PARALLEL ASSEMBLER FOR FUZZY GENOME SEQUENCE ASSEMBLY

Sara Nasser Adrienne Breland Frederick C. Harris Jr.

Department of Computer Science and Engineering, University of Nevada Reno
Reno, NV 89557
{sara, adrienneb, fredh}@cse.unr.edu

Abstract

Assembly is an NP-Hard problem, which involves comparing fragments that have a time complexity of $O(n^2)$. This paper presents a parallel approach for sequence assembly. The parallel technique is based on classification to group organisms by similarity rather than an embarrassingly parallel approach that requires duplication of the data across all nodes. This process of classification, based on DNA signatures, is useful in parallel assembly as it divides the problem into subtasks. The classification is followed by a fuzzy assembly of these fragments. The assembly of each tasks is achieved by using a modular approach. The parallel implementation of the assembly shows a speedup in assembly while preserving coverage of fragments.

1 Introduction

Computational biology is an expanding interdisciplinary field that applies computational sciences to solve biological problems. Several biological problems that were tedious or impossible to solve manually have been tackled or are being tried using computational techniques. Some of these problems include DNA sequence alignment and DNA sequence assembly. Finding the genome sequence is an important step that leads to understanding of the organism structure and drives further research into discovery of genes and study of the mechanisms that can explain the properties of an organism.

Genome sequences are large in size and can range from several million base pairs in prokaryotes to billions of base pairs in eukaryotes. For example, Wolbachia genome, a bacteria has 126 million base pairs (Mb) , Arabidopsis thaliana, a plant has 120Mb, and the human genome is 3.2 billion base pairs. The whole genome cannot be sequenced all at once because available methods of DNA sequencing can handle only short stretches of DNA at a time. Although genomes vary in size from millions of nucleotides in bacteria to billions of nucleotides in humans, the chemical reactions researchers use to decode the DNA base pairs are accurate for only few hundreds of nucleotides at a time [13] or generally in the range (40 Kbp -1000 Kbp) depending on the technology. Obtaining shotgun sequences has allowed sequencing projects to proceed at a much faster rate, thus expanding the scope of the realistic sequencing venture [12].

The problem of sequence assembly is acquiring data and assembling the DNA fragments or sequences into an entire genome sequence. There are two important aspects to understanding the problems that arise in genome assembly: the genome is cut into smaller portions, and fragments or sequences are cut at random positions. To obtain the original sequence these fragments need to be combined. The fragments are combined by determining overlaps between fragments. Thus, to combine fragments together by finding overlaps, portions of fragments need to appear more than once. Using a sequence fragment once cannot create overlaps. Therefore multiple copies of original sequences are made to ensure that the entire sequence is covered atleast once and several regions occur more than once. This process is generally referred to as coverage of nX, where n is the number of copies and X is the sequence. Coverage of 8X or 10X is widely accepted and it has been shown it is sufficient to reconstruct the entire sequence. Thus for a genome sequence of length 4(million)MB, if the sequence fragments of length around 500 bp are generated we need 80,000 sequences.

Following the sequencing process, an assembler pieces together the many overlapping bases and reconstructs the original sequence [13]. The process explained above is known as the whole-genome shotgun method. There are three main steps involved in the assembly of sequences. The first step, Sequencing, breaks the genomic DNA into fragments by sonication, a technique which uses high-frequency sound waves to make random cuts in DNA molecules [3]. In the assembly phase the sequences are combined to form contiguous sequences. Generally assembly leads to longer sequences of contigs that do not overlap. The final phase is finishing, in this phase contigs are joined by closing physical gaps. This phase is the most time con-

suming phase, which can be improved by using more than one clone libraries are prepared using different vectors. As different vectors clone sequences differently, using more than one vector can help improve coverage. Fragments that were not cloned by one vector could be cloned by the other. Thus gaps could be reduced as overall coverage increases when sequences are generated using different vectors.

The large dataset genome sequences created due to duplication and the processing time complexity makes sequence assembly a viable candidate for parallel processing. Parallel processing is a technique to divide a task into subtasks and to run the subtasks simultaneously on more than one processor, such that, the overall time required to perform the task is less than the time to perform the task sequentially. The results from each processor are combined in the end and some post-processing is done. As processors have become less expensive, parallel systems became affordable and have popularity to solve some NP-Hard problems that could not have been approached earlier.

