Survey of Vectorized Database Techniques

CS 541

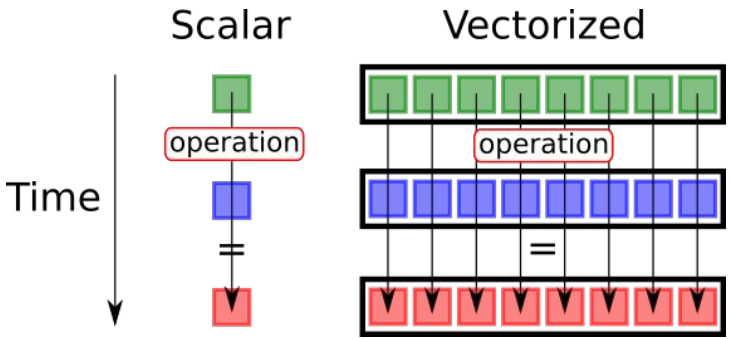
Group: VectDB2

Final Presentation

Shivam Bhat (PUID : 33760929)
Venkat Sai (PUID : 34779773)
Akshay Ramakrishnan (PUID : 33660675)

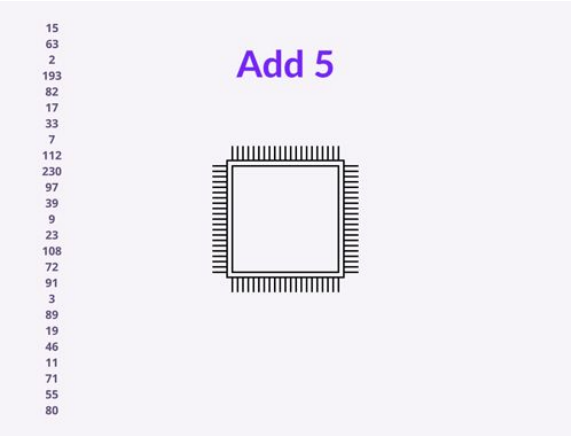
What is Vectorization

This refers to the process of **transforming an algorithm's scalar implementation**, which currently handles one pair of operands at a time, **into a vector implementation** that can handle multiple pairs of operands simultaneously.

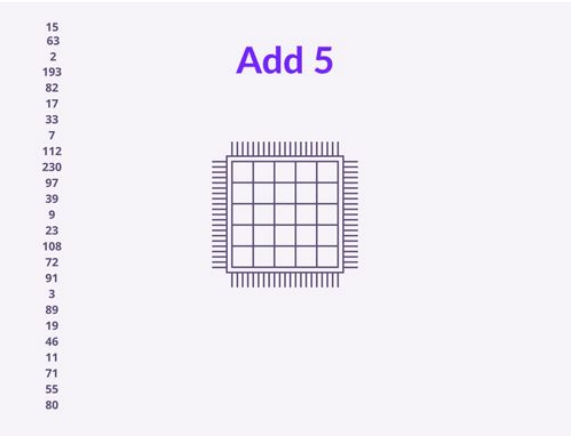


What is SIMD?

Single / same
Instruction / operation on
Multiple
Data (lanes)

