

SIMD Acceleration for Index Structures: A Survey

Beyond Databases Architectures and Structures

Marten Wallewein-Eising, Gunter Saake, David Broneske

University of Magdeburg

September 18, 2018



Agenda

Motivation

SIMD Style Processing

Surveyed Main-Memory Index Structures

Elf

Seg-Tree/Trie

FAST

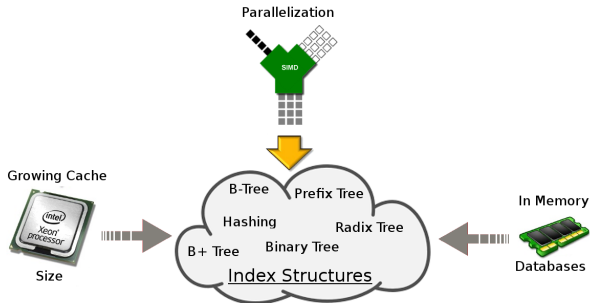
VAST

ART

Qualitative Comparison

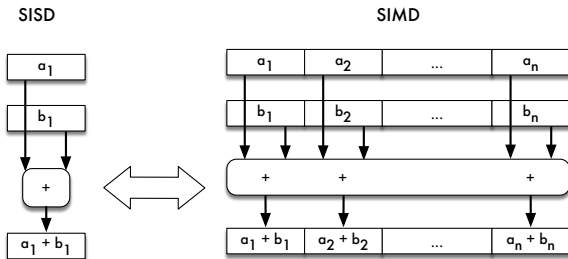
Conclusion

Motivation



- What are state-of-the-art index structures?
- Which optimizations do they have in common?

Single Instruction Multiple Data



- `__m128i _mm_add_epi32 (__m128i a, __m128i b)` Adds 4 signed 32-bit integers in `a` to 4 signed 32-bit integers in `b`.



Surveyed Main-Memory Index Structures

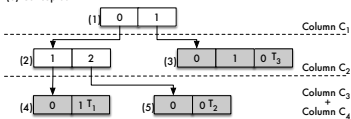
- Elf [Broneske et al. ICDE'17]
- Seg-Tree/Trie [Zeuch et al. EDBT'14]
- FAST: Fast Architecture Sensitive Tree [Kim et al. SIGMOD'10]
- VAST: Vector-Advanced and Compressed Structure Tree [Yamamuro et al. EDBT'12]
- ART: Adaptive Radix Tree [Leis et al. ICDE'13]

Elf

(a) Input Table

C ₁	C ₂	C ₃	C ₄	Ref
0	1	0	1	T ₁
0	2	0	0	T ₂
1	0	1	0	T ₃

(b) Conceptual Elf



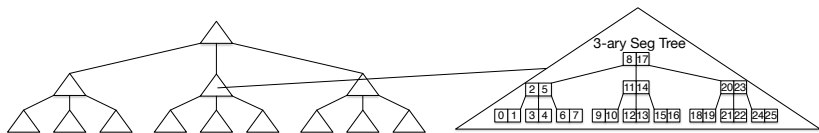
(c) Optimized Memory Layout for Elf

	0	1	2	3	4	5	6	7	8	9
Elf[00]	(1) [02]	(2) [-12]	1	-.6]	-2	-.9]	(4) 0	1	T ₁	(5) 0
Elf[10]	0	T ₂	(3) 0	1	0	T ₃				
Elf[20]										

- Multi-dimensional index structure for column-wise storage
- Prefix-redundancy elimination on distinct column values
- Linearisation for optimized memory layout

[Broneske et al. ICDE'17]

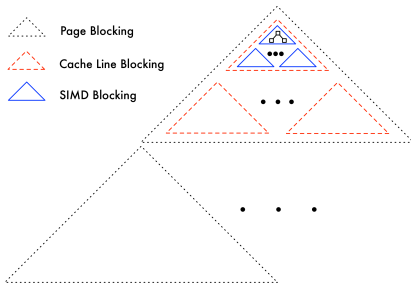
Seg-Tree/Trie



- Each node is a k-ary search tree
- Each node is linearised to use k-ary search
- $k = \frac{|SIMD|}{|Key|}$ partitions, $k - 1$ separator keys are compared in parallel

[Zeuch et al. EDBT'14]

Fast Architecture Sensitive Tree



- Based on binary tree
- Hierarchical blocking: page, cache line and SIMD blocks
- Efficient register, cache line and page usage

[Kim et al. SIGMOD'10]

