementation will asymptotically take $O(nm/p)$ time.

3.2 Distributing *k*-mers

This is an extra step in the parallel implementation. However, using a hashing scheme, the procedure takes near linear time, *i.e.* $O(nm)$. In the parallel version, each processor keeps some k-mers to itself and distributes the others based on the hash value. This may result in some communication overhead but the time complexity in this stage is asymptotically equal to $O(nm/p)$.

3.3 Preparing data

The algorithm requires the multiplicities of k-mers to be computed for later use as a boundary condition when traversing paths. In the sequential algorithm, which is achieved by using suffix arrays to store and use k-mers, it takes $O(nm\log(nm))$ time. In parallel environment, the multiplicity of each k-mers can be calculated as a side step when hashing and distributing the k-mers incurring insignificant time. If the coverage c is not available, then it takes asymptotically $O(nm)$ to compute the coverage. This can be achieved using some fast string matching algorithms, such as the Gusfield's Z-algorithm presented in (Gusfield, 1997).

3.4 Building deBruijn graph

This task requires the bulk of computing power and time. Hashing may help find the adjacent k-mers faster; however, quadratic time is required to compare all k-mers with each other. A slight improvement would be to use the fact that each one of the n sequences was broken into a total of m k-mers. Therefore, those nm edges could be constructed without even comparing with each other. That results in $nm(nm\text{-}1)$ or asymptotically $O(n^2m^2)$ runtime complexity.

3.5 Traversing the Euler path

An Euler path can be identified in $O(E)$, where $nm < E < n^2m^2$. A parallelized version should take $n^2m^2/p + p\mathrm{T}_s + n\mathrm{T}_t$, i.e. asymptotically $O(n^2m^2/p)$ time. The Euler traversal of the graph provides the solution, i.e. sub-paths that can be recorded as contigs. The remaining work that includes scaffolding and finishing is generic in nature, which mostly is performed sequentially. Therefore, we consider that out of the scope of this paper.

4 Analyzing scalability

We will use the time complexities we noted above along with other criteria for evaluating the scalability of these approaches in the following sections.

4.1 Isoefficiency comparison

An algorithm is usually characterized by its time complexity that represents a relationship between runtime and the input data. In case of a parallel algorithm, the speedup, a runtime comparison of the execution speed of the parallel algorithm to its sequential version, is normally used as a metric for measuring performance. The efficiency, which is the speedup per processor, is a gauge of utilization that characterizes a parallel algorithm. Assuming a constant data size W, the efficiency decreases with the increasing number of processors. This is due to the fact that more processors are available to do the same work that previously was achieved by fewer processors resulting in lower utilization. It also results in more communication overhead and start-up times, denoted together by T_0 as correlated with input data in equation 2 above. A parallel algorithm is considered scalable if increasing resources, processors in most cases, requires only a proportional increase in the input data to keep the efficiency from decreasing significantly.

The sequential time complexities for different phases of overlap-layout-consensus approach can be summed up at a high level as:

$$T_1 = nm + nm\log(nm) + \alpha n + n^2m^2 + nm\log n$$

Whereas a parallel version would result in:

$$T_p = pT_s + nT_t + nm/p + (nm/p)\log(nm/p) +$$

$$\alpha n/p + n^2m^2/p + \log p + (nm \log n)/p$$

Using equations (1) and (2), assuming $p \gg 1$, and ignoring any lower asymptotic factors, we simplify the above as:

$$W = e(p^2T_s + pnT_t + p\log p) \qquad (3)$$

A similar workout for the Euler path approach would yield:

$$W = e(p^2T_s + pnT_t + nmp) \qquad (4)$$

It should be noted that all factors of the T_0 side of the equation contribute to the isoefficiency computation. However, we are more interested in the component that causes the data size to grow at the fastest rate with respect to p (Grama *et al.*, 1993) and therefore it follows that, asymptotically, both equations (3) and (4) have a quadratic isoefficiency in terms of the number of processors. Hence, we can deduce that an increase in the number of processors from p to p_{new} would require input data to be increased by a factor of p^2_{new}/p^2. It can be observed that these parallel implementations help decrease execution time and are reasonably scalable considering a small to mid-range parallel computing environment.

