Kcollections

M. Stanley Fujimoto
Cole A. Lyman
Mark J. Clement
Brigham Young University

HICOMB2020

Why

- Many bioinformatic algorithms are based on k-mers
- Prototyping new algorithms based on new algorithms can be difficult because:
  - The number of possible k-mers grows exponentially as k increases
  - Storing k-mers for even moderately sized k becomes impossible on desktop hardware

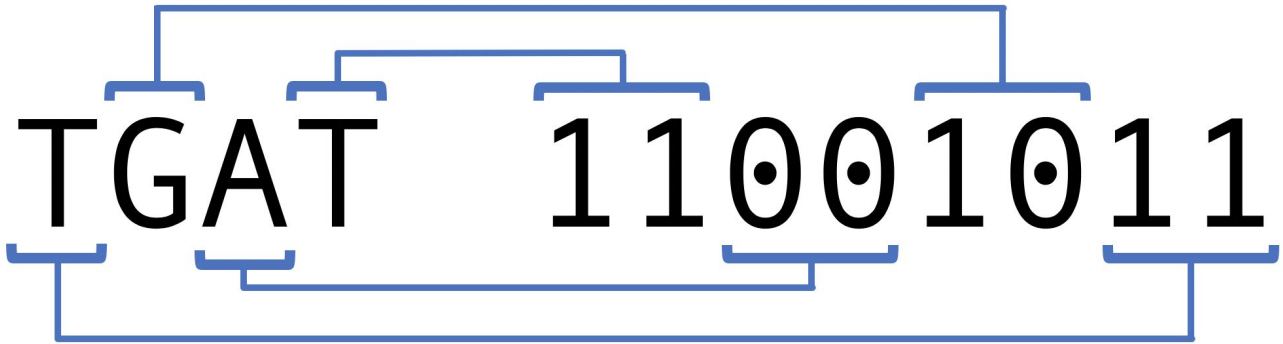
We propose an efficient and fast method for storing k-mers, kcollections, for broad bioinformatic applications

How

- Take advantage of common k-mer serialization techniques to:
  - Store k-mers in an efficient data structure (burst trie)
  - Parallelize insert and look-up operations

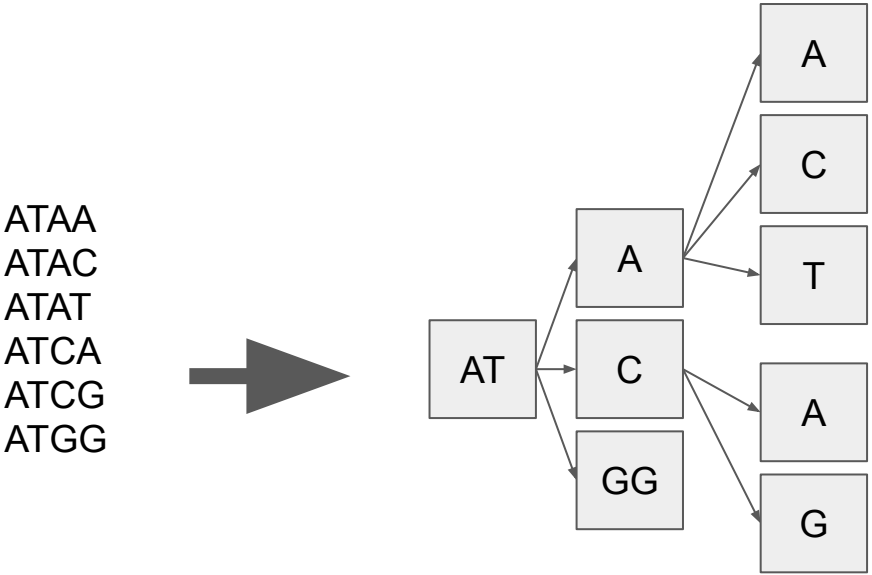
How - Serialization

K-mers are commonly bit-packed using only 2 bits per base for efficient storage.
We exploit the compact, serialized k-mers for further storage and speed efficiency.