The applicability of parallel processing to a problem depends on how well the problem can be divided into subtasks and the whether subtasks can be processed independently. For example, Monte Carlo methods use repeated random sampling to solve a problem, each process is independent of other processes. Therefore, these methods can be parallelized easily [18].

In this paper we present a parallel implementation of the fuzzy sequence assembly, and discuss load balancing issues and the new approach in Section 3. In Section 3 we describe the sequential approach to dividing the process into modules. This modularization leads us to the parallel implementation of the algorithm in Section 4. Results and discussions are presented in Section 5.

2 Background

Utilization of parallel techniques have been sought to a limited extent in bioinformatics. Wide spread use of parallel processing in this domain has been limited by the availability of applications that exploit parallel architectures [6].

Bioinformatics databases have grown tremendously in the past few years. An example is the growth of sequences present in GenBank by 200% over the past 6 years. Techniques such as compression and hashing have been used in the past to address the issues of size and speed in assembly. Compression of DNA sequences was performed to reduce the size of the fragments and perform faster comparisons. Since the first method to compare compressed strings was proposed in [1], there have been several techniques to apply compression to DNA sequences. DNA compression results in increased speed, but is only suitable for exact matching [7, **?**]. Therefore, this technique is not well suited to approximation problems such as assembly. In addition to compression techniques there are methods that perform analysis of data. FASTA is a rapid heuristic search method for protein and DNA alignment that performs statistical analysis of data prior to alignment [8], resulting in an improvement in performance. Hashing techniques are used to search for word patterns of high frequency which makes assembly faster and heuristics have been used to reduce number of comparisons required in assembly. These methods, although useful, have some drawbacks, such as, the need for decompression before assembly. Hashing techniques make assembly very fast but limit comparisons of overlaps to high frequency words.

To comply with the fast growth of these databases, high performance computing has been sought as an answer to increase performance. Application of parallel techniques can make bioinformatics techniques faster without compromising the results. A parallel algorithm for DNA alignment, in which alignment is calculated on each node and then gathered into a single global alignment on the root, is presented in [15]. **mpiBLAST** is an example of an open-source, parallel implementation of NCBI BLAST to improve the performance of BLAST by several orders of magnitude while scaling to hundreds of processors [9]. *ParAlign* is another parallel approach for DNA alignment and search within databases [17]. These tools with few others have led the way to parallel implementation of bioinformatics algorithms.

2.1 Embarrassingly Parallel Computation

The first attempt to parallelize assembly was using an embarrassingly parallel computing technique, where a data set can be divided into completely equal independent tasks that execute simultaneously [18]. This also suggests that the only communication between the processors is when the processes start and end. These are the simplest category of parallelizable problems as they are easy to implement and require no special techniques. Examples of such problems are mandle-brot, Monte Carlo methods and some image processing transformations.

An embarrassingly parallel implementation of sequence assembly is to divide the DNA fragments into groups of fixed size. The groups are assigned to different nodes/processors, where each node has access to the entire fragment data, but only assembles the sequences within the group. This strategy, though simple can increase the performance of assembly as n processes share the work. A master-slave architecture works well for this kind of implementation. The master process performs initial grouping of fragments and distributes the jobs to the slaves. The slaves receive individual jobs and assemble the data given to them and return the assembled sequences (or contigs) to the master. The master post-processes the contigs to eliminate any duplicate contigs and finalizes the assembly.

There are some drawbacks of the embarrassingly par-

allel implementation to assembly. This parallelization can improve performance, but still requires the duplication of the data all the slaves, as all slaves need access to the entire data set. This assembly can create a large number of duplicate contigs as each slave works independently. The presence of duplicate contigs can create another challenge as they increase the post-processing on the master node and network traffic.

To address the issue we propose classification of data in to groups. In this approach groups of sequences are assigned to each processor, the process assembles the fragments within the group. This approach reduces redundant assembly and duplicate contigs, as it performs assembly on limited number of sequences. The slaves need to communicate to exchange contig information. A master-slave architecture is proposed for the parallel implementation based on this partitioning. To increase efficiency classes are created using the technique listed in [10].

