CS267 Assignment 3:
Parallelize Graph Algorithms for de Novo Genome Assembly

Spring 2016

Problem statement
- **Input**: A set of unique k-mers and their corresponding extensions.
- k-mers are sequences of length k (alphabet is A/C/G/T).
- An extension is a simple symbol (A/C/G/T/F).
- The input k-mers form a de Bruijn graph, a special graph that is used to represent overlaps between sequences of symbols.

- **Output**: A set of contigs, i.e., connected components in the input de Bruijn graph.

Example
- **Input**: A set of unique k-mers and their corresponding extensions.
- Example for k = 3:
  - Format: k-mer forward extension, backward extension
  - AAC CF
  - ATC TG
  - ACC GA
  - TGA FC
  - GAT CF
  - AAT GF
  - ATG CA
  - TCT GA
  - CCG FA
  - CTG AT
  - TGC FA

- **Input**: A set of unique k-mers and their corresponding extensions.
- The input corresponds to a de Bruijn graph.
- Example for k = 3:
Example

- **Input:** A set of unique k-mers and their corresponding extensions.
- The input corresponds to a de Bruijn graph.
- Example for k = 3:

\[
\begin{align*}
\text{AAC} & \quad \text{CF} \\
\text{ATC} & \quad \text{TG} \\
\text{ACC} & \quad \text{GA} \\
\text{TGA} & \quad \text{FC} \\
\text{GAT} & \quad \text{CF} \\
\text{AAT} & \quad \text{GF} \\
\text{ATG} & \quad \text{CA} \\
\text{TCT} & \quad \text{GA} \\
\text{CCG} & \quad \text{FA} \\
\text{CTG} & \quad \text{AT} \\
\text{TGC} & \quad \text{FA}
\end{align*}
\]

- **Output:** A set of contigs or equivalently the connected components in the de Bruijn graph

k-mers with "F" as an extension are start vertices

Example

- **Input:** A set of unique k-mers and their corresponding extensions.
- The input corresponds to a de Bruijn graph.
- Example for k = 3:

\[
\begin{align*}
\text{AAC} & \quad \text{CF} \\
\text{ATC} & \quad \text{TG} \\
\text{ACC} & \quad \text{GA} \\
\text{TGA} & \quad \text{FC} \\
\text{GAT} & \quad \text{CF} \\
\text{AAT} & \quad \text{GF} \\
\text{ATG} & \quad \text{CA} \\
\text{TCT} & \quad \text{GA} \\
\text{CCG} & \quad \text{FA} \\
\text{CTG} & \quad \text{AT} \\
\text{TGC} & \quad \text{FA}
\end{align*}
\]

- Consider k-mer: TCT
  - Concatenate last k-1 bases (CT) and forward extension (G) => CTG (following vertex)
  - Concatenate backward extension (A) and first k-1 bases (TC) => ATC (preceding vertex)

The graph is undirected, we can visit a vertex from both directions.

Compact graph representation: hash table

- The vertices are keys
- The edges (neighboring vertices) are represented with a two-letter value

\[
\begin{align*}
\text{AAT} & \quad \text{CF} \\
\text{ATC} & \quad \text{TG} \\
\text{ACC} & \quad \text{GA} \\
\text{TGA} & \quad \text{FC} \\
\text{GAT} & \quad \text{CF} \\
\text{AAT} & \quad \text{GF} \\
\text{ATG} & \quad \text{CA} \\
\text{TCT} & \quad \text{GA} \\
\text{CCG} & \quad \text{FA} \\
\text{CTG} & \quad \text{AT} \\
\text{TGC} & \quad \text{FA}
\end{align*}
\]
We pick a start vertex and we initiate a contig.

Contig: A A T

Contig: A A T G
Graph traversal

• We take the last k bases of the contig and look them up in the hash table.

Contig: AA T G C

AAC  CF
ATC  TG
ACC  GA
TGA  FC
GAT  CF
AAT  GF
ATG  CA
TCT  GA
CCG  FA
CTG  AT
TGC  FA

• We add the new forward extension to the contig.

Contig: A A T G C

AAC  CF
ATC  TG
ACC  GA
TGA  FC
GAT  CF
AAT  GF
ATG  CA
TCT  GA
CCG  FA
CTG  AT
TGC  FA

• We take the last k bases of the contig and look them up in the hash table.

Contig: A A T G C

AAC  CF
ATC  TG
ACC  GA
TGA  FC
GAT  CF
AAT  GF
ATG  CA
TCT  GA
CCG  FA
CTG  AT
TGC  FA

• We terminate the current contig since the forward extension is an “F”.

Contig: A A F G C
Graph traversal

- We iterate until we exhaust all start vertices; we have found all the contigs.

Parallelization hints

1. Distribute the hash table among the processors.
   - UPC is convenient: Store the hash table in the shared address space.
   - You may want to use upc_all_alloc().

2. Each processor stores part of the input in the distributed hash table.
   - What happens if two processors try to write the same bucket at the same time?
   - We need to avoid race conditions (UPC provides locks and global atomics).

3. We want to traverse the graph in parallel.
   - Can we determine independent traversals by examining the input?
   - How can we distribute the work among processors?