High-Performance Graph Traversal for De Bruijn Graph-Based Metagenome Assembly

Vasudevan Rengasamy   Kamesh Madduri

School of EECS
The Pennsylvania State University

`{vxr162, madduri}@psu.edu`

SIAM CSE 2017

Overview

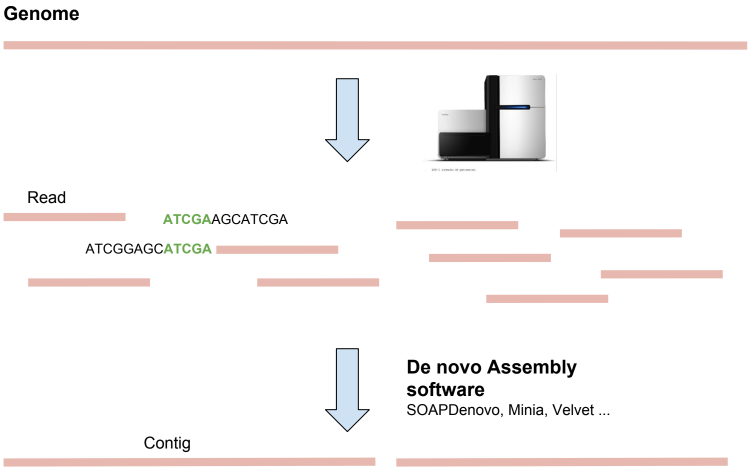
Introduction

METAPREP overview

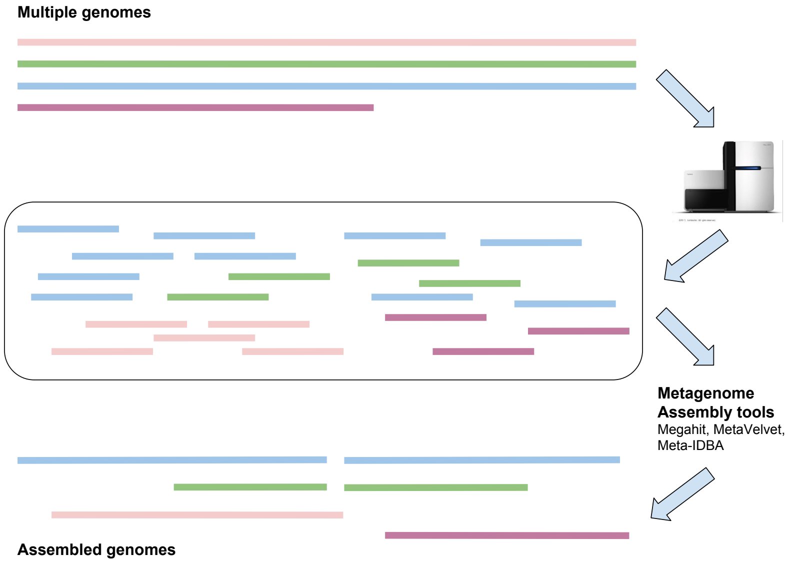
Experiments and Results
- Parallel Scaling
- Comparison to prior work
- Metagenome assembly results

Conclusions and Future work

What is de novo genome assembly?

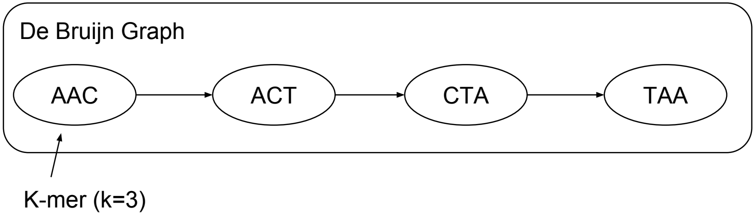


What is metagenome assembly?



