Scalable Methods for Genome Assembly

Priyanka Ghosh
Washington State University, School of EECS
priyanka.ghosh@wsu.edu

ABSTRACT

Genome assembly is a fundamental problem in the field of bioinformatics wherein the goal lies in the reconstruction of an unknown genome from short DNA fragments obtained from it. With the advent of high-throughput sequencing technologies, billions of reads can be generated in a few hours. My reseach deals with addressing the challenges of conducting genome assembly at scale and developing new methods for conducting extreme-scale genome assembly suited for microbial genomes, and complex eukaryotic genomes. Our approach to the problem is two-fold, wherein we target fast and space-efficient assemblies of moderate sized genomes on a single node with our shared-memory based implementation (*FastEtch*). Secondly we aim to tackle the assembly of very large genomes with our distributed-memory based approach (*PaKman*), in an attempt to minimize the turnaround time such that it could potentially pave the way to conducting the assembly process in real-time.

1 INTRODUCTION

Genome assembly is an actively researched topic with wide applications in the field of computational biology. Genome and metagenome assembly is also recognized as one of the key applications in the DoE Exascale Computing Project(ECP), with goals to accelerate these studies for cancer research. Given an input of short fragments known as "DNA reads" obtained from an unknown genome, the goal of *De novo* genome assembly involves the reconstruction of the unknown long genome (DNA sequence). Over the last decade, with the advent of multiple high-throughput sequencing technologies (e.g. NGS), billions of reads can now be generated in a matter of hours, leading to vast amounts of data accumulation per day.

Parallel tools for conducting genome assembly presents a promising solution in order to meet the growing demands of handling such huge amounts of data, especially with the availability of new parallel architectures. However, at present there exists a wide gap in terms of the processing power between massively parallel NGS technologies and the ability of current state-of-the-art assemblers to analyze and assemble large genomes.

There are numerous short read assemblers that have been developed over the last decade (e.g., [1, 5, 7, 8]). Nearly all of them use a string data structure called the de Bruijn graph (DBG) to compute their assembly, and follow a 3-step approach: The de Bruijn graph is constructed using all the substrings of length k (called k-mers) in the input reads as "vertices", and all the $k+1$-mers that exist in the input reads as "edges". The next step involves error correction, which locates and prunes paths induced by erroneous k-mers that are likely to be manifestations of sequencing errors. The last step is to generate a set of assembled *contigs* by traversing paths in the pruned and corrected graph. Of the three steps, the first step of de Bruijn graph construction is typically most memory- and time-intensive, potentially taking hours to even days for assembling moderate sized eukaryotic genomes, and requiring tens to hundreds of gigabytes of memory.

Furthermore the essential challenges in performing genome assembly include: a) The presence of sequencing errors in the generated short reads. Even with errors less than 1%, can quickly overwhelm the computational capacity, with large number of false k-mer vertices. b) Read datasets comprise of billions of reads, when fragmented generate several billion vertices in the de Bruijn graph, thereby presenting a problem that is computationally demanding with respect to both time and memory. c) Several genomes are repetitive in nature i.e. a substring might appear more than once in different positions in the reference genome sequence, thereby adding further complexity to the assembly algorithm.

Thus the primary contributions of my PhD thesis can be summarized as follows:

- Our approach to the problem of genome assembly is implemented on two separate platforms, the first is named *FastEtch*. This is a new method for generating genome assemblies using probabilistic data structures (namely *Count-Min* sketch [3]). This parallel version executes efficiently on shared-memory platforms with a minimal computational footprint (both memory and time) without compromising on assembly quality. Upon testing *FastEtch* for datasets ranging from 1-55 GB on a single node of the NERSC Cori machine (scaling up to 32 cores) , we were able to obtain performance improvements of up to 30% in comparison to several other state-of-the-art approaches.
- Our second approach named *PaKman* is a fully distributed (MPI + OpenMP) extreme-scale genome assembler aiming to tackle the assembly of very large genomes (typically above 1TB data), with the goal of achieving processing capabilities in real-time - i.e., by scaling to thousands of processes and generating complex genome assemblies in minutes on the NERSC Cori supercomputer. The development of *PaKman* is currently in progress. Going forward targeting manycore architectures like Tilera, Intel Xeon Phi will permit additional scope for further exploiting data-locality optimizations.
- As part of future work, we intend to extend our algorithm to other assembly frameworks such as transcriptome assembly as well as the high-coverage ultra-deep sequencing method, with data sizes roughly 100x larger.

2 METHODOLOGY

2.1 Shared-memory approach: *FastEtch*

The *FastEtch* algorithm for genome assembly has two main steps: a) Graph Construction: Constructs an "approximate de Bruijn graph" using the input reads and CM sketch (a sublinear space data structure used for summarizing massive data streams in applications). b) Contig Generation: Generates contigs by performing edge detection and path enumeration of the approximate de Bruijn graph. This method is the first approach, to the best of our knowledge, to use the Count-Min Sketch in the graph construction step. The approximate de Bruijn graph, stores only a subset of vertices (k-mers), and detects the edges on-the-fly as contigs are generated - thereby generating savings on both vertex and edge space requirements, and consequently, the time to solution. Additionally, through a choice of parameters, it also offers a way to control the quality-performance (time, memory) trade-off in the output.

