Lazer: A Memory-Efficient Framework for Large-Scale Genome Assembly

Sayan Goswami, Arghya Kusum Das, Richard Platania, Kisung Lee, Seung-Jong Park

Division of Computer Science & Engineering, Louisiana State University

LSU Center for Computation & Technology — Interdisciplinary | Innovative | Inventive

LSU College of Engineering, School of Electrical Engineering & Computer Science

ABSTRACT

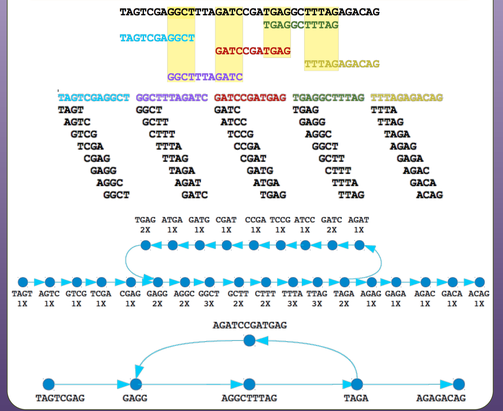
Genome sequencing technology has witnessed tremendous progress both in terms of throughput as well as the cost per base pair. However, when it comes to sequence assembly, there still exists a dilemma in the state-of-the-art technology. On one hand, we have a number of distributed assemblers that can utilize several nodes but require massive amounts of memory. On the other hand, there are a few assemblers that can assemble mammalian genomes on a single node but cannot scale up. In this work, we present a distributed assembler that is both scalable and memory efficient. Using partitioned *de Bruijn* graphs we enhance memory-to-disk swapping and reduce network communication in the cluster. Experimental results show that our framework can assemble a human genome dataset (452GB) in 14.5 hours using two nodes and 23 minutes using 128 nodes.

BACKGROUND

- **Genome sequencing**: The process of chemically parsing nucleotides in a DNA
- Current sequencing machines cannot sequence the entire genome at one go.
- Solution:
  - Replicate the genome many times over,
  - break it into shorter segments (called short-reads),
  - sequence the shorter segments at random.
- Rationale: A high probability that these shorter segments will overlap
- Find these overlaps and you get the original genome

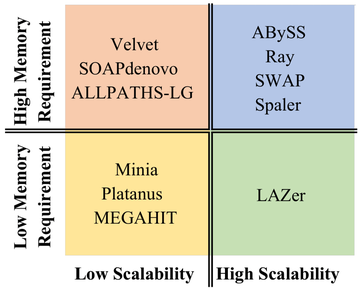
```
Original: ACCAATTCGAATATCTCTGTCATGAATTAAATCCTCATTGTGCAAAGATGCTCTG
Clone 1:  ACCAATTCGAATATCTCT   GTCATGAATTAAATCCTCA   TTGTGCAAAGATGCTCTG
Clone 2:  ACCAATTCGAA   TATCTCTGTCATGAAT   TAAATCCTCATTGTG   CAAAGATGCTCTG
```

- **De novo assembly**: reconstructing the genome without references.
- Break down short-reads into smaller overlapping sub-sequences of size *k* (*k*-mers).
- Build a *de Bruijn* graph using the following rules:
  - Each unique *k*-mer is represented as a vertex in the *de Bruijn* Graph.
  - An edge exists between two vertices if their corresponding *k*-mers have an overlap of *k* -1 characters between them.
- Compress the linear chains in the graph.



MOTIVATION

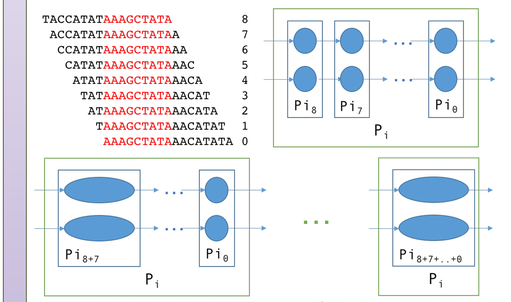
- Next Generation Sequencers:
  - High throughput,
  - High coverage,
  - Low cost
- Illumina HiSeq: 3 billion reads per run at 0.15 per million bases.
- Consequence: Genome assembly is becoming more and more demanding both in terms of memory and processing power.
- Problem: Most of the assemblers are either scalable, or memory efficient but not both.
- Examples:
  - Minia[2]: Uses bloom filters– memory efficient but not distributed.
  - ABySS[6], Ray[1]: Distributed but not scalable
  - SWAP[5]: fast and highly scalable, but enormous memory footprint.
- Our motivation: To build a distributed, scalable and memory-efficient assembler



METHODOLOGY

- Both scalability and memory efficiency can be achieved with partitioned graphs
  - Avoid loading entire graph into memory at once
  - Each partition can be processed independently – more parallelism
- We use minimum substring partitioning[4]
  - Slide a smaller window over the *k*-mers to obtain a set of substrings
  - A *k*-mer is hashed on the basis of the alphabetically smallest substring.

- To reduce the memory footprint even further we propose a 2-level partitioning scheme
  - The first level (L1) partition of a *k*-mer is determined by its minimum substring.
  - The second level (L2) is determined by the substring's position within the *k*-mer.
- This ensures that during an L1 compression, at most two L2 partitions have to be in memory at once



the size of the substring is 9 and that of k is 17, thus there are 4^9 L1 partitions and 9 L2 partitions per L1 partition. The next figure explains the local compression of an L1 partition. The compression begins with the vertices in L2 partition P_{i8} as seeds. The vertices in P_{i7} are loaded in a hash-map, merged with the seeds and removed from map. This process continues until the vertices of all 9 L2 partitions are merged.