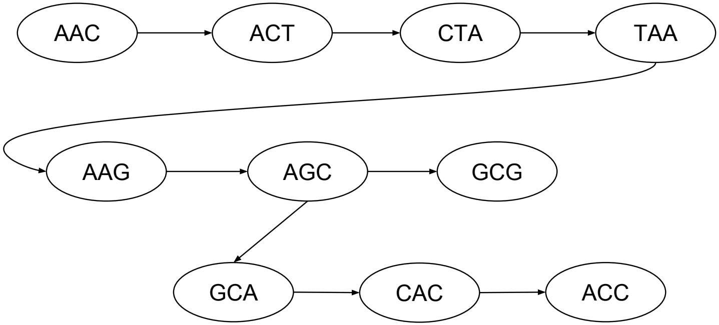
What is a de Bruijn graph?

Read: AACTAA



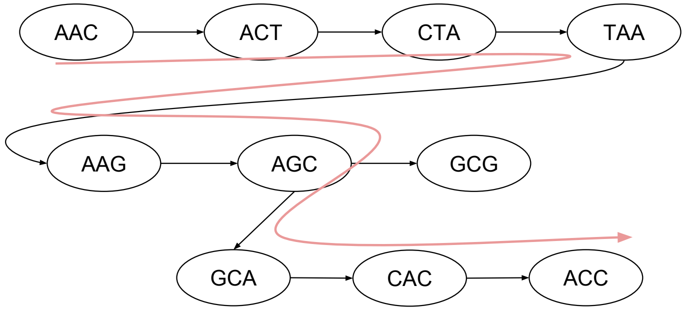
What is a de Bruijn graph?

Reads: AACTAA, TAAGCG, AGCACC



De Bruijn Graph (k=3)

How is it used in assembly?



De Bruijn Graph (k=3)

- Concatenate *k*-mer strings in each path.
- Assembled string: AACTAAGCACC.

Challenges in metagenome assembly

1. Uneven coverage of genomes.
2. Repeated sequences across genomes.
3. Variable sizes of genomes.
4. Large dataset sizes (as the output from multiple sequencing runs may be merged).

Metagenome assembly tools (MEGAHIT, MetaVelvet, metaSPAdes, etc.) attempt to overcome these challenges.

MEGAHIT [Li2016] metagenome assembler

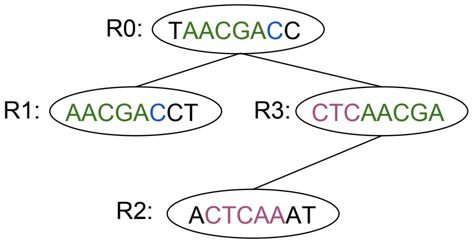
- State-of-the-art metagenome assembler.
- Uses a highly compressed de Bruijn graph representation.
- Refines assembly quality by using multiple *k*-mer lengths.
- Supports single-node shared memory parallelism (both CPUs and GPUs).

A preprocessing strategy for Metagenome assembly

1. After filtering low frequency *k*-mers, partition de Bruijn graph into weakly connected components (WCCs).
2. Assemble each large component independently.
3. Introduced by Howe et al. [Howe2014].

Recent work on metagenome partitioning [Flick2015]

- Construct an undirected **read graph** instead of a de Bruijn graph.
- Find connected components in the read graph using a distributed memory parallel approach based on Shiloach-Vishkin algorithm.
- Read graph components correspond to de Bruijn graph WCCs.



Motivation for our work

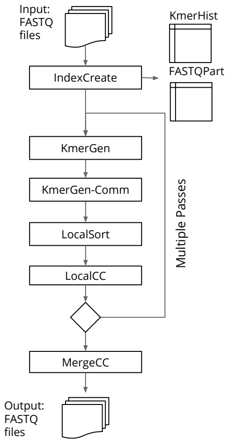
- Can we improve on [Flick2015] approach?
- How does read graph partitioning impact assembly?

Our approach

- New **Meta**genome **Prep**rocessing tool METAPREP.
- Main memory use is parameterized.
  - Multipass approach: Only enumerate a subset of *k*-mers in each pass.
  - e.g., 10 passes $\Rightarrow$ 10$\times$ memory reduction.
- Only one inter-node communication phase in our distributed memory connected components.

