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

Spring 2015

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

<table>
<thead>
<tr>
<th>k-mer</th>
<th>forward extension</th>
<th>backward extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAC</td>
<td>CF</td>
<td></td>
</tr>
<tr>
<td>ATC</td>
<td>TG</td>
<td></td>
</tr>
<tr>
<td>ACC</td>
<td>GA</td>
<td></td>
</tr>
<tr>
<td>TGA</td>
<td>FC</td>
<td></td>
</tr>
<tr>
<td>GAT</td>
<td>CF</td>
<td></td>
</tr>
<tr>
<td>AAT</td>
<td>GF</td>
<td></td>
</tr>
<tr>
<td>ATG</td>
<td>CA</td>
<td></td>
</tr>
<tr>
<td>TCT</td>
<td>GA</td>
<td></td>
</tr>
<tr>
<td>CCG</td>
<td>FA</td>
<td></td>
</tr>
<tr>
<td>CTG</td>
<td>AT</td>
<td></td>
</tr>
<tr>
<td>TGC</td>
<td>FA</td>
<td></td>
</tr>
</tbody>
</table>

Example

• **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** $k = 3$:

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

Compact graph representation: hash table

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

<table>
<thead>
<tr>
<th>Buckets</th>
<th>Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>key: AAC</td>
<td>TGC, back_ext: C, forw_ext: F</td>
</tr>
<tr>
<td>key: ATC</td>
<td>TG, back_ext: C, forw_ext: F</td>
</tr>
<tr>
<td>key: ACC</td>
<td>GA, back_ext: A, forw_ext: G</td>
</tr>
<tr>
<td>key: GAT</td>
<td>CF, back_ext: C, forw_ext: F</td>
</tr>
<tr>
<td>key: AAT</td>
<td>GF, back_ext: C, forw_ext: F</td>
</tr>
<tr>
<td>key: ATG</td>
<td>CA, back_ext: C, forw_ext: F</td>
</tr>
<tr>
<td>key: TCT</td>
<td>GA, back_ext: C, forw_ext: F</td>
</tr>
<tr>
<td>key: CCT</td>
<td>FA, back_ext: C, forw_ext: F</td>
</tr>
<tr>
<td>key: ATC</td>
<td>TG, back_ext: C, forw_ext: F</td>
</tr>
<tr>
<td>key: ACC</td>
<td>GA, back_ext: A, forw_ext: G</td>
</tr>
<tr>
<td>key: GAT</td>
<td>CF, back_ext: C, forw_ext: F</td>
</tr>
<tr>
<td>key: AAT</td>
<td>GF, back_ext: C, forw_ext: F</td>
</tr>
<tr>
<td>key: ATG</td>
<td>CA, back_ext: C, forw_ext: F</td>
</tr>
<tr>
<td>key: TCT</td>
<td>GA, back_ext: C, forw_ext: F</td>
</tr>
<tr>
<td>key: CCT</td>
<td>FA, back_ext: C, forw_ext: F</td>
</tr>
<tr>
<td>key: AAC</td>
<td>CF, back_ext: A, forw_ext: G</td>
</tr>
<tr>
<td>key: TGC</td>
<td>FA, back_ext: A, forw_ext: G</td>
</tr>
<tr>
<td>key: ATG</td>
<td>CA, back_ext: A, forw_ext: G</td>
</tr>
<tr>
<td>key: AAT</td>
<td>GF, back_ext: A, forw_ext: G</td>
</tr>
<tr>
<td>key: CTG</td>
<td>AT, back_ext: A, forw_ext: G</td>
</tr>
<tr>
<td>key: TGC</td>
<td>FA, back_ext: A, forw_ext: G</td>
</tr>
</tbody>
</table>
We pick a start vertex and we initiate a contig.

**Serial algorithm**

```
Algorithm 1 De Bruijn Graph Construction And Traversal

1: Input: A set of k-mers and their corresponding forward and backward extensions
2: Output: A set of contigs
3: /* Initialization */
4: hashTable = CREATEHashTable();
5: startNodesList = CREATEEmptyList();
6: /* De Bruijn Graph Construction */
7: for each (k-mer, forwardExt, backwardExt) in input do
8:   ADDkmerToHashTable(hashTable, (k-mer, forwardExt, backwardExt))
9: if (backwardExt) then
10:   ADDkmerToList(startNodesList, (k-mer, forwardExt))
11: end if
12: end for
13: /* De Bruijn Graph Traversal */
14: for each (k-mer, forwardExt) in startNodesList do
15:   currentContig = CREATENewSequence(k-mer)
16:   currentForwardExtension = forwardExt
17:   while (currentForwardExtension is not F) do
18:     currentKmer = LASTkmer(currentContig)
19:     currentForwardExtension = LOOKUP(hashTable, currentKmer)
20:     end while
21:   STOREContig(currentContig)
22: end for
```

**Graph construction**

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

```
AAT
```

```
entries

back_ext: F
forw_ext: A
```

- We add the forward extension to the contig.

**Graph traversal**

```
Contig: A A T
```

```
CTG: A
TCT: G
```

```
Contig: A A T G
```

```
CTG: A
TCT: G
```

```
AAT
```
Graph traversal

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

Contig: AA T G C

- We add the new forward extension to the contig.

Contig: A A T G C

Graph traversal

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

Contig: A A T G C

- 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?