We compared the performance of *FastEtch* with other state-of-the-art de Bruijn graph based short-read assemblers, viz. Velvet [8], SOAPdenovo [5], ABySS [7], Minia [2], IDBA-UD [6] and SPAdes [1]. In addition to the time and memory performance statistics, we also report quality metrics such as the N50 length which is similar to a median contig length, and the percentage of the genome covered.

As seen in Table 1, *FastEtch* and its variants report 50% savings with respect to time, and stand third with respect to memory usage, with Minia and IDBA-UD running in less memory. We observe that the overall N50 lengths among almost all assemblers are comparable, except for SPAdes which produces the highest N50 length, however is unable to produce an output for the Human (Chr2+3) dataset. Optimizing the code to generate more memory savings could improve the performance-quality trade-off achieved by FastEtch even further.

This work on *FastEtch* was awarded '*Best Student Paper*' award at the 2016 ACM-BCB (Bioinformatics, Computational Biology, and Health Informatics) conference.

2.2 Distributed-memory approach: *PaKman*

Continuing to address the scalability challenge, we target assemblies for larger complex genomes, with our distributed-memory based algorithm *PaKman*. The size and complexity of the large genomes presents a formidable challenge in terms of both computation power and inter-process communication. Thus the main steps in this assembly method involves: a) Reading the input: dividing the input read dataset effectively among the process. b) k-mer counting: identifying k-mers and binning them in a manner to collate the individual k-mer counts and construct a string graph (unlike the de Bruijn graph) comprising of fat nodes (or *macro nodes*). c) Contig generation: contigs are enumerated through an efficient traversal of these *macro nodes*. The algorithm for k-mer counting ensures that adjoining k-mers contributing to the same contig are grouped within the same partition in order to improve the overall performance by minimizing the amount of data-movement. The contig enumeration algorithm illustrates a lossless, fully exhaustive walk among the *macro nodes* in a deterministic manner in order to produce far longer contigs.

Table 1: Comparative evaluation of C.elegans (size=5.3GB) and Human Chr2+3 (size=51GB) assemblies generated by *FastEtch* and five other state-of-the-art assemblers for k=32.

C.elegans (cov=50x)	**Time (mins)**	**memory (GB)**	**N50 (bp)**	**% Genome covered**
FastEtch (τ=1k)	10.01	30.9	5,221	85.05
Bi-FastEtch (τ=6k)	12.21	26.2	4,280	81.2
SOAPdenovo	25.17	42.01	622	88.9
Velvet	24.56	47.1	5,008	90.6
Minia	37.23	7.4	5,925	86.08
IDBA-UD	23.48	8.9	6,200	86.3
SPAdes	365.48	45.5	13,553	91.2
Chr2+3 (cov=100x)	**Time (mins)**	**memory (GB)**	**N50 (bp)**	**% Genome covered**
FastEtch (τ=10k)	70.83	115.9	2,620	72.11
Bi-FastEtch (τ=30k)	74.55	99.6	2,146	68.11
SOAPdenovo	88.53	84.74	2,046	73.22
Velvet	130.46	121.33	2,697	76.92
Minia	186.22	17.25	2,751	76.57
IDBA-UD	116.25	43.98	2,837	77.28
SPAdes	out of memory			

PaKman is in its final stages of development and preliminary evaluation thus far suggests that our method shows an overall 2.3x improvement in runtime compared to the existing large scale implementation, HipMer [4] for the C.elegans dataset (cov=100x, size=11GB). Our objective is to continue our evaluation and present our findings for larger datasets, eg: the full human genome (3 Gbp) with a $>$ 500GB dataset.

REFERENCES

[1] Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A Gurevich, Mikhail Dvorkin, Alexander S Kulikov, Valery M Lesin, Sergey I Nikolenko, Son Pham, Andrey D Prjibelski, et al. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, 19(5):455–477, 2012.

[2] Rayan Chikhi and Guillaume Rizk. Space-efficient and exact de Bruijn graph representation based on a Bloom filter. *Algorithms for Molecular Biology*, 8(1):1, 2013.

[3] Graham Cormode and S Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[4] Evangelos Georganas, Aydın Buluç, Jarrod Chapman, Steven Hofmeyr, Chaitanya Aluru, Rob Egan, Leonid Oliker, Daniel Rokhsar, and Katherine Yelick. HipMer: an extreme-scale de novo genome assembler. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 14. ACM, 2015.

[5] Ruiqiang Li, Hongmei Zhu, Jue Ruan, et al. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research*, 20(2):265–272, 2010.

[6] Yu Peng, Henry Leung, Siu-Ming Yiu, and Francis YL Chin. IDBA-UD: a de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth. *Bioinformatics*, 28(11):1420–1428, 2012.

[7] Jared T Simpson, Kim Wong, Shaun D Jackman, et al. ABySS: a parallel assembler for short read sequence data. *Genome Research*, 19(6):1117–1123, 2009.

[8] Daniel R Zerbino and Ewan Birney. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18(5):821–829, 2008.