5 Experiment and results

To test and validate the time complexity analysis discussed above, we ran some tests using a modified version of a parallel implementation based upon the Overlap layout consensus approach using C programming language in MPICH environment (Gropp *et al.*, 1996), an implementation of Message Passing Interface (MPI) framework, using up to 32 nodes in a cluster environment. The cluster runs RH-Linux with an MPICH version 1.2 with a 2GB memory per node. The raw traces of *Drosophila Yakuba* were obtained from ENSEMBL (Hubbard *et al.*, 2007) along with the corresponding quality scores. A multi-phase program was run and total runtime was used to compute efficiency at different data points (see Fig. 3). Each data point represents number of processors from a range of 2 to 32 and the size of the dataset from 13Mb to 200Mb. Most of the data points validate the Isoefficiency derived in equation 3 in the previous section. For example, doubling the number of processors in this case requires almost four fold increase in the data size to maintain the efficiency. At few points, a slight deviation is observed that can be attributed to the communication factor. It should be noted that a similar parallel implementation of Euler path method was not available at the time of this writing and therefore validating equation 4 using empirical data will be done in future.

5.1 C2C comparison

A closer look at the above relations reveals that the communication factor is significant in both approaches.

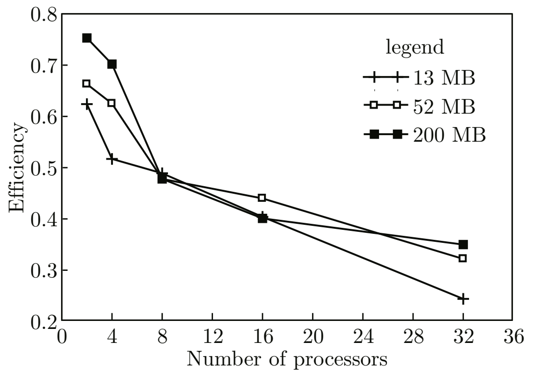

Fig. 3 Efficiencies for different datasets using parallel Overlap Layout Consensus based implementation

Both employ graph theoretical solutions that tend to be communication intensive due to complex and interrelated paths where vertices and edges are spread across multiple processors. The parallel version of Euler path approach has to process a relatively larger number of vertices and edges due to the nature of de-Bruijn graph, which is centric to the whole approach, resulting in a large C2C ratio compared to the overlap-layout-consensus approach. It must be noted here that a mere comparison of the C2C ratio between the two approaches does not necessarily favor one or the other algorithm; however, it affects the scalability as the number of processors grow larger.

5.2 Space comparison

Although most of the metrics used in the analysis of parallel algorithms are time-centric, *i.e.* focusing on time derivatives such as speedup and efficiency, the space complexity is also a critical factor in evaluating scalability. A general discussion on the subject can be found elsewhere (Reif and Spirakis, 1992). For the purpose of this paper, we can observe that both approaches can effectively utilize space using a good hashing scheme; however, a larger number of vertices and edges in Euler approach generally require more memory space.

6 Conclusions

Most published work has analyzed parallel software programs based upon various assembly algorithms by comparing runtime, lengths of the contigs, and error rate. There is a need to provide some additional evaluation of the algorithms, such as their efficiency and scalability for varying data size and complexity. In this paper, we presented a high level approach to analyze and compare two different methodologies used in genome assembly process to establish the importance of, and to provide an insight into, the relationship between the size of the input data and the number of processors available using generic parallel algorithms for the corresponding methodologies. Although a detailed comparison of different implementations is not feasible due the fact that each assembler uses unique components to improve speedup, accuracy, or handling of large data sets, having such insight into the overall approach should enable bioinformaticians to select a class of algorithms based on the type and size of the data they are working with and therefore study the cost effectiveness of a proposed parallel system.

7 Future directions

Ever since its first successful use, Sanger sequencing (Sanger *et al.*, 1977) has so far been the method of choice to obtain the reads used by assemblers. However, with the advent of new technologies (Bentley, 2006; Metzker, 2005), sometimes referred to as next generation sequencing, shorter reads containing fewer than 10^2 bases are obtained, which is very cost effective. The general implementations of overlap-layout-consensus are not very suitable for this kind of data because of the large number and smaller size of the reads, which also exacerbates the problem of repeats. Euler approach effectively addresses the issue of repeats and is designed to work with shorter reads. It appears that, as the next generation sequencing becomes a mainstream methodology and gains more popularity, the Euler path approach stands to attain a wider acceptance and a larger role to play in the realm of sequence alignment and, in particular, genome assembly.

