SIMD Acceleration for Index Structures

Marten Wallewein-Eising
Otto-von-Guericke University
Magdeburg, Germany
marten.wallewein-eising@st.ovgu.de

***Abstract*—**

- **summary:**
  Give short an overview of SIMD and modern index structures
  Explain what are the problems of the "old" index structures made for disk-based database systems
  Explain which approaches were made to adapt index structures to modern systems and what they have in common and what are differences
- **Why is this work important:**
  Give a state of current development of the index structures
  Collect common approaches to adapt other index structures TODO: ReThink
- **K-ary search trees, FAST, VAST and ART compared**
- **Contribution: What are important approaches used by different implementations to adapt index structures to modern systems**

***Index Terms*—SIMD, index**

I. Introduction

After decades of creating and improving index structures for disk-based database systems, nowadays even large databases fit into the main memory. Since index structures like the B^+-tree [TODO: ref] or the radix tree [TODO: ref] have an important part in database-systems to realise SCAN or range-based search operations, these index structures experienced many adaptions to fulfill the needs of modern database-systems. Instead of overcoming the bottleneck of IO-oprations from disk to RAM, the target of modern index-structures is to improve the usage of CPU cache and processor architectures.

In this paper we compare different approaches to adapt index structures to use Single Instruction Multiple Data (SIMD) [1] operations and to better overcome the bottleneck from RAM to CPU cache. We consider the K-ary Search Tree (Seg-tree) [TODO: ref], Adapted Radix Tree (ART) [4], Fast Architecture Sensitive Tree (FAST) [6], and Vector-Advanced and Compressed Structure Tree (VAST) [5]. As the authors of VAST-Tree show, important causes for increased calculation time are cache misses and branch mispredictions. To overcome branch mispredictions and to decrease CPU cycles, SIMD [TODO: ref] is used in modern index structures for tree traversal. The authors of the k-ary search show how to use SIMD to compare mutliple keys in one CPU cycle. To decrease cache misses, the authors of FAST and ART show how to adapt index structures to the cache line size.

All approaches use SIMD only for key comparison within tree traversal and try to decrease the key size to fit more keys into one SIMD register. Therefore FAST and Seg-tree only provide implementations for search algorithms. We consider the approaches VAST and ART make to implement operations like update and insert und name ideas to use SIMD for them. Consequently, with this work we make the following contributions:

- We compare different adaptions of index structures to fulfill requirements of modern database systems
- We highlight the usage of SIMD and the cache line adaptions in all approaches
- We show openings in the adaptions to use other approaches to use SIMD

We organized the rest of the paper as follows. In section 2 we give the preliminaries for SIMD in general and for the use in index structures. In section 3 we analyse the different approaches of adapted index structures and evaluate the comparision in section 4. In section 5 we name related work. In section 5 we present our conclusion und describe future work in section 7.

II. Preliminaries

A. *Single instruction multiple data*

Short overview of the following:

- How SIMD works
- SIMD vs vertical vector processing

B. *Considered index structures*

Short overview of the following: TODO: Maybe the problems of these old index structures?

- Binary tree???
- B^+ tree
- Radix tree

III. Adapted Tree Structures

TODO: Compare all 4 or merge FAST and VAST together/ extend FAST with VAST??

A. *K-ary search tree*

B. *FAST*

C. *ART*

D. *VAST*

IV. Evaluation

In common:

- SIMD instructions used to compare the search key with multiple keys of the index

- Segmenting tree to blocks for a better usage of cache lines, save the data of the nodes in an adapted way
- The keys should be as short as possible to compare more keys in one step and to decrease the passed data to the cache line
- Each approach improves the tree traversal

Differences:

- Node compression in VAST, Path compression in ART and K-ary seg trie
- FAST and K-ary trees readonly to improve traversal, ART and FAST adapt insert too
- FAST uses and K-ary trees will use GPU calculation instead of CPU

Why performance can not be compared in a useful way...

V. Related Work

TODO:

- ART and VAST compared to FAST??
- Ideas and implementations of the adapted trees already in III...
- KD-Tree with SIMD

VI. Conclusion

VII. Future work

Open questions, use SIMD for tree creation/updates instead of only for traversal

References

[1] Mohammad Suaib, Abel Palaty and Kumar Sambhav Pandey, “Architecture of SIMD Type Vector Processor” in International Journal of Computer Applications (0975 - 8887) Volume 20 No.4, April 2011.

[2] Jingren Zhou and Kenneth A. Ross “Implementing Database Operations Using SIMD Instructions” in ACM S1GMOD ’2002 June 4-6, Madison, Wisconsin, USA

[3] Steffen Zeuch, Frank Huber and Johann-Christoph Freytag “Adapting Tree Structures for Processing with SIMD Instructions” in Proc. 17th International Conference on Extending Database Technology (EDBT), March 24-28, 2014, Athens, Greece

[4] Viktor Leis, Alfons Kemper and Thomas Neumann “The Adaptive Radix Tree: ARTful Indexing for Main-Memory Databases” in ???

[5] Takeshi Yamamuro, Makoto Onizuka,Toshio Hitaka, and Masashi Yamamuro “VAST-Tree: A Vector-Advanced and Compressed Structure for Massive Data Tree Traversal” in EDBT 2012, March 26-30, 2012, Berlin, Germany.

[6] Changkyu Kim, Jatin Chhugani, Nadathur Satish, Eric Sedlar, Anthony D. Nguyen, Tim Kaldewey, Victor W. Lee, Scott A. Brandt and Pradeep Dubey “FAST: Fast Architecture Sensitive Tree Search on Modern CPUs and GPUs” in SIGMOD10, June 6-11, 2010, Indianapolis, Indiana, USA.