A THEORETICAL ANALYSIS OF SCALABILITY OF THE PARALLEL GENOME ASSEMBLY ALGORITHMS

Munib Ahmed, Ishfaq Ahmad
Department of Computer Science and Engineering,
University of Texas At Arlington, Arlington, Texas 76019, U.S.A
munib.ahmed@mavs.uta.edu, iahmad@cse.uta.edu

Samee Ullah Khan
Department of Electrical and Computer Engineering
North Dakota State University, ND 58108-6050, USA
samee.khan@ndsu.edu

Abstract

A rapid growth of the sequenced genomic data over the last two decades has far exceeded the advancement of both the algorithms and the computing horsepower required to expeditiously process and analyze it. Several algorithms have been devised and implemented to assist the process of genome fragments assembly: one of the most challenging and computationally intensive processes that may take weeks to assemble large size genomes. A few such algorithms have also been parallelized to speed up the process. However, there is a need to analyze such parallel algorithms using the specific metrics of parallel computing to ascertain their scalability and efficiency. The fact that the problem size can vary from a few million units of data to several billions, along with the vast differences in the degree of repetition in data sets, calls for the ability to establish an association between the nature of the problem and the algorithm that best solves it. This paper analyzes the scalability of two most widely used parallel genome assembly algorithms using Isoefficiency (Grama, Gupta & Kumar 1993) metric which will help provide a guideline to determine when and how to choose a particular genome assembly technique based on the nature and the size of the problem being solved.

Keyword: Genome Assembly, Isoefficiency, Parallel Computing, Scalability.

1 Introduction

The field of Bio-informatics has seen phenomenal advances since the early 1950s when the structural composition of DNA was laid out. Several hundred gigabytes of genomic data have been deciphered and stored. However, the limits imposed by the sequencing technology, that allows to read only a few hundreds of DNA bases at a time, makes the full genome assembly process very difficult. The whole genome shotgun sequencing (WGSS), a process of breaking up a DNA molecule so it can be read, results in millions of 700 to 800 bases long fragments, also called "reads", with very little details on how to put them back together to create the complete DNA blueprint of the specie being studied. Also, the sequencing process is all but perfect and yields many poor quality fragments further increasing the difficulty in assembling those fragments. To compensate for the poor quality, more than one (typically 6 to 8) copies of the DNA strands are

sequenced at the same time to allow for multiple overlapping fragments. This is called coverage and although it improves the chances of getting better overlaps, it increases the size of the data many-fold. Two most widely used assembly approaches are (1) Overlap-layout-consensus and (2) Euler Superpath technique. Both transform the problem domain into graph theory using Hamiltonian path and Euler path respectively. We will discuss those in detail in section 2. Section 3 provides an in-depth analysis of the parallel version of both methods and their scalability. Section 4 provides experiment details and the results, followed by conclusion in section 5.

2 Genome Assembly Algorithms

Although the genome construction efforts have been going on for a long time, the research gathered momentum with the commencement of Human Genome project. Many different algorithms have been proposed and implemented but the most widely used is what is known as Overlap-layout-consensus described below. Also, a relatively newer approach is to reduce the problem into a graph and then find an Euler path that would result in the assembled genome. In this paper we will focus on these two approaches. In order to avoid unnecessary complexity we will not discuss some less relevant details of these algorithms. Only a high level description that is pertinent to the topic of this paper is presented here.

2.1 Overlap-layout-consensus method

This most widely used approach to assemble genomes has many variations but the central idea is same, i.e. find overlaps and lay them out, generally as a directed graph, and then find Hamilton path. Normally, a Hamilton path is not found but the traversal yields many sub-paths that are recorded as contigs. A multiple alignment of all such contigs results in the consensus sequence. A parallel implementation is available for academic use in public domain (Huang et al. 2003).

(i) 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 $K_i = \{S_{i[x][x+k-1]}\}$ for all x where $1 \leq x < |S_i|-k$. 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 in such a sorted array. Unique overlaps are further verified using a local alignment algorithm like Smith-Waterman (Smith & Waterman 1981) or one of its several variations. For n sequences yielding m k-mers per sequence, a sequential algorithm to generate & sort k-mers and then align promising sequences takes $nm + nm\log(nm) + \alpha n$ where α is a product of genome coverage (how many copies of a sequence are possibly present in dataset) & 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 should take $pT_s + nT_t + nm/p + (nm/p)\log(nm/p) + \alpha n/p$ where T_s is the start time and T_t is the transfer time of a sequence between two processors. Note that T_s and T_t are very small compared to actual processing times.