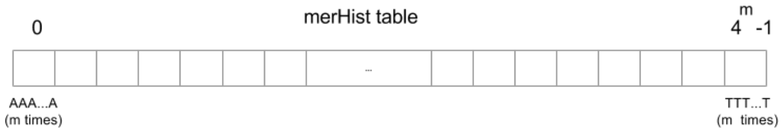
MetaPrep overview

	MetaPrep step	Function
	IndexCreate	Create index files for parallel runs.
1	KmerGen	Enumerate $\langle k\text{-mer}, \text{read}_i \rangle$ tuples.
2	KmerGen-Comm	Transfer $\langle k\text{-mer}, \text{read}_i \rangle$ tuples to owner tasks.
3	LocalSort	Sort tuples by k-mers.
4	LocalCC	Identify connected components (CCs).
5	MergeCC	Merge components across tasks, create output FASTQ file with reads from largest CC.



A simple strategy for static work partitioning

- Precompute an m-mer histogram ($m \ll k$, defaults are $k = 27$, $m = 10$)
- Used to partition k-mers across MPI tasks and threads in a load balanced manner.

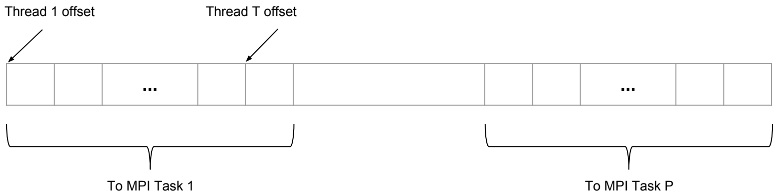


Notation

Notation	Description
M	Total number of k-mers enumerated
R	Paired-end read count
S	Number of I/O passes
P	Number of MPI tasks
T	Number of threads per task

k-mer Enumeration

- Generate $\langle k$-mer, read_id$\rangle$ tuples.
- Threads add *k*-mers to a common buffer. Offsets precomputed.
- Output: a buffer on each MPI task.
  - *k*-mers are partially sorted.
- Time: $O(\frac{MS}{PT})$, Space: $\frac{24M}{SP}$ bytes.



Send Buffer at MPI task i

Sort by *k*-mer

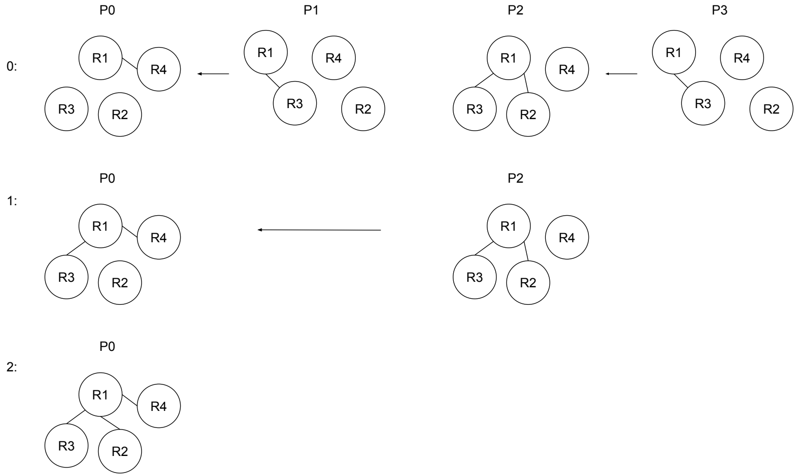
- Sort tuples by *k*-mer value to identify reads with common *k*-mer and create read graph edges.
- Radix sort implementation.
  - Reuse send buffer $\Rightarrow$ No additional memory .
  - Partition tuples into T disjoint ranges.
  - Sort ranges in parallel using T threads.
- Time: $O(\frac{M}{PT})$, Space: $\frac{24M}{SP}$ bytes.