3 Module Assembly Sequential Approach

Equipped with a sequential program to classify sequence fragments into taxonomic groups, we began analyzing the parallel implementation that best suits assembly. This section presents the details of parallel implementation.

The process of assembly starts with identification of DNA signatures, followed by fragment classification to divide the data sets into smaller categories. The classes ideally represent two significant properties: 1) they contain fragments belonging to the same region in the genome and 2) they have continuity and can form contigs for the local regions of the genome. This grouping is followed by assembly, which is divided into three major steps. These steps of a sequential assembly are used to design the tasks for the parallel processing as shown in Figure 1. The steps start with assembly of the K individual classes.

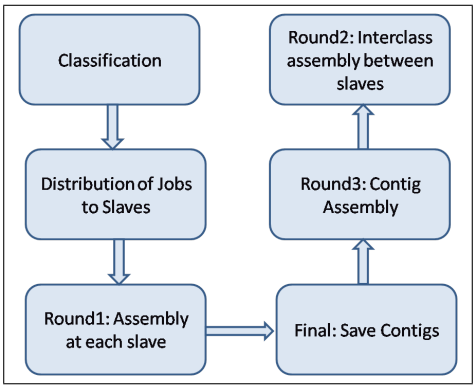

Figure 1: Module Assembly

3.1 Round 1: Divide-and-Conquer

A divide-and-conquer strategy is used in the first round of assembly. The sequences are classified using the approach presented in [10]. Sequence classification can be approached in two ways: 1) forming few classes of large size, 2) forming smaller classes that are compact, and, few of the small classes represents one large group. Because improper loads on nodes can occur as the sizes of a class is unknown, classes of larger size can become a bottleneck. We use the second approach as it creates smaller groups that can be assembled faster. This approach also balances the load on a slave and is easier to implement, as assigning more than one job to a slave is simple and does not add overhead during post-processing. Consider a genome, that contains a total of 8 million base pairs. A 10X coverage of this genome results in 80 million base pairs or over 100,000 sequences of length 700 base pairs. If the classification of this dataset results in two classes, assuming even distribution of the groups there will be two large groups of 50,000 sequences each for each slave. The situation gets worse if the data is not distributed evenly and can result in load balancing issues as illustrated in Figure 2. Thus, we choose to select smaller groups, so that the slaves get smaller classes to assemble, which also results in faster in-class assembly. Further details of load balancing will be covered in Section 4.1.

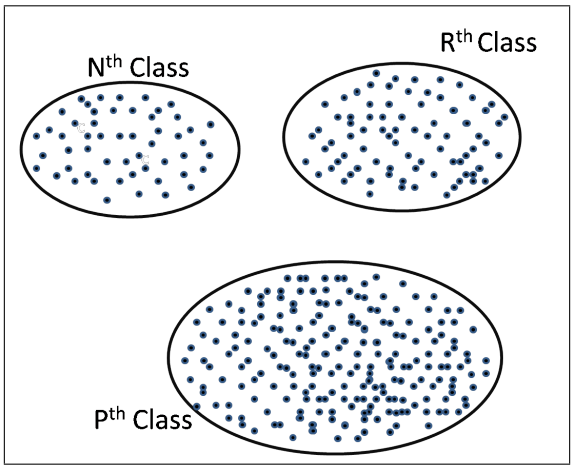

Figure 2: Unbalanced Classes

Each of the classes is assembled to form contigs within the class. Each class acts as an independent unit and does not interact with any other class. Two kinds of results are obtained after this module is completed: 1) Contigs formed within the class (assembly of one or more fragments by overlap), and 2) Singlets in the class (singlet refers to a fragment that was not assembled with any other sequence). Singlets are found in a class if a sequence cannot be part of any contig. In this case, a singlet can also be

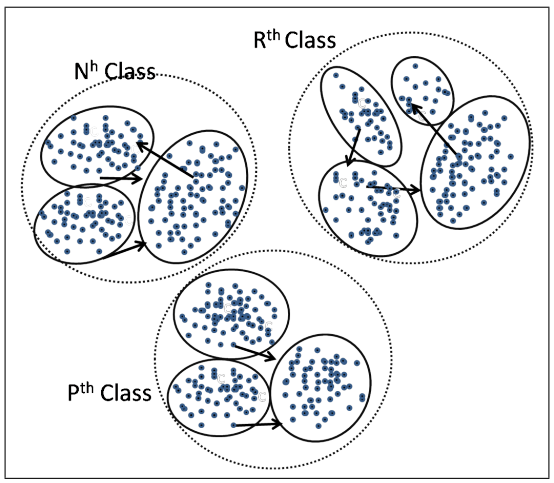

Figure 3: Round 2 Assembly: Review-and-Regroup

found if a sequence was not classified correctly or even if it was classified correctly it does not have a good overlap with any of the sequences present in the class. Singlets are determined during the assembly process by identifying sequences that could not assemble. In such case the singlet can be assembled with sequences from another class.