(ii) 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, i.e. an edge u->v between u and v is removed if two other edges u->z and z->v are found and the sum of non-overlapping prefix lengths of u and z in u->z and z->v respectively is equal to the non-overlapping prefix of u in u->v. The reduced graph is to be traversed to find a Hamilton path which itself is NP-Complete. In practice, however, it is unlikely to find such a path both computationally as well as data-wise but the effort using heuristic algorithms results in sub-paths that are recorded as contigs. In terms of time complexity, a scan of nm entries constructs a basic graph and then removing the redundant arcs can be done in $O(n^2m^2)$ whereas a parallel implementation should be $O(n^2m^2/p + \log p)$ ignoring latency during merge operations among processors.

(iii) Finding consensus

Using the contigs produced above, a multiple alignment is performed to derive a consensus sequence. Recently, algorithms (Shi et al. 2005) have been presented that would require $O(nm\log n)$ to align n sequences assuming $m << n$. A parallel version would result in $O(nm\log n/p)$

2.2 Euler path method

Euler path based approach to fragment assembly, proposed by Pevzner (2001), reduces the problem into

an Eulerian path problem which lends itself to a polynomial computational complexity compared to the NP-completeness of the Hamiltonian path seen in the Overlap layout consensus approach as discussed in previous section. A parallel implementation of the algorithm has been reported elsewhere (Shi et al. 2005).

Following are major phases of the assembly and their time complexities.

i) K-mers generation: 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. In the parallel implementation the time taken is $O(nm/p)$.

ii) K-mers distribution: It is an extra step in parallel implementation and, using a hashing scheme, takes near linear time, i.e. $O(nm)$. In 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 asymptotic time complexity in this stage is $O(nm/p)$.

iii) Preparing data: The algorithm requires the multiplicities of k-mers to be computed for later use as a boundary condition when traversing the path. In sequential algorithm, that is achieved by using suffix arrays to store and use k-mers and therefore takes $O(nm\log(nm))$. In parallel environment, the multiplicity of each k-mer can be calculated as a side step when hashing and distributing the k-mers hence incurring insignificant time. If the coverage C, i.e. number of strands of DNA that were used to generate fragments, is not available, it takes $O(nm)$ to compute it. This can be achieved using some fast string matching algorithms, e.g. Gusfield's Z-algorithm.

iv) Forming contigs (building deBruijn graph): This phase 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 and therefore those nm edges can be constructed without even comparing with each other. That results in $nm(nm-1)$ or $O(n^2m^2)$ runtime complexity for this phase yielding E links (edges) among nm k-mers (vertices).

e) Traversing the Euler path: An Euler path can be found in $O(E)$ where $nm < E < n^2m^2$. A parallelized version should take $n^2m^2/p + pT_s + nT_t$, i.e. $O(n^2m^2/p)$

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, mostly done sequentially, and is therefore out of scope of this paper .

3 Analyzing scalability using Isoefficiency

An algorithm is usually characterized by its time complexity showing a relationship between runtime and the input data. In case of a parallel algorithm, the *speedup*, a runtime comparison to its sequential version, is normally used as the yardstick for measuring performance and finally the *efficiency*, the speedup per processor, is a gauge of utilization that characterizes a parallel algorithm. Assuming a constant data size W, the efficiency decreases with increasing number of processors as more processors are available to do same work previously done by fewer thus contributing to lesser utilization and more overhead in terms of communication and start up times denoted together as T_0. A parallel algorithm is considered scalable if increasing the resources, processors in most cases, requires only a proportional increase in the input data to keep the efficiency from decreasing significantly. Grama at el. (1993) presented a metric called *Isoefficiency* to relate the number of processors to the input data while keeping a constant efficiency thereby allowing to measure scalability of an algorithm, i.e. how much data should be increased to maintain efficiency in response to an increased number of processors. The research provided some useful relations that we can borrow to perform our analysis. Firstly, the overhead time can be derived in terms of sequential & parallel times as

$$T_0 = pT_p - T_1 \quad (1)$$

where T_p is the parallel time taken by p processors to process the input data and T_1 is the sequential time, i.e. using single processor. Secondly, $W \propto T_0$, stating that as we increase the number of processors the overhead time increases and therefore requires the input data to be increased proportionally to maintain the efficiency. Considering efficiency E to be the constant,

$$W = ET_0 \quad (2)$$

In the following section, we will compare the scalabilities of the two genome assembly approaches using the above mentioned metric and parameters.

