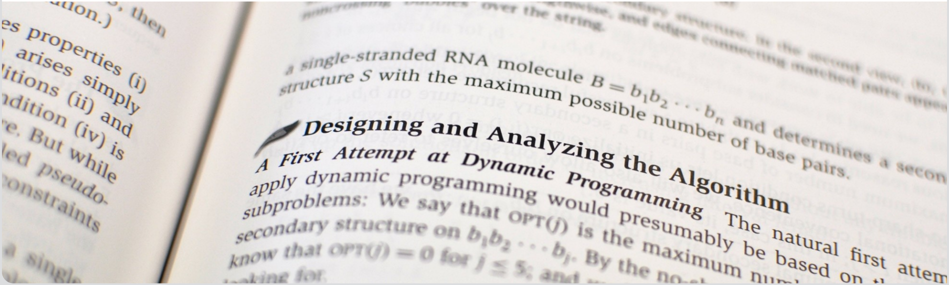
Retrieval and Perfect Hashing using Fingerprinting

Ingo Müller[1,2], Peter Sanders[1], Robert Schulze[2], and Wei Zhou[1,2]

Department of Informatics of KIT[1] and HANA Core of SAP AG[2]



The Retrieval Problem

Perfect Hash Function (PHF)

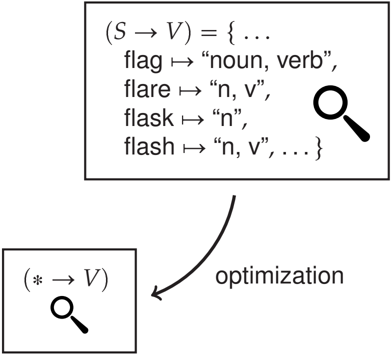
- Map each key $s \in S$ to unique integer $i \in ID$

Retrieval data structure

- Associate value $v \in V$ to each $s \in S$

Classical implementation: **hash table**

- Store key/ID or key/value pairs



Optimization: do not store S

- Undefined behavior for $s \notin S$

Applications

- Look-up in dictionaries of in-memory DBMSs (like the SAP HANA database [1])
- Many more. . . (see Botelho et al. [2])

Known Methods

Perfect Hash Functions

- Practical implementations exist: BPZ [3], CHD [4], etc.
- Store only constant, sometimes optimal number of extra bits
- Retrieval: use a PHF to index an array of values

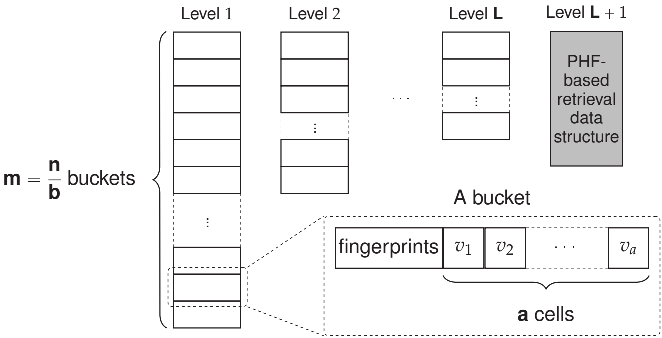
Direct Retrieval Data Structures

- CHM [5], etc.

	Prior work	**Our solution**
Construction	complicated inherently sequential $\Rightarrow$ slow	simpler easily parallelizable $\Rightarrow$ faster
Dynamic operations	no (rebuild)	yes
Cache misses per query	$\geqslant 2$	$1 + \epsilon$

Fingerprint Retrieval (FiRe)

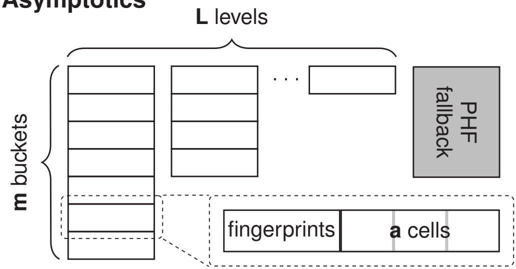
Overview



- **bucket** $= \text{hash}_1(key) \in \{1, \ldots, m\}$, **fingerprint** $= \text{hash}_2(key) \in \{1, \ldots, k\}$
- **Recursively overflow** to next level on fingerprint collision/full bucket
- Fingerprints implemented as bit vector for simplicity and speed

Fingerprint Retrieval (FiRe)

Asymptotics

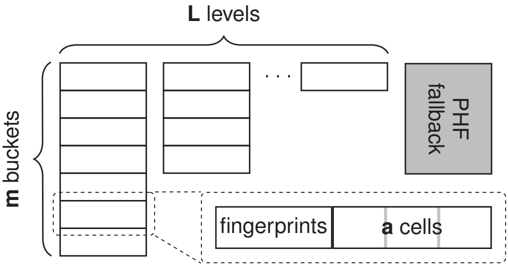


- n elements
- $m = \frac{n}{b}$ buckets
- a cells per bucket
- r-bit values

- Expected **linear construction time**
  - L FiRe levels, $O(n)$ for each level
  - Even for $L \rightarrow \infty$: geometric series, as only a constant fraction of the elements overflow
- **Constant worst-case query time**, since L is constant