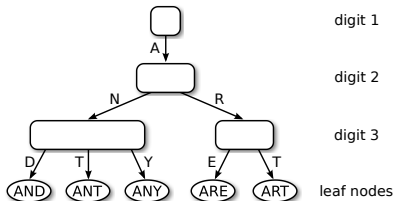


Vector-Advanced and Compressed Structure Tree

- Extension of FAST
- Decrease branch misses avoiding conditional branches
- Uses key compression
 - Lossy compression for inner nodes
 - Lossfree compression leaf nodes
- Decompression and error correction of lossy compression has less impact compared to the performance increase with SIMD

[Yamamuro et al. EDBT'12]

Adaptive Radix Tree



- Uses different node types with different number of keys and children
- Due to overfill or underfill of nodes, the node type is changed
- Reduced space consumption due to lazy expansion and path compression

[Leis et al. ICDE'13]

Qualitative Comparison

Criterion	Seg-Tree/ Trie	FAST	ART	VAST	Elf	Impact
Horizontal vectorization	x	x	x	x	-	high

Legend: x = implements the issue; o = partially implements the issue;
- = does not implement the issue

Table: Implementation of the considered performance criteria and their impact

Qualitative Comparison

Criterion	Seg-Tree/ Trie	FAST	ART	VAST	Elf	Impact
Horizontal vectorization	x	x	x	x	-	high
Minimized key size	o	-	x	x	-	high

Legend: x = implements the issue; o = partially implements the issue;
- = does not implement the issue

Table: Implementation of the considered performance criteria and their impact

Qualitative Comparison

Criterion	Seg-Tree/ Trie	FAST	ART	VAST	Elf	Impact
Horizontal vectorization	x	x	x	x	-	high
Minimized key size	o	-	x	x	-	high
Adapted node sizes / types	-	x	-	x	-	low

Legend: x = implements the issue; o = partially implements the issue;
- = does not implement the issue

Table: Implementation of the considered performance criteria and their impact

Qualitative Comparison

Criterion	Seg-Tree/ Trie	FAST	ART	VAST	Elf	Impact
Horizontal vectorization	x	x	x	x	-	high
Minimized key size	o	-	x	x	-	high
Adapted node sizes / types	-	x	-	x	-	low
Decreased branch misses	-	x	-	x	-	medium

Legend: x = implements the issue; o = partially implements the issue;
- = does not implement the issue

Table: Implementation of the considered performance criteria and their impact

Qualitative Comparison

Criterion	Seg-Tree/ Trie	FAST	ART	VAST	Elf	Impact
Horizontal vectorization	x	x	x	x	-	high
Minimized key size	o	-	x	x	-	high
Adapted node sizes / types	-	x	-	x	-	low
Decreased branch misses	-	x	-	x	-	medium
Exploit cache lines using blocking and alignment	-	x	-	x	x	medium

Legend: x = implements the issue; o = partially implements the issue;
- = does not implement the issue

Table: Implementation of the considered performance criteria and their impact

Qualitative Comparison

Criterion	Seg-Tree/ Trie	FAST	ART	VAST	Elf	Impact
Horizontal vectorization	x	x	x	x	-	high
Minimized key size	o	-	x	x	-	high
Adapted node sizes / types	-	x	-	x	-	low
Decreased branch misses	-	x	-	x	-	medium
Exploit cache lines using blocking and alignment	-	x	-	x	x	medium
Use of Compression	o	-	x	x	x	medium

Legend: x = implements the issue; o = partially implements the issue;
- = does not implement the issue

Table: Implementation of the considered performance criteria and their impact

Qualitative Comparison

Criterion	Seg-Tree/ Trie	FAST	ART	VAST	Elf	Impact
Horizontal vectorization	x	x	x	x	-	high
Minimized key size	o	-	x	x	-	high
Adapted node sizes / types	-	x	-	x	-	low
Decreased branch misses	-	x	-	x	-	medium
Exploit cache lines using blocking and alignment	-	x	-	x	x	medium
Usage of Compression	o	-	x	x	x	medium
Adapt search algorithm for linearized nodes	x	-	-	-	x	low

Legend: x = implements the issue; o = partially implements the issue;
- = does not implement the issue

Table: Implementation of the considered performance criteria and their impact



Conclusion

How to adapt index structures to modern database systems:

- Compare as many keys as possible in parallel with SIMD
 - Direct performance increase by a factor of X (where X keys fit a SIMD register)
- Efficient usage of cache line
- Decrease branch misses
- Use compression or/and adapted search algorithms



Conclusion

How to adapt index structures to modern database systems:

- Compare as many keys as possible in parallel with SIMD
 - Direct performance increase by a factor of X (where X keys fit a SIMD register)
- Efficient usage of cache line
- Decrease branch misses
- Use compression or/and adapted search algorithms

Thank you for your attention!