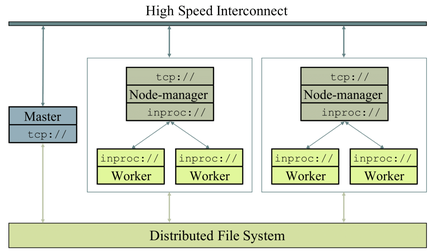
The assembly pipeline consists of three phases:

1. **Map:**
   1. read parts of the input from a distributed file system,
   2. parse them into *k*-mers, and
   3. map these *k*-mers into local partitions.
2. **Reduce:**
   1. Load Balancing: count all intermediate key-value pairs in all the machines
   2. Fairly distribute the L1 partitions to reducers (Longest Processing Time)
   3. Reducers merge different values for same keys and compress the chains that exist locally.
3. **Compression:**
   1. Load the vertices in a distributed hash-map
   2. Traverse the linear chains to produce *contigs*

IMPLEMENTATION

- Built on top of **ZeroMQ** (`http://zeromq.org/`)- an embeddable framework for concurrency and communication.
- Provides a threading library based on Hewitt's **Actor Model** [3].
- No global state – all actor interactions through atomic message passing.
- No locks, semaphores or critical sections – ideal for building distributed systems.
- Provides an intelligent transport layer for low latency transmission of small packets.
- Ideal for the fine-grained parallelism required in graph compression.
- Also provides an uniform message passing semantics between actors (`inproc`), processes (`ipc`) and boxes (`tcp`).

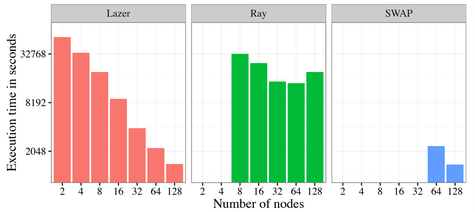
- The following figure shows the overview of the architecture used.
  - A single master process which is responsible for the distribution of tasks, synchronization between nodes and load-balancing.
  - Each node runs a slave process consisting of a Node-manager actor which spawns and manages worker actors.
  - All processes have access to a distributed file system and can send/receive messages to/from each other via a high speed network.



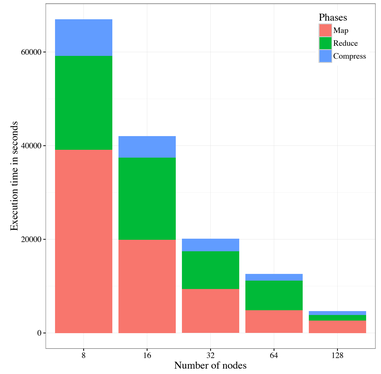
Since a manager and its workers reside in the same address space, they can connect to each other via `inproc` protocol. Message transmission over `inproc` is faster than both `tcp` and `ipc`, and involves less memory-copying since it allows passing pointers. The node-managers connect to the master via `tcp`.

EVALUATION

- Experimental test-bed: QueenBee II
  - Two 10-core 2.8 GHz E5-2680v2 Xeon processors
  - 64GB memory per node
  - 250 GB scratch space per node
  - Access to Lustre file system
  - 56 Gb/sec (FDR) InfiniBand 2:1 oversubscribed mesh
- Evaluated Lazer, Swap, and Ray using the Yoruban male dataset (Accession #SRA000271) with 452GB of raw reads
  - Tests on 2 nodes all the way up to 128 nodes
  - Observed near linear scalability



We were not able to run SWAP on 32 nodes due to out-of-memory errors. Furthermore, we could not run Ray on all the configurations due to cluster wall-clock limitations. We also observed that Ray's performance starts degrading beyond 64 nodes.



We also evaluated our assembler on hexaploid wheat genome (Synthetic W7984) having 1.1 TB of error corrected sequences.. The figure alongside reports the preliminary results along with the breakdown of the different phases.

We haven't been able to successfully run the other assemblers on this dataset, so we are unable to provide a comparative analysis yet.

CONCLUSIONS & FUTURE WORK

We introduced a distributed assembly framework that utilizes a smart partitioning scheme for *de Bruijn* graphs to achieve both scalability and memory efficiency.

Experimental results show that our framework significantly reduces the memory footprint while ensuring fast assembly.

In future, we plan to enhance our framework with *contig* extension and scaffolding to create a full pipeline.

References

[1] S. Boisvert, F. Laviolette, and J. Corbeil. Ray: simultaneous assembly of reads from a mix of high-throughput sequencing technologies. Journal of Computational Biology, 17(11), 2010.

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

[3] C. Hewitt, P. Bishop, and R. Steiger. A universal modular actor formalism for artificial intelligence. In IJCAI, 1973.

[4] Y. Li, P. Kamousi, F. Han, S. Yang, X. Yan, and S. Suri. Memory efficient minimum substring partitioning. In VLDB, volume 6, 2013.

[5] J. Meng, B. Wang, Y. Wei, S. Feng, and P. Balaji. SWAP-Assembler: scalable and efficient genome assembly towards thousands of cores. BMC bioinformatics, 15, 2014.

[6] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. Jones, and I. Birol. ABySS: a parallel assembler for short read sequence data. Genome research, 2009.

Acknowledgements

This work was supported in part by National Science Foundation under grants MRI-1338051 and CC-NIE-1341008, LA Board of Regents under grant LEQSF(2016-19)-RD-A-08, and IBM Faculty Award.