Functional Hash Maps in a Data Parallel Language

William Henrich Due [1] Martin Elsman [1] Troels Henriksen [1]

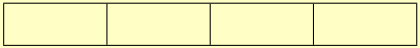
[1]Department of Computer Science, University of Copenhagen

August 22nd, 2025

Contact: widu@di.ku.dk

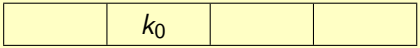
Open Adressing Example

- Keys $k_0, k_1 \in K$.
- Hash function $h : K \to \{0, 1, 2, 3\}$.
- $h(k_0) = h(k_1) = 1$.



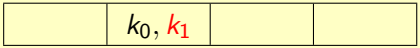
Open Adressing Example

- Keys $k_0, k_1 \in K$.
- Hash function $h : K \to \{0, 1, 2, 3\}$.
- $h(k_0) = h(k_1) = 1$.



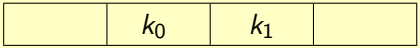
Open Adressing Example

- Keys $k_0, k_1 \in K$.
- Hash function $h : K \to \{0, 1, 2, 3\}$.
- $h(k_0) = h(k_1) = 1$.



Open Adressing Example

- Keys $k_0, k_1 \in K$.
- Hash function $h : K \to \{0, 1, 2, 3\}$.
- $h(k_0) = h(k_1) = 1$.

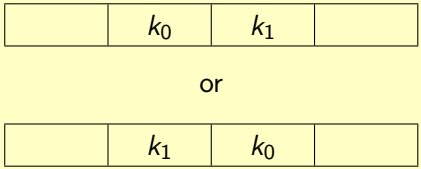


Core Ideas

- Concurrency.

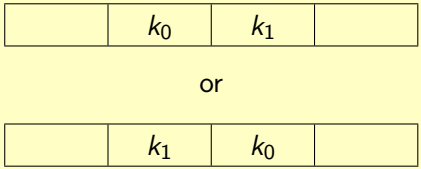
Core Ideas

- Concurrency.
- Collision resolution.



Core Ideas

- Concurrency.
- Collision resolution.
- Functional Array Languages.



$$\texttt{map} : (\alpha \to \beta) \to [n]\alpha \to [n]\beta$$

Perfect Hashing with FKS

- Find a collision-free hash function.
- $\{k_0, k_1, k_2\} \subseteq K$.
- Pick some $h \in H$ from a universal hash family.

$h(k_0)$	$h(k_1)$	$h(k_2)$
0	0	1

Perfect Hashing with FKS

- Find a collision-free hash function.
- $\{k_0, k_1, k_2\} \subseteq K$.
- Pick some $h \in H$ from a universal hash family.
- Pick perfect hash functions $h_0, h_1, h_2 \in H$.

$h(k_0)$	$h(k_1)$	$h(k_2)$
0	0	1

Bin Size	2	1	0
Squared Bin Size	4	1	0
Offset (o_i)	0	4	5
Hash Function	h_0	h_1	h_2

Perfect Hashing with FKS

- Find a collision-free hash function.
- $\{k_0, k_1, k_2\} \subseteq K$.
- Pick some $h \in H$ from a universal hash family.
- Pick perfect hash functions $h_0, h_1, h_2 \in H$.

$h(k_0)$	$h(k_1)$	$h(k_2)$
0	0	1

Bin Size	2	1	0
Squared Bin Size	4	1	0
Offset (o_i)	0	4	5
Hash Function	h_0	h_1	h_2

	$o_0 + h_0(k_0)$		$o_0 + h_0(k_1)$	$o_1 + h_1(k_2)$
	k_0		k_1	k_2

Comparison

	FKS	Open Addressing
Hashing	Universal Hash Family	Any[1]
Lookup	$O(1)$	Expected $O(1)$
Construction	Expected $O(n)$	$O(n)$
Dynamic	Yes[2]	Yes
Duplicate Keys	No	Yes

[1] May ruin the time complexity.
[2] Seems impractical.

Benchmarks

	64-bit integer keys ($n = 10^7$)		
	Construction	*Lookup*	*Membership*
Futhark (hash maps)	18.3	3.3	1.6
Futhark (binary search)	40.9	6.2	5.8
Futhark (Eytzinger)	42.3	4.3	2.4
cuCollections	2.7	1.1	0.9

All times in milliseconds.

Benchmarks

	String keys ($n = 10^7$)		
	Construction	*Lookup*	*Membership*
Futhark (hash maps)	33.2	4.3	2.8
Futhark (binary search)	83.0	5.7	5.8
Futhark (Eytzinger)	85.3	5.3	5.3
cuCollections	2.7	1.3	1.2

All times in milliseconds.