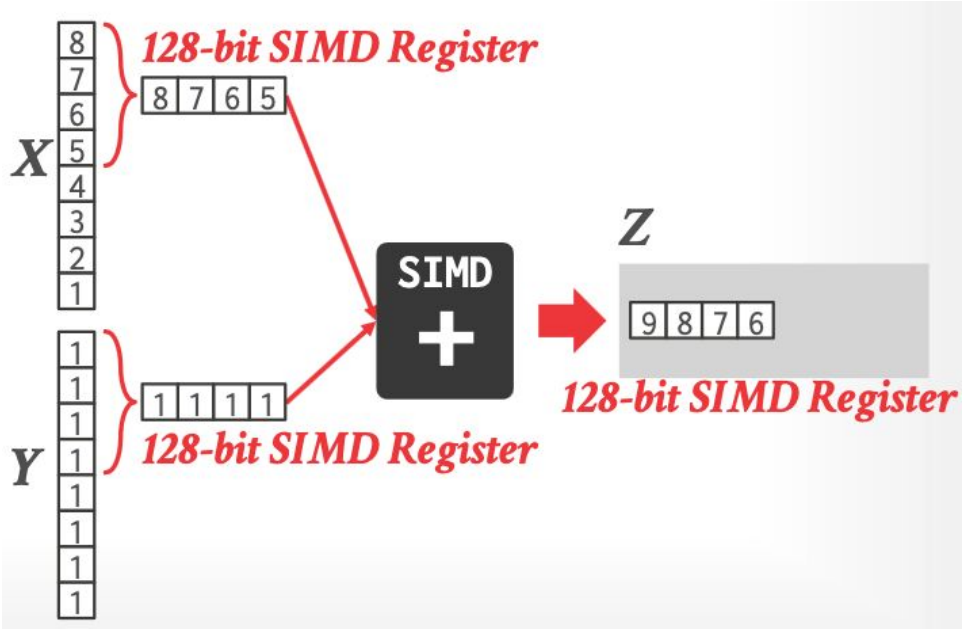


SISD



SIMD

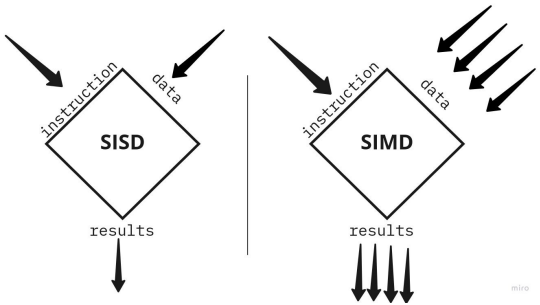
SIMD EXAMPLE



Source: Orestis Polychroniou, Arun Raghavan, and Kenneth A. Ross. 2015. Rethinking SIMD Vectorization for In-Memory Databases. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15). Association for Computing Machinery, New York, NY, USA, 1493–1508. https://doi.org/10.1145/2723372.2747645
Andy P, CMU 15-721 Course

Why SIMD?

- CPUs provide SIMD/vector instructions that apply the same operation to multiple data items.
- **Improve memory usage** and cache efficiency, as **vectorized operations typically require fewer memory accesses** and can be more easily optimized for cache locality.
- This can **reduce energy usage** e.g. *fivefold* because fewer instructions are executed. We also often see ***5-10x* speedups.**
- Unlike a scalar implementation, a **vectorized DBMS system can apply the the filter operation** to entire vectors of data at once
- Useful for **programs that are very intensive on numeric operations** like image/audio/video processing , crypto/Hashing and physics engines



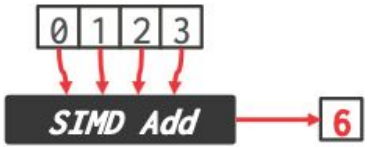
Adoption in Industry

All major ISAs have microarchitecture support
SIMD operations.

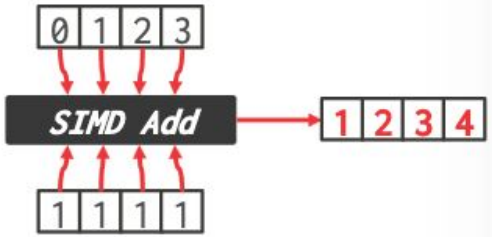
- x86: MMX, SSE, SSE2, SSE3, SSE4, AVX, AVX2, AVX512
- PowerPC: Altivec
- ARM: NEON, SVE
- RISC-V: RVV

Vectorization Types

Horizontal Vectorization



Vertical Vectorization



Source: Orestis Polychroniou, Arun Raghavan, and Kenneth A. Ross. 2015. Rethinking SIMD Vectorization for In-Memory Databases. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15). Association for Computing Machinery, New York, NY, USA, 1493–1508. https://doi.org/10.1145/2723372.2747645
Andy P, CMU 15-721 Course