How - Efficient Storage, Trie

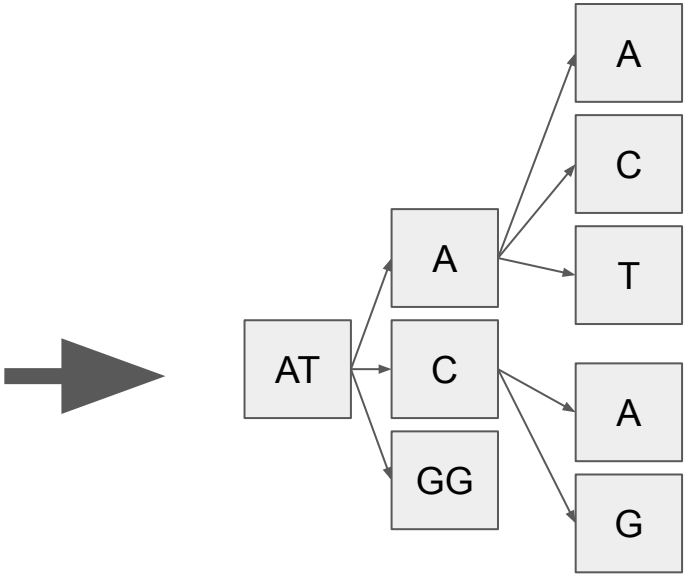
Shared prefixes amongst k-mers are redundant. Remove redundant information by storking k-mers in a trie.



How - Efficient Storage, Burst Trie

Use a burst trie to manage/minimize the creation of new children vertices. Children vertices are stored in a condensed array.

ATAA
ATAC
ATAT
ATCA
ATCG
ATGG

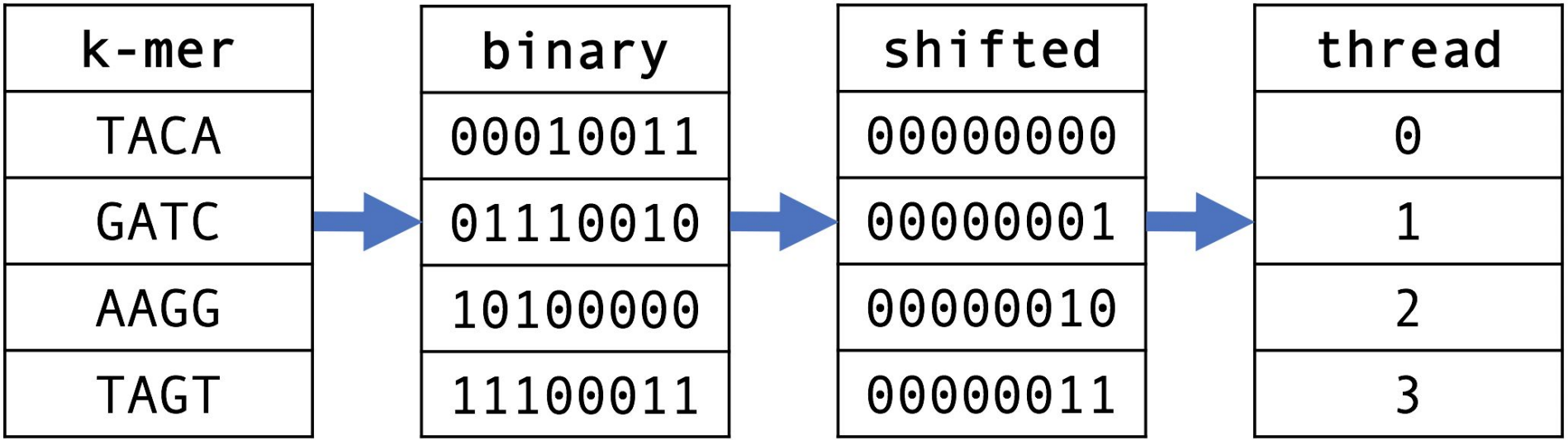


Children Vertex Array

idx	k-mer	Binary	int
0	CAAA	00000001	1
1	GAAA	00000010	2
2	CCAA	00000101	5
3	GTAA	00001110	14
4	GACA	00010010	18
5	AGCA	00011000	24
6	CGCA	00011001	25
7	GGCA	00011010	26
8	TGCA	00011011	27
9	AAGA	00100000	32
…	…	…	…