Identify connected components

- Find connected components using edges from local *k*-mers.
- Union-by-index and path splitting.
- No critical sections.
  - Store edges that merges components (similar to [Patwary2012]).
  - Process edges again in case of lost updates.
- Time: $O(\frac{M}{PT} log^* R)$, Space: $\frac{12M}{SP} + 4R$ bytes.

Merge components

- Merge component forests in each MPI task in *log P* iterations.
- Time: $O(R \log P \log^* R)$, Space: $8R$ bytes.



Experiments and Results

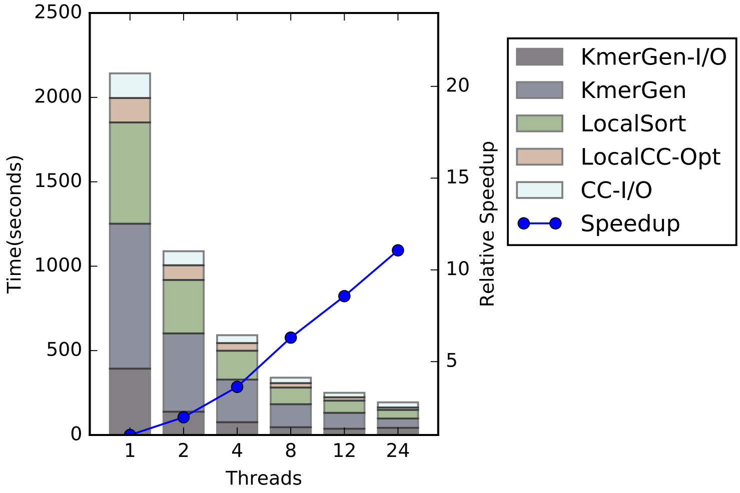
Description of datasets

Dataset	Read Count R ($\times 10^6$)	Size (Gbp)	Source
Human metagenome	67.6	6.75	NCBI (SRR1804155)
Iowa, Continuous corn soil	1132.8	112.0	JGI (402461)

Machine configuration

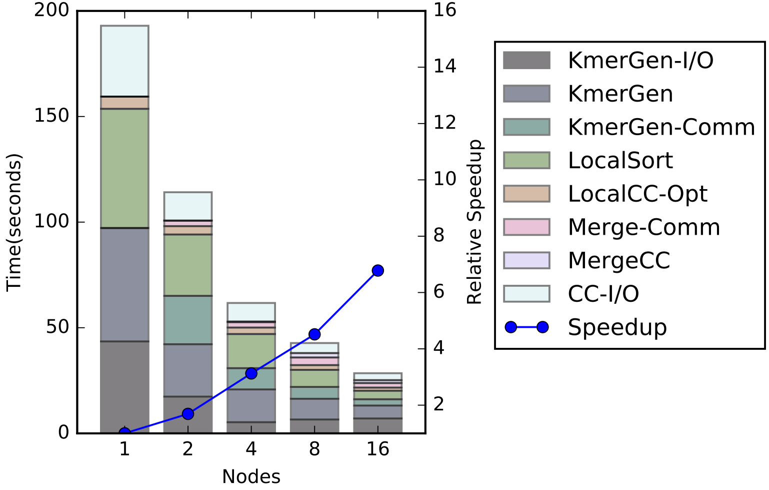
- Edison supercomputer at NERSC.
- Each node has 2$\times$ 12-core Ivy bridge processors and 64 GB DDR3 memory.