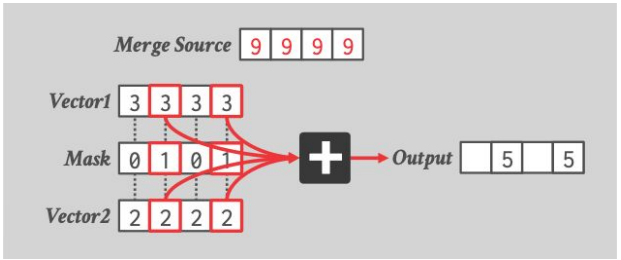
VECTORIZATION FUNDAMENTALS

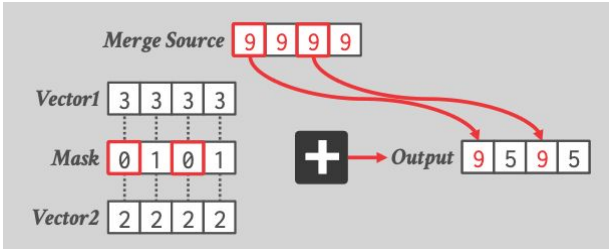
There exists some fundamental SIMD operations that the DBMS will use to build more complex functionality:

- Masking
- Permute
- Selective Load/Store
- Compress/Expand
- Selective Gather/Scatter

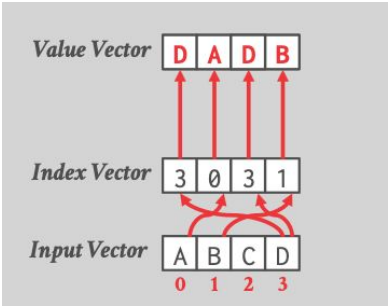
Source: Orestis Polychroniou, Arun Raghavan, and Kenneth A. Ross. 2015. Rethinking SIMD Vectorization for In-Memory Databases. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15). Association for Computing Machinery, New York, NY, USA, 1493–1508. https://doi.org/10.1145/2723372.2747645