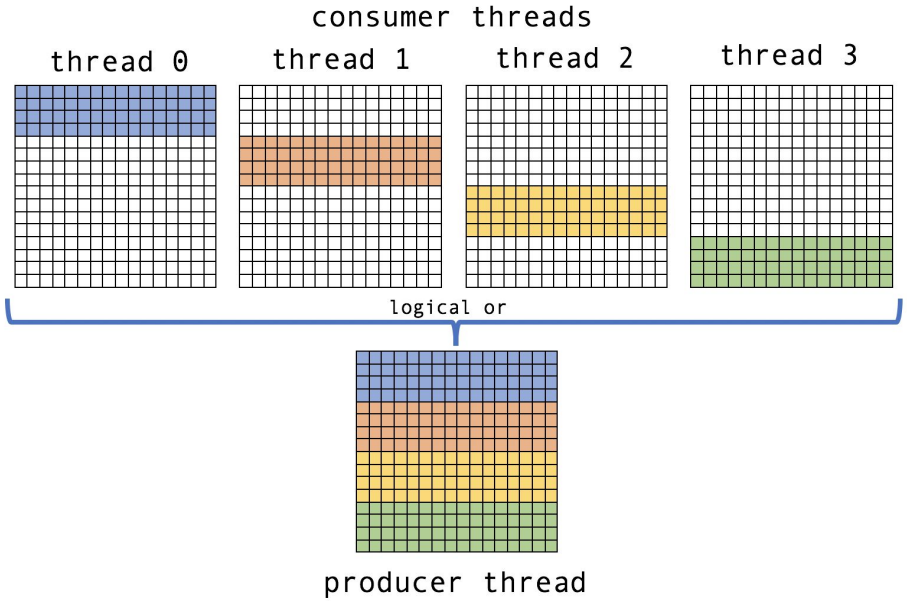
How - Parallelization, Map

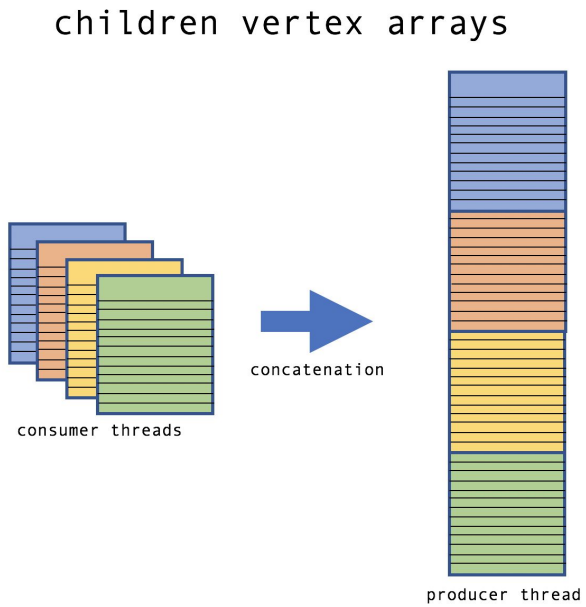
Multi-threaded insert is done by mapping incoming k-mers to appropriate threads which are responsible for a partition of the trie. Bit shifting quickly identifies the appropriate partition/thread a k-mer should be sent to.



How - Parallelization, Reduce

Merging partitions is simple: use bitwise operation to merge housekeeping variables and concatenate children vertices from each partition.





How - Parallelization, Look-Ups

Look-ups are thread-safe.

How - Parallelization, Look-Ups

1. Serialize k-mer query: `AAGA -> 00100000`

Children Vertex Array

idx	k-mer	Binary	int
0	CAAA	00000001	1
1	GAAA	00000010	2
2	CCAA	00000101	5
3	GTAA	00001110	14
4	GACA	00010010	18
5	AGCA	00011000	24
6	CGCA	00011001	25
7	GGCA	00011010	26
8	TGCA	00011011	27
9	AAGA	00100000	32
…	…	…	…

Presence Array

0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	1	1	1	0	0	0	0
1	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	1	1	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	1	0	0	1	0	0	1	0
0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0
0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0
0	0	0	0	1	0	0	0	1	0	0	0	1	1	1	0
1	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0
0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0
0	0	0	0	0	0	0	1	1	0	1	0	0	1	0	0
0	0	0	0	0	1	0	1	0	0	0	0	1	1	0	0
0	1	1	0	0	0	0	1	0	0	1	0	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
0	0	0	1	0	0	1	1	1	0	0	0	0	0	1	0

How - Parallelization, Look-Ups

1. Serialize k-mer query: `AAGA -> 00100000`
2. Convert serialized k-mer to int: `00100000 -> 32`

Children Vertex Array

idx	k-mer	Binary	int
0	CAAA	00000001	1
1	GAAA	00000010	2
2	CCAA	00000101	5
3	GTAA	00001110	14
4	GACA	00010010	18
5	AGCA	00011000	24
6	CGCA	00011001	25
7	GGCA	00011010	26
8	TGCA	00011011	27
9	AAGA	00100000	32
…	…	…	…

Presence Array

0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	1	1	1	0	0	0	0
1	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	1	1	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	1	0	0	1	0	0	1	0
0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0
0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0
0	0	0	0	1	0	0	0	1	0	0	0	1	1	1	0
1	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0
0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0
0	0	0	0	0	0	0	1	1	0	1	0	0	1	0	0
0	0	0	0	0	1	0	1	0	0	0	0	1	1	0	0
0	1	1	0	0	0	0	1	0	0	1	0	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
0	0	0	1	0	0	1	1	1	0	0	0	0	0	1	0