Single node scaling for Human Metagenome Dataset

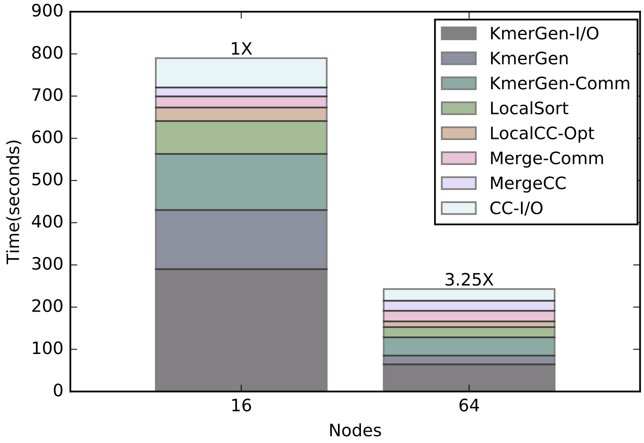


Execution using 4 I/O passes.

Multi-node scaling for Human Metagenome Dataset



Multi-node scaling for Iowa Continuous Soil Dataset



For 16 node run, Number of passes = 8. For 64 node run, Number of passes = 2.

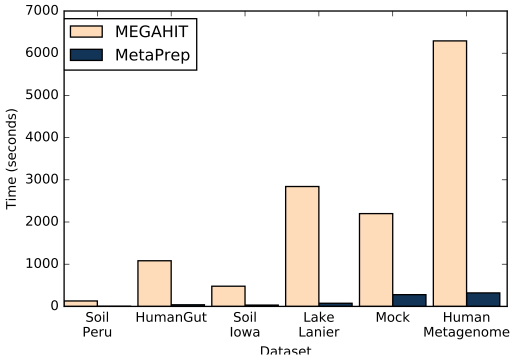
Comparison with read graph connectivity [Flick2015]

Table 1: Comparison for Human metagenome. AP_LB (Active Partitions with Load balancing) denotes read graph connectivity work [Flick2015]

Step	MetaPrep (seconds)	AP_LB (seconds)
Communication	5.1	24.6
Sort tuples	4.5	40.4
Read graph partitioning	7.0	
Total	16.6	69.0

- 4.15$\times$ speedup over AP_LB on 16 nodes.
- 21 CC iterations for AP_LB vs 4 for MetaPrep.

MEGAHIT [Li2016] assembly time vs METAPREP time



- Preprocessing time (MetaPrep) $\ll$ Assembly time.
- Many metagenome datasets have a single giant component.

Conclusions

1. Developed a new memory efficient parallel workflow for partitioning metagenome dataset into connected components.
2. Speedup of $4.15\times$ over AP_LB approach by [Flick2015].
3. We can process a metagenome dataset with 1.13 billion reads (Iowa continuous corn soil) in 14 minutes using 16 nodes of Edison.
4. Preprocessing time (MetaPrep) $\ll$ Assembly time.

Future Work

1. For most datasets, we observe creation of a single large connected component after partitioning the read graph.
   - Can we split this giant component using *k*-mer frequency information in a principled manner?
2. Reduce data exchanged in the inter-node communication step of connected components.

Acknowledgment

This research is supported in part by NSF award #1439057. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References I

Patrick Flick, Chirag Jain, Tony Pan, and Srinivas Aluru.
A parallel connectivity algorithm for de Bruijn graphs in metagenomic applications.
In *Proc. Int'l. Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*, 2015.

Adina Chuang Howe, Janet K. Jansson, Stephanie A. Malfatti, Susannah G. Tringe, James M. Tiedje, and C. Titus Brown.
Tackling soil diversity with the assembly of large, complex metagenomes.
Proceedings of the National Academy of Sciences, 111(13):4904–4909, 2014.

References II

Dinghua Li, Ruibang Luo, Chi-Man Liu, Chi-Ming Leung, Hing-Fung Ting, Kunihiko Sadakane, Hiroshi Yamashita, and Tak-Wah Lam.
MEGAHIT v1.0: A fast and scalable metagenome assembler driven by advanced methodologies and community practices.
Methods, 102:3–11, 2016.

Md Mostofa Ali Patwary, Peder Refsnes, and Fredrik Manne.
Multi-core spanning forest algorithms using the disjoint-set data structure.
In *Proc. IEEE Int'l. Parallel & Distributed Processing Symposium (IPDPS)*, 2012.

Thank You