Masking

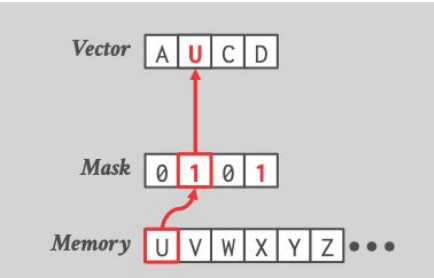


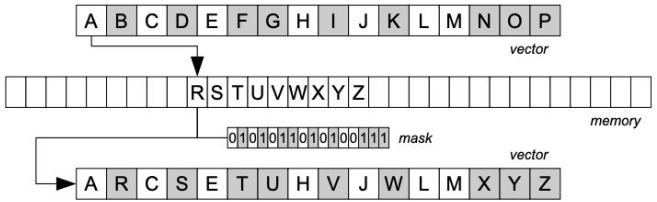


Permute

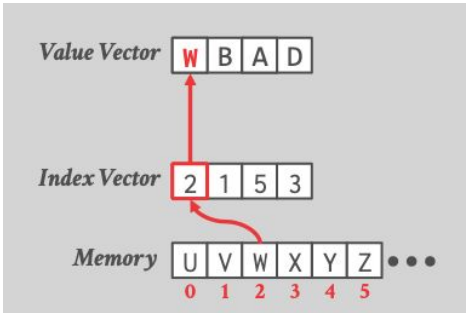


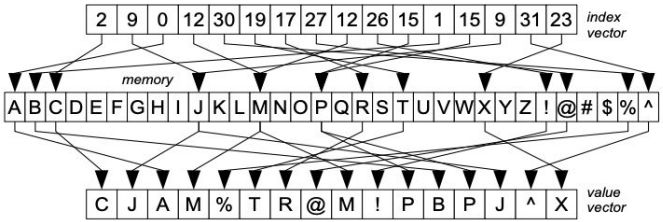
Selective Load



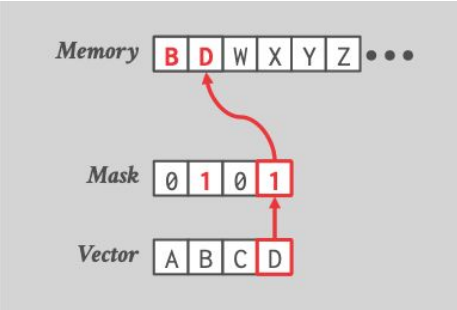


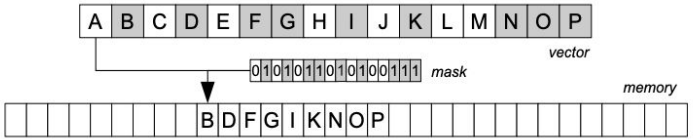
Selective Gather



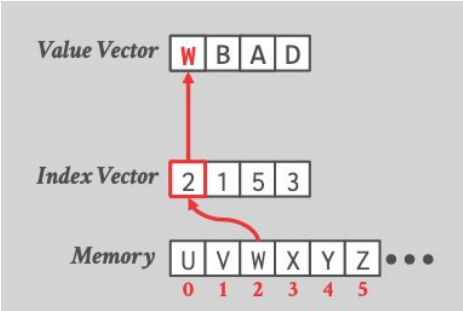


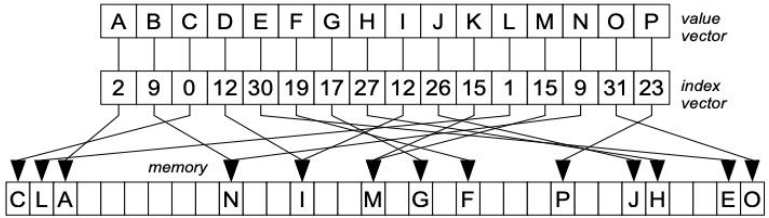
Selective Store





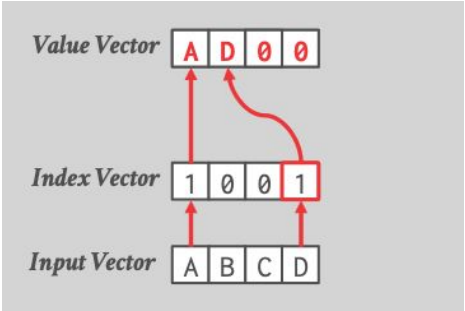
Selective Scatter



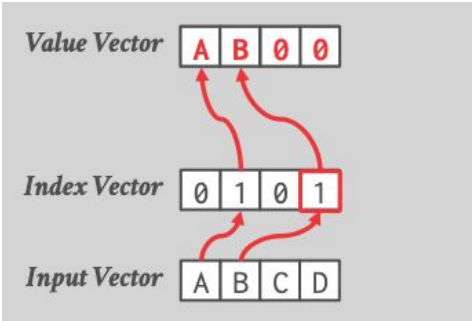


Source: Orestis Polychroniou, Arun Raghavan, and Kenneth A. Ross. 2015. Rethinking SIMD Vectorization for In-Memory Databases. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15). Association for Computing Machinery, New York, NY, USA, 1493–1508. https://doi.org/10.1145/2723372.2747645
Andy P, CMU 15-721 Course

Compress



Expand



Source: Orestis Polychroniou, Arun Raghavan, and Kenneth A. Ross. 2015. Rethinking SIMD Vectorization for In-Memory Databases. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15). Association for Computing Machinery, New York, NY, USA, 1493–1508. https://doi.org/10.1145/2723372.2747645

How is everything tied together?

VECTORIZED OPERATORS

Selection Scans

SELECT * FROM table
WHERE key >= $low AND key <= $high

Scalar (Branching)

```
i = 0
for t in table:
  key = t.key
  if (key≥low) && (key≤high):
    copy(t, output[i])
    i = i + 1
```

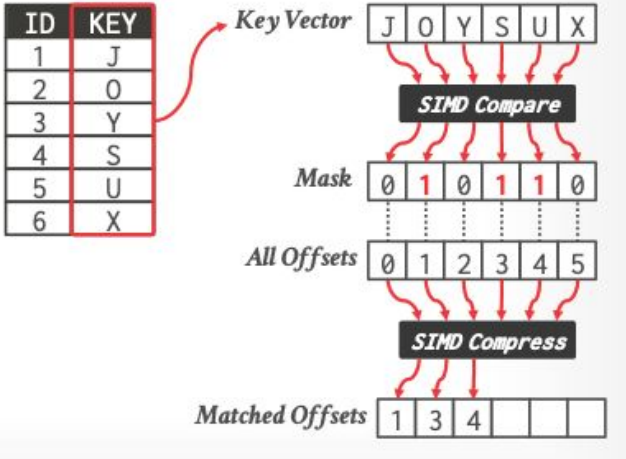
Scalar (Branchless)

```
i = 0
for t in table:
  copy(t, output[i])
  key = t.key
  m = (key≥low ? 1 : 0) &&
     ↳(key≤high ? 1 : 0)
  i = i + m
```