Fingerprint Retrieval (FiRe)

Formulae



- n elements
- $m = \frac{n}{b}$ buckets
- a cells per bucket
- r-bit values

Let a_1 be the expected number of elements in a bucket

- **Space overhead** per element $s \approx \frac{r \cdot (a - a_1) + \text{size}(\textit{fingerprints})}{a_1}$ bits
- **Cache misses** per query $l \approx \frac{b}{a_1}$
- Calculation of a_1: see our paper

Fingerprint Retrieval (FiRe)

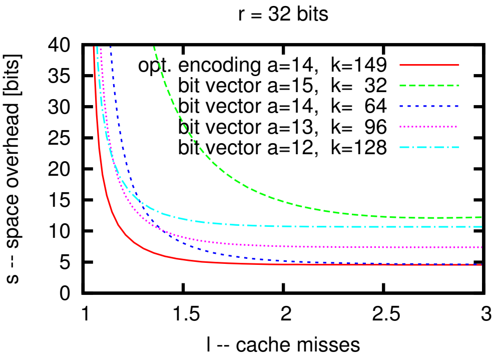
Space/Time Trade-Off

Space overhead s and expected number of **cache misses** l depend on

- a: #cells per bucket
- b: average #elements per bucket $(= \frac{n}{m})$
- k: #possible fingerprint values
- r: size of each value

How to choose parameters?

- a and k such that a bucket fits into a **cache line**
- Depending on desired **trade-off**

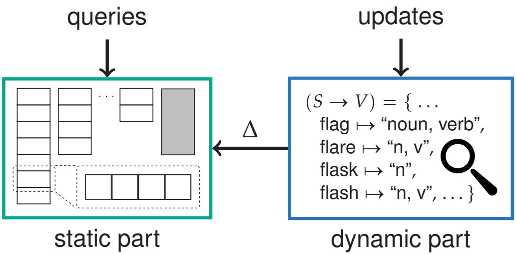


Fingerprint Retrieval (FiRe)

Dynamization

Updates and **deletions** (easy)

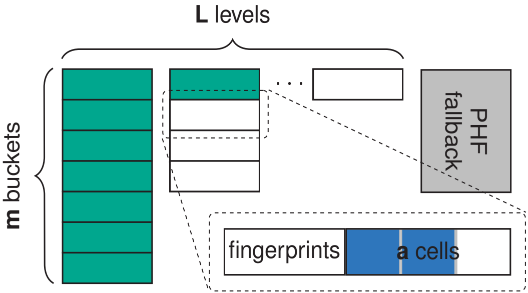
- Update associated value in place
- Ignore deletions



Insertions

- Needs a **dynamic part** with keys + some book-keeping information
- Answer queries with the **static part** (FiRe)
- **Idea:**
  - Overflow new and old element if fingerprints collide
  - "Block" fingerprint for future inserts
  - Rebuild when some stability criterion is violated

Fingerprint Perfect Hashing (FiPHa)



Fingerprint-Based **Perfect Hashing** (FiPHa)

- Special case with large buckets of “empty” values
- Associated ID is calculated as $\text{rank}_{\text{bucket}}(v) \cdot a + \text{rank}_{\text{fingerprint}}(v)$
- Very **space efficient** (2.79 bits overhead with 2.78 cache misses)

Experimental Results

Settings

Configurations of a, b, k such that

- $l = 1.05$ cache misses: **FiRe5**
- $l = 1.25$ cache misses: **FiRe25**
- $l = 1.50$ cache misses: **FiRe50**
- Retrieval data structure with **FiPHa** as PHF (3.78 cache misses)

Base lines

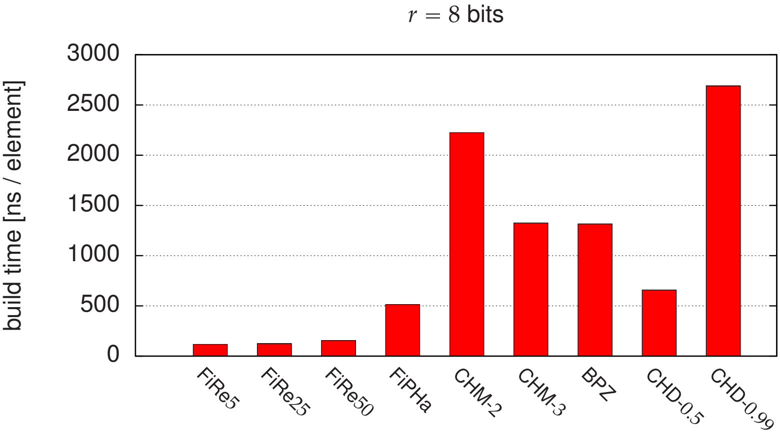
- BPZ, CHD-0.5/0.99 from the **C Minimal Perfect Hashing Library** [6]
- CHM-2/3 from our implementation