3.2 Round 2: Review-and-Regroup

The second module corrects any mis-classifications made during the first module, reassigns singlet fragments between classes, and performs intra-class assembly. If all classes are compared a total of p(p-1)/2 comparisons will be required, which can add network overhead. Additionally, if there are large number of singlets then the parallel version can slow down to the performance of the sequential version. To avoid comparing all possible classes we use heuristics to determine whether the classes have any similarity, and only classes that exhibit similar signatures to each other are used to perform intra-class assembly. The heuristics are described in Section 3.4. Round 2 can be further enhanced by comparing singlets in all classes with each other.Figure 3 depicts a representation of hierarchical Round 2 grouping.

3.3 Round 3: Collect-and-Combine

As the name of the module suggests this section of the algorithm collects the contigs assembled in Round 1 and Round 2. It combines the local contigs to form larger contigs and connects overlapping regions between classes. The contigs are processed to remove duplicates and longer contigs are created by further assembly.

3.4 Class Restrictions

Round 2 assembly to review and regroup sequence fragments, becomes computationally intensive as the number of singlets and classes increases. Therefore, to keep the task from growing, restrictions are imposed on classes. Restriction on classes ensures that classes communicate with certain classes only and not all the classes present in the data set. Class restrictions are imposed by two techniques described in this section.

3.4.1 Nucleotide Content

Two frequencies are used for analyzing the closeness of sequence fragments to each other, Base pair content and GC Content.

Base pair frequencies (BPF) of a fragment is the frequency of A,C,T and G present in the fragments. BPC allows us to compare if two sequences can a potential overlap without actually assembling. It is calculated for the ends of the fragments, rather than the whole sequence. Using the entire fragment may include information that is not required for assembly. As assembly requires similarity at the ends of the fragments and if these are not similar then comparing the rest of the fragment cannot yield information important to assembly. If the BPC at the sequence ends is not similar then the sequences have less or no chance of assembly. Thus each class has a range of BPC, that is compared before intra-class assembly is performed.

$$BPF_i = \frac{\sum X_i}{n} \quad (1)$$

In Equation 1, *i* refers to a nucleotide type and *n* is the length of sequence compared.

The four nucleotides of a DNA strand are connected by hydrogen bonds between them. The nucleotide A bonds specifically with T and the nucleotide C bonds with G. AT pairs have two hydrogen bonds and GC bond pairs have three hydrogen bonds, making the bonds more thermostable. Thus, the GC content in an organism can sometimes be used to determine certain characteristics about that organism.

Organisms are generally biased in the distribution of A, C, G and T. This fundamental property of organisms is used in separating one genome from another. Certain organisms contain higher percentages of GC and are thus known as GC rich, while some other organisms are dominated by AT and are known as AT rich. Organisms belonging to the phylum *actinobacteria* are listed as GC rich [11]. On other hand, *Arabidopsis thaliana*, a very popular plant research organism has less than 40% CG content and thus has more AT content. Theoretically, GC content percentages of elements in a class are similar and within a small range.GC content is expressed as the percentage of C and

G present in the fragment and is calculated as follows:

$$\frac{C+G}{A+C+G+T} \times 100 \quad (2)$$

In Equation 2, A, C, G and T refer to the frequencies of the occurrences of the four base pairs. In order to restrict comparisons of singlets with classes that are close, GC content of classes is used. Each class has number of fragments that are grouped by similarity. GC content is one of the factors that is used to form these classes. We compute the low and high GC content of each class. A singlet is compared with elements of this class if it's GC content is within this range. Using GC content also helps in separation of organisms, which is ideal for metagenomic assembly. The application to metagenomic assembly is discussed in Section 5.