How - Parallelization, Look-Ups

1. Serialize k-mer query: `AAGA -> 00100000`
2. Convert serialized k-mer to int: `00100000 -> 32`
3. Check presence array if pos 32 bit is set

Children Vertex Array

idx	k-mer	Binary	int
0	CAAA	00000001	1
1	GAAA	00000010	2
2	CCAA	00000101	5
3	GTAA	00001110	14
4	GACA	00010010	18
5	AGCA	00011000	24
6	CGCA	00011001	25
7	GGCA	00011010	26
8	TGCA	00011011	27
9	AAGA	00100000	32
…	…	…	…

Presence Array

0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	1	1	1	0	0	0	0
1	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	1	1	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	1	0	0	1	0	0	1	0
0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0
0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0
0	0	0	0	1	0	0	0	1	0	0	0	1	1	1	0
1	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0
0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0
0	0	0	0	0	0	0	1	1	0	1	0	0	1	0	0
0	0	0	0	0	1	0	1	0	0	0	0	1	1	0	0
0	1	1	0	0	0	0	1	0	0	1	0	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
0	0	0	1	0	0	1	1	1	0	0	0	0	0	1	0

How - Parallelization, Look-Ups

1. Serialize k-mer query: `AAGA -> 00100000`
2. Convert serialized k-mer to int: `00100000 -> 32`
3. Check presence array if pos 32 bit is set
4. Bitshift array

Presence Array

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	1	1	1	0	0	0	0

Children Vertex Array

idx	k-mer	Binary	int
0	CAAA	00000001	1
1	GAAA	00000010	2
2	CCAA	00000101	5
3	GTAA	00001110	14
4	GACA	00010010	18
5	AGCA	00011000	24
6	CGCA	00011001	25
7	GGCA	00011010	26
8	TGCA	00011011	27
9	AAGA	00100000	32
…	…	…	…

How - Parallelization, Look-Ups

1. Serialize k-mer query: `AAGA -> 00100000`
2. Convert serialized k-mer to int: `00100000 -> 32`
3. Check presence array if pos 32 bit is set
4. Bitshift array
5. Popcount of array: 9

Children Vertex Array

idx	k-mer	Binary	int
0	CAAA	00000001	1
1	GAAA	00000010	2
2	CCAA	00000101	5
3	GTAA	00001110	14
4	GACA	00010010	18
5	AGCA	00011000	24
6	CGCA	00011001	25
7	GGCA	00011010	26
8	TGCA	00011011	27
9	AAGA	00100000	32
…	…	…	…

Presence Array

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	1	1	1	0	0	0	0

How - Parallelization, Look-Ups

1. Serialize k-mer query: `AAGA -> 00100000`
2. Convert serialized k-mer to int: `00100000 -> 32`
3. Check presence array if pos 32 bit is set
4. Bitshift array
5. Popcount of array: 9
6. Retrieve item at index 9 in children vertex array

Children Vertex Array

idx	k-mer	Binary	int
0	CAAA	00000001	1
1	GAAA	00000010	2
2	CCAA	00000101	5
3	GTAA	00001110	14
4	GACA	00010010	18
5	AGCA	00011000	24
6	CGCA	00011001	25
7	GGCA	00011010	26
8	TGCA	00011011	27
9	AAGA	00100000	32
…	…	…	…

Presence Array

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	1	1	1	0	0	0	0

What - Performance Comparison

Program	Subprogram	Elapsed Wall Clock Time (hh:mm:ss)	Maximum RAM Used (GB)	Threads
SSBT	SSBT hashes	00:00:00	0.009	1
	SSBT count	00:30:21	30.144	16
	SSBT build	00:00:00	0.009	1
	SSBT compress	00:01:05	1.855	1
	total	00:31:26	30.144	-
BFT		03:53:38	**16.281**	1
Python Set		02:00:56 (22:20:24)	261.527	1
kcollections		**00:27:11**	24.105	16

TABLE I: Time and memory usage for indexing the human genome. The BFT requires k-mer generation in a pre-process step, we use jellyfish and include those results. Both the build time and overall running time are provided for the Python set due to the large disparity between them.

What - Performance Comparison

Program	Elapsed Wall Clock Time (mm:ss)	Maximum RAM Used (GB)
BFT	01:25	16.166
Python Set	**01:00**	261.676
SSBT	11:17	**5.396**
kcollections	01:29	22.840

TABLE II: Time and memory usage for 20M queries against the human reference genome. 10M k-mers that exist and 10M that do not exist in the index.

Acknowledgements

- Dr. Mark J. Clement
- Cole A. Lyman
- BYU Computational Sciences Laboratory