Datasets

- Keys: 100 million unique random 32-bit integers
- Values: integers of size $r = 8$ bits

Experimental Results

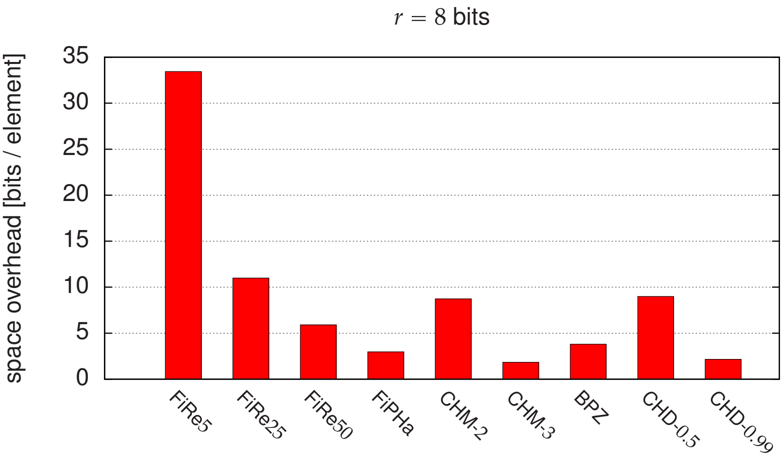
Build Times



- 4–17 times **faster sequential construction** for FiRe
- FiPHa slower, but faster than competitors

Experimental Results

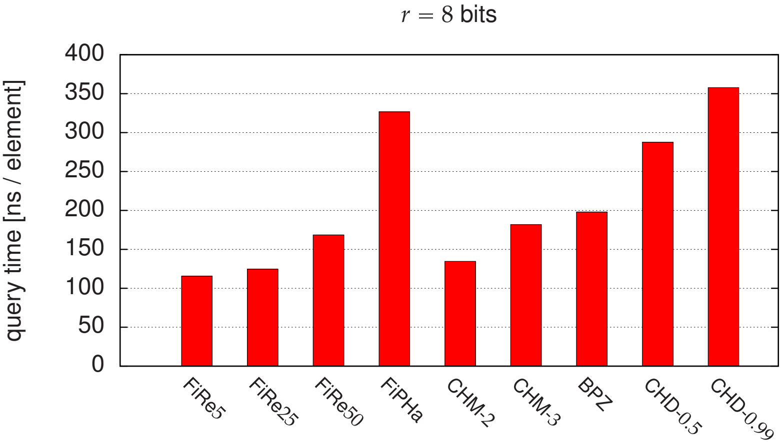
Space Overhead



- FiRe50 has **comparable overhead to most competitors**
- FiPHa almost on par with best competitor (CHD-0.99)

Experimental Results

Query Times



- FiRe has the **best query times** due to low number of cache misses
- FiPHa comparable to CHD-0.99, but has much faster construction

Summary and Future Work

Fingerprint Retrieval (FiRe) and **Perfect Hashing** (FiPHa)

- **Simple concept**, easy implementation
- Fast evaluation due to **low number of cache-misses**
- Extremely **fast construction**, even with sequential implementation
- Small space overhead
- Highly **configurable trade-off**
- Support for **updates**, **insertions**, and **deletions**

Future Work

- Find more compact, yet practical **representation of fingerprints**
- Adapt idea of **cuckoo-hashing** to fingerprinting
- Improve trade-off with **different settings** for each level
- Adapt fingerprinting idea to **other data structures**

Thank you

References I

[1] F. Färber *et al.*, "SAP HANA Database: Data management for modern business applications," *SIGMOD Rec.*, vol. 40, no. 4, pp. 45–51, 2012. [Online]. Available: http://doi.acm.org/10.1145/2094114.2094126

[2] F. C. Botelho and N. Ziviani, "External perfect hashing for very large key sets," in *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*. ACM, 2007, pp. 653–662.

[3] F. C. Botelho, R. Pagh, and N. Ziviani, "Simple and space-efficient minimal perfect hash functions," in *Algorithms and Data Structures*. Springer, 2007, pp. 139–150.

[4] D. Belazzougui, F. C. Botelho, and M. Dietzfelbinger, "Hash, displace, and compress," in *Algorithms-ESA 2009*. Springer, 2009, pp. 682–693.

[5] Z. J. Czech, G. Havas, and B. S. Majewski, "An optimal algorithm for generating minimal perfect hash functions," *Information Processing Letters*, vol. 43, no. 5, pp. 257–264, 1992.

[6] D. de Castro Reis, D. Belazzougui, F. C. Botelho, and N. Ziviani, "CMPH – C Minimal Perfect Hashing Library." [Online]. Available: http://cmph.sourceforge.net