3.4.2 Hierarchical Clustering

The second method to impose class restriction is by limiting comparisons of classes by the fuzzy distance between classes. The distance $d_{i,j}$ of a sequence from each cluster can be calculated as follows:

$$d_{i,j} = min(\mu_j^r), for\ all\ j = 0, \ldots, k \quad (3)$$

Here μ is calculated using a weighted fuzzy average (WFA). Let $\{x_1, ..., x_P\}$ be a set of *P* real numbers. The weighted fuzzy average using the weight w_p for x_p is given as:

$$\mu^r = \sum_{p=1}^{P} w_p^{(r)} x_p,\ r\ = 0, 1, 2, \ldots \quad (4)$$

Here *x* is the parameter or DNA feature and *p* the number of DNA features. The number of the iteration is given as *r*. A hierarchical clustering of classes are created by combining smaller classes whose distance is less that $d_{i,j} < \delta$. In this approach hierarchical clustering is used in Round 2 to restrict class comparisons to subclasses within the same larger group. Thus classes that are closer in nucleotide frequencies are virtually assigned to a group and only compared with classes within the virtual group. Classes from different virtual groups are not compared during round 2.

4 Master-Slave Architecture

Master-slave architecture is a popular technique of dividing tasks in a message passing parallel implementation. In this technique, one processor is the master and the rest of the processors are slaves. The master performs the classification of sequences, and assigns tasks to the slaves. The slaves perform the assembly and return results back to the master. The master performs the final processing before finishing the task.

4.1 Load Balancing

Load balancing is an important issue in parallel implementation of any algorithm. Load balancing aims to distribute jobs evenly across all process to maximize the overall performance. Several factors affect the balancing of a parallel system, such as different processing speeds of processors and different data sizes. If all processors are given equal size data sets and if everything else is fixed then maximum speedup can be expected. Generally, all slaves may not finish at the same time. As sequences are classified based on nucleotide similarities, the classes formed are not necessarily the same size. Therefore, a class that is much bigger than other classes can be found.

Unbalanced classes can be avoided by imposing restriction during classification. For example, if classes are large then they cannot be merged. Nevertheless, if the signatures are quite similar then this restriction will not prevent fragments from belonging to a class.

To address this issue, a method for dividing large classes during assembly is presented. In a regular master-slave implementation of assembly, each slave gets a fixed number of classes. In the proposed load balancing approach, depicted in Figure 4, if a class is large in size then an embarrassing parallel technique is applied to divide the class between more than one processor. Each slave is responsible for assembling an assigned portion of the class, even though they have access to all the sequences in the class. This splicing does not affect the assembly and may add certain amount of overload to the slaves sharing the class. But overall is faster than one slave performing the entire assembly.

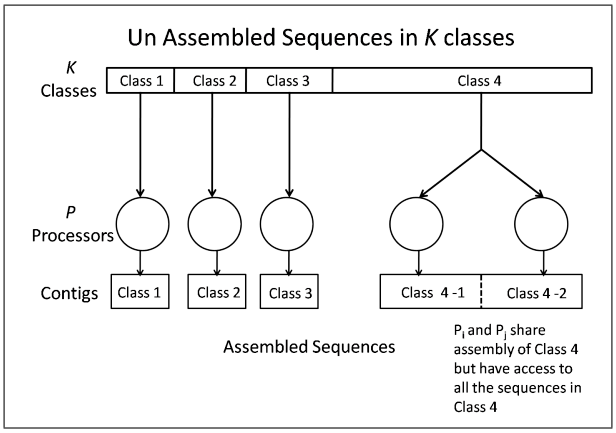

Figure 4: Static Load Balancing using Mixed Mode Simple Parallelization

5 Results and Analysis

Accuracy of assembly is measured by the amount of original genome that can be recovered. Parallel implementation generally aims at increasing the performance of a process. Thus we will measure the two outcomes: coverage and speedup. The speedup attained via the parallel processing is measured to analyze the performance of the parallel algorithm. The speedup factor *S(n)*, is defined as a measure of relative performance between a multiprocessor system and a single processor system, and is given by

$$S(n) = \frac{t_s}{t_p} \tag{5}$$

Where, t_s is the execution time using one processor and, t_p is the execution time using a multiprocessor system. Assembly of three genomes *Yersinia pestis Pestoides F plasmid CD* complete sequence, containing 71,507 base pairs, was performed are results are shown below. Figure 5 compares the speedup of the assembly for each genome.The speedup for assembly is given in Figure 5, which indicates that the parallel approach increases the performance of assembly. The parallel code using MPI was test on a research grid cluster running opterons with 2GB of RAM. Microorganisms live in communities, and their