References

[1] Batzoglou, S., Jaffe, D., Stanley, K., Butler, J., Gnerre, S., Mauceli, E., Berger, B., Mesirov, J., Lander, E. 2002. Arachne: A whole-genome shotgun assembler. Genome Research 12, 177–189.

[2] Bentley, D. 2006. Whole-genome re-sequencing. Curr Opin Genet Dev 16, 545–552.

[3] Edgar, R. 2004. MUSCLE: Multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Res 32, 1792–1797.

[4] Ewing, B., Hillier, L., Wendl, M., Green, P. 1998. Base-calling of automated sequencer traces using Phred. I. Accuracy assessment, genome research. Cold Spring Harbor Laboratory Press 8, 175–185.

[5] Grama, A., Gupta, A., Kumar, V. 1993. Isoefficiency: Measuring the scalability of parallel algorithms and architectures. IEEE parallel and Distributed Technology 1, 12–21.

[6] Gropp W., Lusk, E., Doss, N., Skjellum, A. 1996. A high-performance, portable implementation of the MPI message passing interface standard. Parallel Computing 22, 789–828.

[7] Gusfield, D. 1997. Algorithms on Strings, Trees and Sequences. Cambridge University Press, Cambridge, England.

[8] Huang, X., Madan, A. 1999. CAP3: A DNA sequence assembly program. Genome Research 9, 868–877.

[9] Huang X., Wang, J., Aluru, S., Yang, S., Hillier, L. 2003. PCAP: A whole-genome assembly program. Genome Research 13, 2164–2170.

[10] Hubbard T., Aken, B., Beal, K., Ballister, M., Caccamo, Y., Chen, L., Clarke, G., Coates, F., Cunningham, T., Cuts, T., Down, S., Dyer, S., Fitzgerald, J., Fernandez-Banet, S., Graf, S., Haider, M., Hammond, J., Herrero, R., Holland, K., Howe, K., Howe, N., Johnson, A., Kahari, D., Keefe, F., Kokocinski, E., Kulesha, D., Lawson, I., Longden, C., Melsopp, K., Megy, P., Meidl, B., Ouverdin, A., Parker, A., Prlic, S., Rice, D., Rios, M., Schuster, I., Sealy, J., Severin, G., Slater, D., Smedley, G., Spudich, S., Trevanion, A., Vilella, J., Vogel, S., White, M., Wood, T., Cox, V., Curwen, R., Durbin, X., Fernandez-Suarez, P., Flicek, A., Kasprzyk, G., Proctor, S., Searle, J., Smith, A., Ureta-Vidal, E. 2007. Ensembl 2007. Nucleic Acids Research 35, D610–D617.

[11] Metzker, M. 2005. Emerging technologies in DNA sequencing. Genome Research 15, 1767–1776.

[12] Pevzner, P., Tang, H., Waterman, S. 2001. An eulerian path approach to DNA fragment assembly. Proceedings of National Academy of Sciences of the United States of America 98, 9748–9753.

[13] Ralston, A. 1982. De Bruijn sequences - A model example of the interaction of discrete mathematics and computer science. Math Mag 55, 131–143.

[14] Reif, J., Spirakis, P. 1992. Expected parallel time and sequential space complexity of graph and digraph problems. Algorithmica 7, 597–630.

[15] Sanger, F., Nicklen, S., Coulson, A. 1977. DNA sequencing with chain-terminating inhibitors. Proceedings of the National Academy of Sciences USA 74, 5463–5467.

[16] Shi, W., Zhou. W. 2005. A parallel Euler approach for large-scale biological sequence assembly. In: Proceedings of the Third International Conference on Information Technology and Applications, Sydney, Australia, 437–441.

[17] Smit, A, Hubley, R., Green, P. RepeatMasker Open-3.0. 1996-2004 (http://www.repeatmasker.org).

[18] Smith, T., Waterman, M. 1981. Identification of common molecular subsequences. Journal of Molecular Biology 147, 195–197.

[19] Southern, E. 1975. Detection of specific sequences among DNA fragments Sseparated by gel electrophoresis. Journal of Molecular Biology 98, 503–517.

[20] Watson, J., Crick, F. 1953. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. Nature 171, 737–738.

[21] Zhao, F., Zhao, F., Li, T., Bryant, D.A. 2008. A new pheromone trail-based genetic algorithm for comparative genome assembly. Nucleic Acids Res 36, 3455–3462.