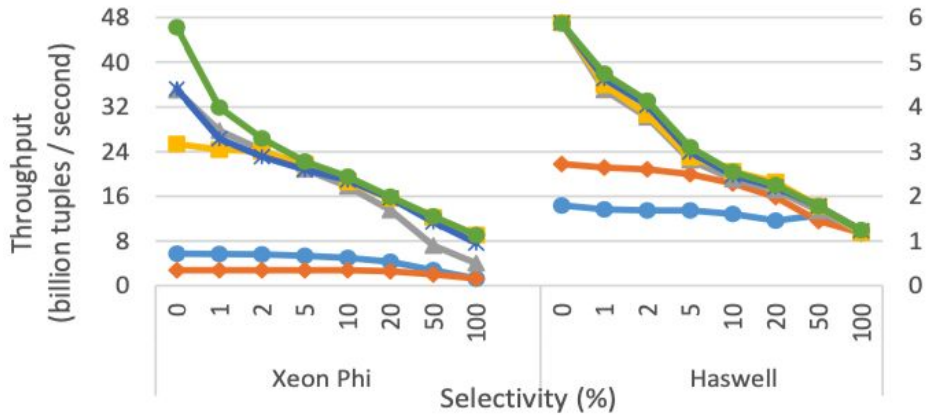
Vectorized Selection Scan

```
i = 0
for v_t in table:
  simdLoad(v_t.key, v_k)
  v_m = (v_k≥low ? 1 : 0) &
      ↳(v_k≤high ? 1 : 0)
  simdStore(v_t, v_m, output[i])
  i = i + |v_m≠false|
```

```
SELECT * FROM table
 WHERE key >= "O" AND key <= "U"
```