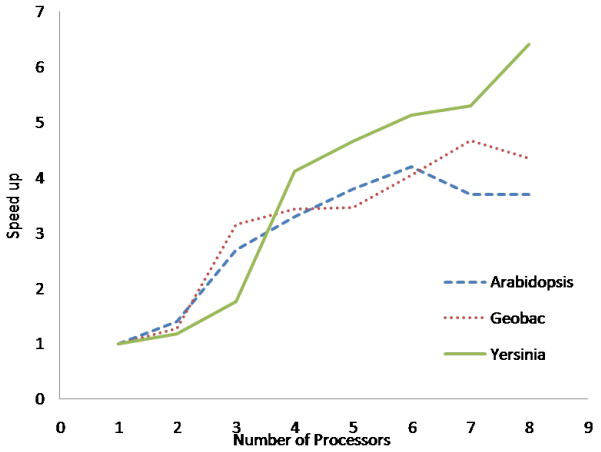

Figure 5: Speedup for Assembly of three genome sequences *Yersinia pestis Pestoides F plasmid CD*, *Arabidopsis thaliana genomic DNA, chromosome 3, BAC clone:F11I2*, and *Geobacillus thermodenitrificans NG80-2 plasmid pLW1071*

structure and behavior is influenced by their habitat. Most microorganisms genomes are known from pure cultures of organisms isolated from the environment, be it a natural organism-associated (i.e, human) or artificial system. New techniques in genomic sciences have emerged that allow an organism to be studied in its natural habitat as part of a community. Research has broadened from studying single species to understanding microbial systems and their adaptations to natural environments. These techniques have been achieved by developing methods that can sequence mixed DNA directly from environmental samples [2, 14]. This field of metagenomics (environmental genomics) involves the sampling of microbial DNA from natural environments rather than relying on traditional single-species cultivation techniques. This sampling, coupled with rapid developments in molecular biology, is changing our understanding of bacterial evolution and naturally existing microbial systems. Metagenomics is the application of modern genomic techniques to the study of microbial communities in their natural environments, bypassing the need for isolation and lab cultivation of individual species [4, 16].

Whole-genome shotgun sequencing of environmental DNA gained attention as a powerful method for revealing genomic sequences from various organisms in natural environments [2, 16]. In a metagenomic sample an organism's DNA is not only sliced into small fragments but also mixed with other organisms' DNA fragments, thus creating a mixed population of fragments. This group of heterogeneous fragments needs to undergo assembly. A classification based approach for assembly suits metagenomic data as it can group organisms into classes. Preliminary tests are performed on an Acid Mine Drainage Metagenome data set. This metagenome has been assembled thus is a good benchmark for assembly and verification of assembly. The two scaffolds belong to *Ferroplasma* and *Leptospirillum* sets.

Results of assembling scaffolds obtained from the AMD environmental genome sequences from NCBI are shown in Table 1.

	Percentage Genome Recovered
Genome	mpiCFGS
Ferrosplasma	96.57%
Leptospirillum Sp Type	94.02%

Table 1: AMD Scaffolds Parallel Implementation
In the table, mpiCFGS refers to the parallel implementation presented in this paper to assemble sequences from the AMD metagenome. The shotgun sequences were created artificially with average length of 700 bp. Contigs up to length 4000 bps were obtained.

Preliminary results of assembling scaffolds of AMD data belonging to *Ferroplasma* and *Leptospirillum* sets are displayed show a good recovery of the genomes.

6 Conclusions and Future Work

The results indicate that parallel implementation improves the speed of assembly. Parallel implementation also allows us to assemble larger data sizes at reasonable speeds, which was difficult due to memory limitations. As nodes share information after initial assembly, intra-class assembly is also performed. We compared different parallel im-

plementations for assembly to analyze which implementation suits best. A simple load balancing technique is proposed, which can be improved by addition of dynamic load balancing. An improvement in load balancing can also result in a smoother speedup.

