SIMD Acceleration for Index Structures

Marten Wallewein-Eising

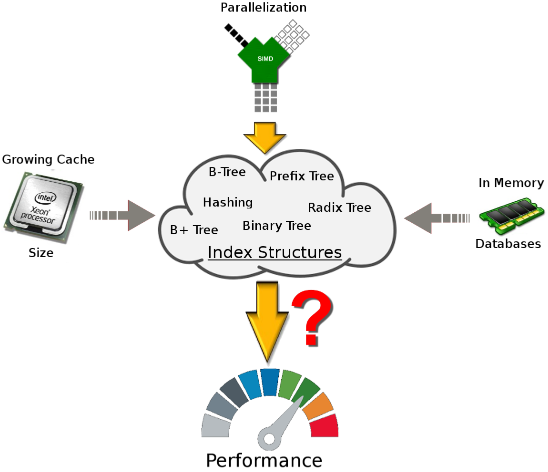
Otto von Guericke Univerity, Magdeburg

January 26, 2018

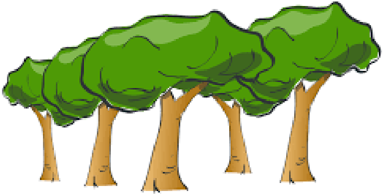
Agenda

- **Motivation**
- **Excursion: B^+-Tree**
- **SIMD Style Processing**
- **Adapted Tree structures**
  - Seg-Tree/Trie
  - FAST
  - VAST
  - ART
- **Evaluation**
- **Conclusion**

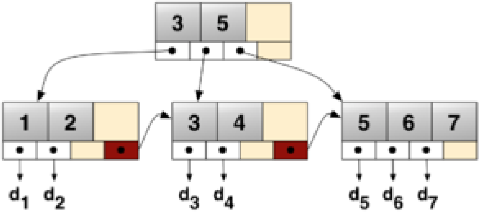
Motivation


Parallelization
SIMD
Growing Cache
Size
In Memory
Databases
B-Tree
Prefix Tree
Hashing
Radix Tree
B+ Tree
Binary Tree
Index Structures
Performance

Excursion: B^+-Tree

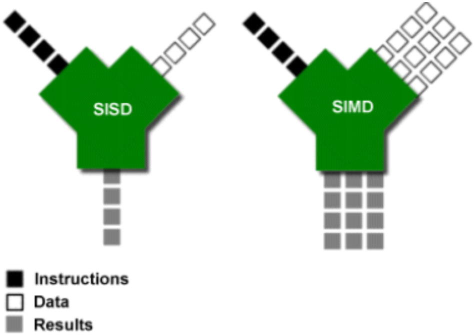


B^+-Tree



- N-ary tree with often a large number of children per node
- Only leaf nodes contain keys
- Leaf nodes often linked for range based scans

Single Instruction Multiple Data

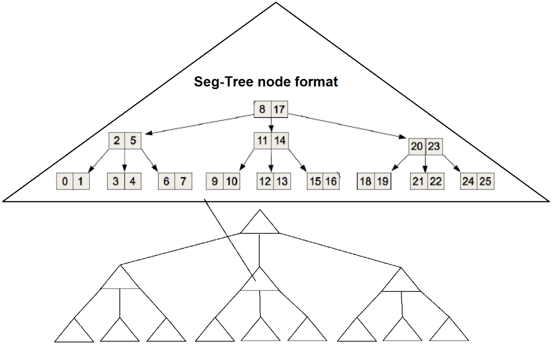


- **__m128i _mm_cmpgt_epi32 (__m128i a, __m128i b)**
  Compares 4 signed 32-bit integers in a and 4 signed 32-bit integers in b for greater-than.

Adapted Index Structures

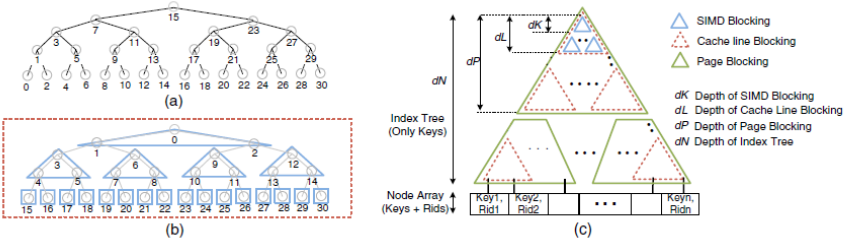
- Seg-Tree/Trie
- FAST: Fast Architecture Sensitive Tree
- VAST: Vector-Advanced and Compressed Structure Tree
- ART: Adaptive Radix Tree

Seg-Tree/Trie



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

Fast Architecture Sensitive Tree



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

Adapted Index Structures

- VAST: Vector-Advanced and Compressed Structure Tree
  - Extension of FAST
  - Uses key compression on lower levels of the tree
- ART: 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

Evaluation

Implementation of the considered performance criteria and their impact:

Criterium	Seg-Tree/Trie	FAST	ART	VAST	Impact
Horizontal vectorization	x	x	x	x	high
Minimized key size	o	-	x	x	high
Adapted node sizes and types	-	x	-	x	low
Decreased branch misses	-	x	-	x	medium
Full use of cache line using blocking and alignment	-	x	-	x	medium
Usage of Compression	o	-	x	x	medium
Adapt search algorithm for linearised nodes	x	-	-	-	low

Legend: x: implements the issue, o: partially implements the issue, -: not implements the issue

Conclusion

How to adapt index structures to modern database systems:

- Compare as many keys as possible in parallel with SIMD
  - Direct performance increase up to a multiple
- Efficient usage of cache line
- Decrease branch misses
- Use compression or adapted search algorithms

Sources

- `http://infolab.stanford.edu/~nsample/cs245/handouts/hw2sol/sol2.html`
- `https://www.clker.com/clipart-bosque.html`
- S. Zeuch, F. Huber and J.-C. Freytag "Adapting Tree Structures for Processing with SIMD Instructions" in EDBT, 2014.
- C. Kim, J. Chhugani, N. Satish, E. Sedlar, A. D. Nguyen, T. Kaldewey, V. W. Lee, S. A. Brandt and P. Dubey "FAST: Fast Architecture Sensitive Tree Search on Modern CPUs and GPUs" in SIGMOD, pp. 339-350, 2010.
- V. Leis, A. Kemper and T. Neumann "The Adaptive Radix Tree: ARTful Indexing for Main-Memory Databases" in ICDE, pages 38-49, 2013.

Sources

- T. Yamamuro, M. Onizuka, T. Hitaka, and M. Yamamuro "VAST-Tree: A Vector-Advanced and Compressed Structure for Massive Data Tree Traversal" in EDBT, pp. 396-407, 2012.

Thank you for your attention!