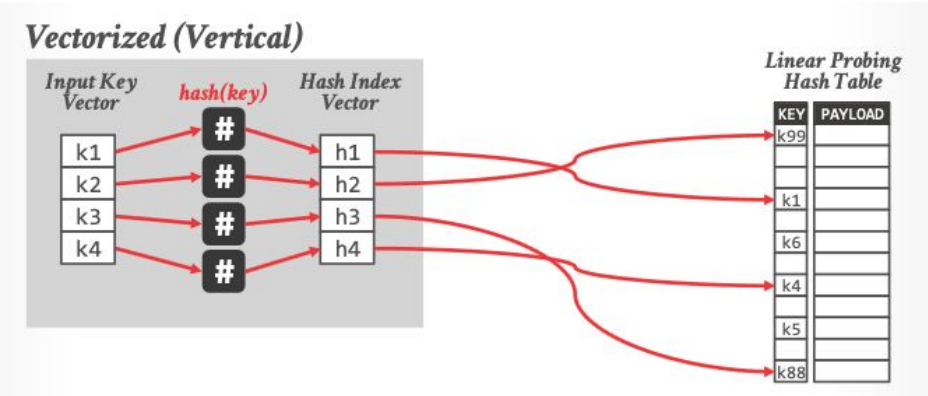


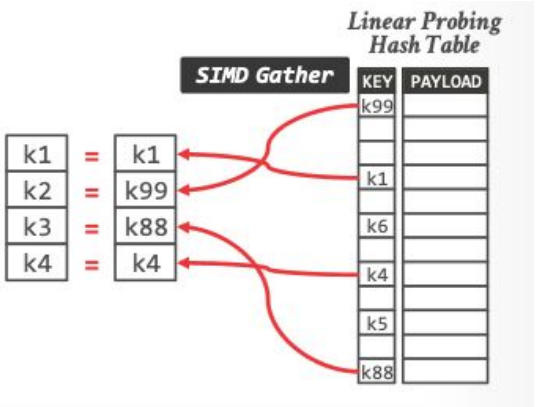
Vectorized Selection Scan Ourperforms



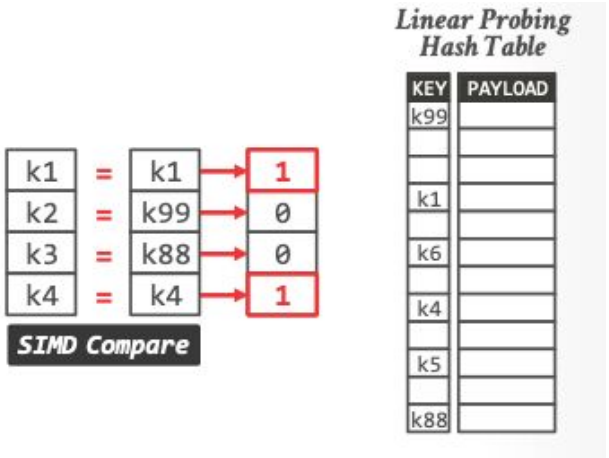
Source: Orestis Polychroniou, Arun Raghavan, and Kenneth A. Ross. 2015. Rethinking SIMD Vectorization for In-Memory Databases. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15). Association for Computing Machinery, New York, NY, USA, 1493–1508. https://doi.org/10.1145/2723372.2747645