Future goal of this research is to move in the direction of metagenomic assembly. This parallel assembly is a potential tool for metagenomic data as the fragment data set is much larger than single organisms. Metagenome sets also contain more than one organism which can be assembled individually. Thus each processor can run assembly job independenent of other processors. The challenge with a parallel assembly of metagenomes is classification of data into groups that can represent organisms within it. There are certain issues with classification of closely related data, irrespective of the classification accuracy the parallel approach has shown improvements.

7 Acknowledgments

This research was supported in part by the NSF EPSCoR CIP Fellowship (RING-TRUE III Award number: 0447416). The authors also wish to thanks Dr. Alison Murray from Dessert Research Institute, Reno, NV for her insight to the problem.

References

[1] Amihood Amir and Gary Benson. Two-dimensional periodicity in rectangular arrays. *SIAM Journal on Computing*, 27(1):90–106, 1998.

[2] O. Beja, M.T. Suzuki, E.V. Koonin, L. Aravind, A. Hadd, L.P. Nguyen, R Villacorta, M Amjadi, C Garrigues, SB Jovanovich, RA Feldman, and EF DeLong. Construction and analysis of bacterial artificial chromosome libraries from a marine microbial assemblage. *Environmental Microbiology*, **2**:516–529, 2000.

[3] TA Brown. *Genomes*. Garland Science, 3^{rd} edition, 2006.

[4] K Chen and L Pachter. Bioinformatics for whole-genome shotgun sequencing of microbial communities. *PLoS Computational Biology*, **1**:106–112, 2005.

[5] E.S. Gough and M.D. Kane. Evaluating parallel computing systems in bioinformatics. In *Third International Conference on Information Technology: New Generations*, pages 233–238, 2006.

[6] Chen Lei, Lu Shiyong, and J. Ram. Compressed pattern matching in dna sequences. In *Computational Systems Bioinformatics Conference*, pages 62–68, 2004.

[7] D.J. Lipman and W.R. Pearson. Rapid and sensitive protein similarity searches. *Science*, **227**:1435–1441, 1985.

[8] mpiBLAST. mpiblast: Open-source parallel blast. http://www.mpiblast.org/, NCBI, 2008.

[9] Sara Nasser, Adrienne Breland, Frederick C. Harris, and Monica Nicolescu. A fuzzy classifier to taxonomically group dna fragments within a metagenome. In *North American Fuzzy Information Processing Society*, pages 1–6, 2008.

[10] NCBI. National center for biotechnology information. http://www.ncbi.nlm.nih.gov/, NIH, 2007.

[11] Edmund Pillsbury. A history of genome sequencing. Technical report, Yale University Bioinformatics, 2001.

[12] Mihai Pop, Steven L. Salzberg, and Martin Shumway. Genome sequence assembly: Algorithms and issues. *IEEE Computer*, pages 47–54, July 2002.

[13] M.R. Rondon, P.R. August, A.D. Bettermann, S.F. Bradly, T.H. Grossman, M.R. Liles, KA Loiacono, BA Lynch, IA MacNeil, C Minor, CL Tiong, M Gilman, MS Osburne, J Clardy, J Handelsman, and RM Goodman. Cloning the soil metagenome: a strategy for accessing the genetic and functional diversity of uncultured microorgansims. *Applications Environmental Microbiology*, **66**:2541–2547, 2000.

[14] Thomas Royce and Rance Necaise. A parallel algorithm for dna alignment. *Crossroads, AMC Student Magazine*, **9**(3):10 – 15, 2003.

[15] JL Stein, TL Marsh, KY Wu, H Shizuya, and EF DeLong. Characterization of uncultivated prokaryotes: isolation and analysis of a 40-kilobase-pair genome fragment from a planktonic marine archaeon. *Journal of Bacteriology*, **178**:591–599, 1996.

[16] Rognes T. Paralign: a parallel sequence alignment algorithm for rapid and sensitive database searches. *Nucleic Acids Research*, **29**(7):1647–52, 2001.

[17] Jeff Wallace, Gregory Vert, and Sara Nasser. An efficient method for compressing and searching genomic databases. In *High Performance Computing and Simulation*, pages –, 2007.

[18] Barry Wilkinson and Michael Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice Hall, 2^{nd} edition, 2004.