The sequential time complexities for different phases of overlap-layout-consensus approach can be summed up 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 the equation (1) and (2), we simplify the above as

$$W = E\,(p^2T_s + pnT_t + p\log p) \quad (3)$$

A similar workout for the Euler path approach should yield

$$W = E\,(p^2T_s + pnT_t + nmp) \text{ for } p>>1 \quad (4)$$

It can be observed that, due to the asymptotically higher factor, both (3) and (4) have a quadratic complexity in terms of the number of processors. We can therefore 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_{new}^2/p^2. A closer look at the above relations reveal that the communication overhead is significant in both approaches whereas the computational work is more time consuming in overlap-layout-consensus as compared to the Euler path approach.

4 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 figure 1). 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 4 times 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.

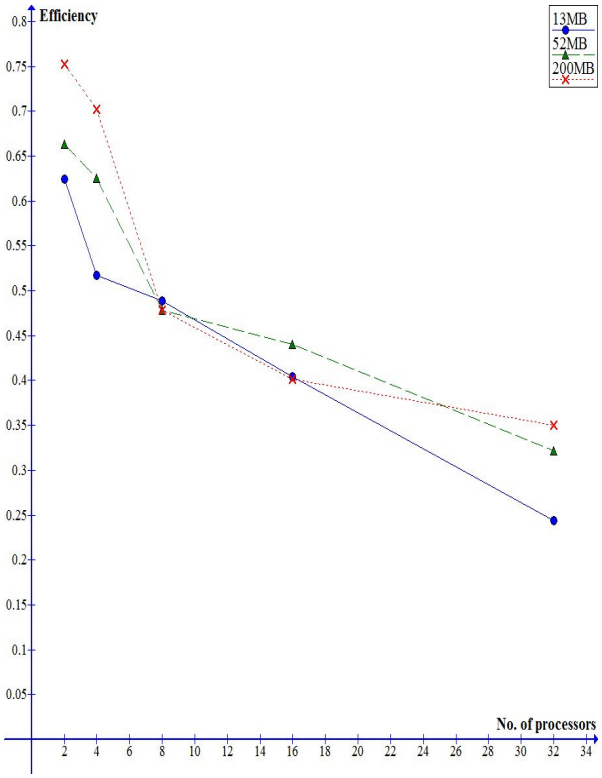

Figure 1: Efficiencies for different datasets.

5 Conclusion

Most published work has analyzed parallel software programs based upon various assembly algorithms by comparing runtime, contigs' lengths, and error rate. There is a need to compare and provide some additional evaluation of the algorithms such as their efficiency and scalability for varying data size and complexity. In this paper, we have 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, i.e. overlap-layout-consensus and Euler Superpath. Having such insight 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.

REFERENCES

Grama, A., Gupta, A., & Kumar, V 1993, 'Isoefficiency: Measuring the scalability of parallel algorithms and architectures', *IEEE parallel and Distributed Technology*, 1(3):12-21.

Gropp, W, Lusk, E, Doss, N, Skjellum, A 1996, 'A high-performance, portable implementation of the MPI message passing interface standard', *Parallel Computing*, vol. 22, pp. 789-828.

Huang et al. 2003, 'PCAP: a whole-genome assembly program', *Genome Res.*", 13, 2164–2170.

Hubbard et al. 2007, 'Ensembl 2007', *Nucleic Acids Res.* Vol. 35, Database issue:D610-D617.

Pevzner et al 2001, 'An Eulerian Path Approach to DNA Fragment Assembly', *Proceedings of National Academy of Sciences of the United States of America*, 98(17):9748-9753.

Shi et al. 2005, 'A Parallel Euler Approach for Large-Scale Biological Sequence Assembly', *Proceedings of the Third International Conference on Information Technology and Applications*.

Smith, TF & Waterman, MS 1981, 'Identification of common molecular subsequences', *Journal of Molecular Biology*. Vol. 147, 195–197.