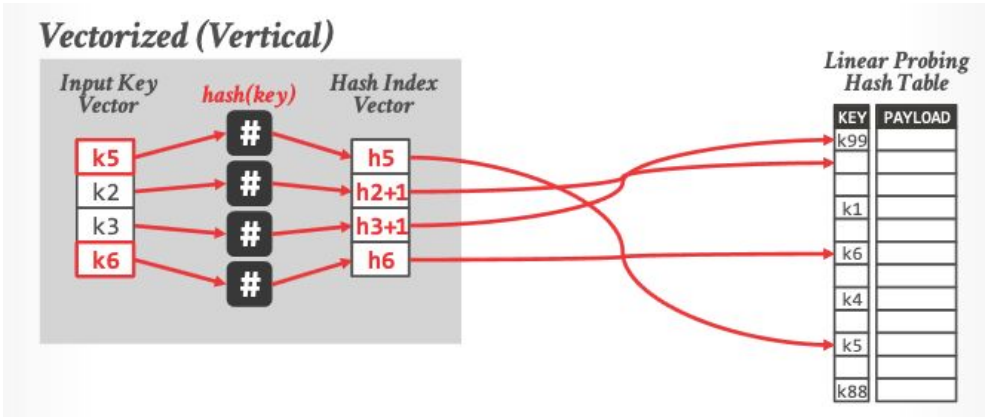
Vectorized Hashing





Vectorized Hashing





Other Operations

- Hashing
- Bloom Filters
- Histograms
- Partitioning

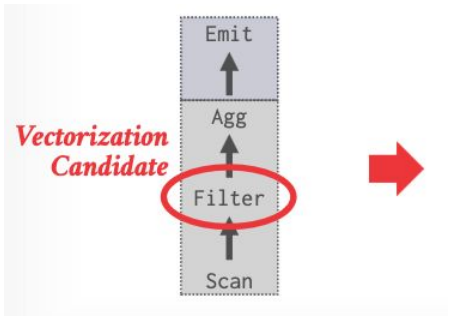
Relaxed Operator Fusion

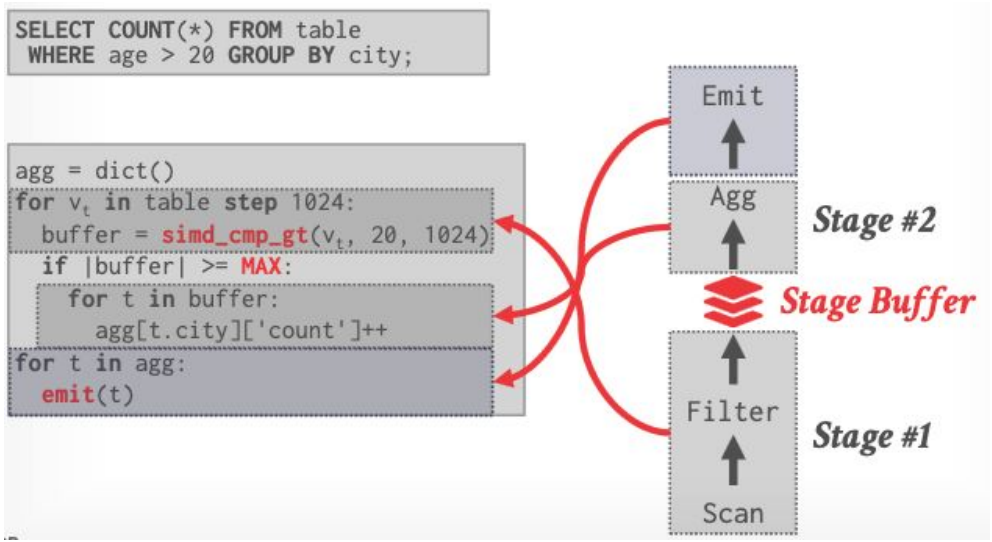
ROF is a query processing model that allows the DBMS to take advantage of inter-tuple parallelism inherent in the plan using a combination of prefetching and SIMD vectorization to support faster query execution on data sets that exceed the size of CPU-level caches.

1. **Software Prefetch**: instructions can move blocks of data from memory into the CPU caches before they are needed, thereby hiding the latency of expensive cache misses

2. **SIMD** instructions can exploit vector-style data parallelism to boost computational throughput .

Relaxed Operator Fusion



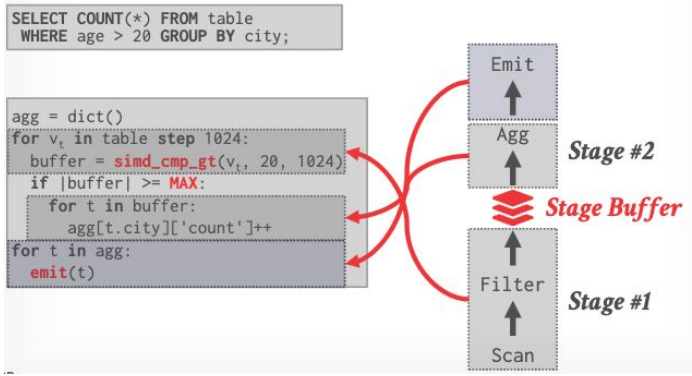


Source: Orestis Polychroniou, Arun Raghavan, and Kenneth A. Ross. 2015. Rethinking SIMD Vectorization for In-Memory Databases. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15). Association for Computing Machinery, New York, NY, USA, 1493–1508. https://doi.org/10.1145/2723372.2747645

Prefetching

The DBMS can tell the CPU to grab the next vector while it works on the current batch - Non Blocking Call

1. DBMS instead decomposes pipelines into stages.
2. A stage is a partition of a pipeline in which all operators are fused together.
3. Stages within a pipeline communicate solely through cache-resident vectors of tuple IDs.
4. Tuples are processed sequentially through operators in any given stage one-at-a-time.
5. If the tuple is valid in the stage, its ID is appended into the stage's output vector.



Source: Orestis Polychroniou, Arun Raghavan, and Kenneth A. Ross. 2015. Rethinking SIMD Vectorization for In-Memory Databases. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15). Association for Computing Machinery, New York, NY, USA, 1493–1508. https://doi.org/10.1145/2723372.2747645
Andy P, CMU 15-721 Course

Leveraging Vectorization in Databases

Adaptive Radix Tree

1. The Adaptive Radix Tree (ART) is a data structure that provides efficient and scalable indexing for storing and retrieving key-value pairs.
2. It dynamically adapts its structure based on the number of keys and the distribution of key lengths, allowing for efficient storage and retrieval of keys with varying lengths.
3. ART is designed to be memory-efficient and optimized for keys with varying lengths, making it well-suited for scenarios where the key length distribution is unpredictable or where memory usage needs to be minimized.

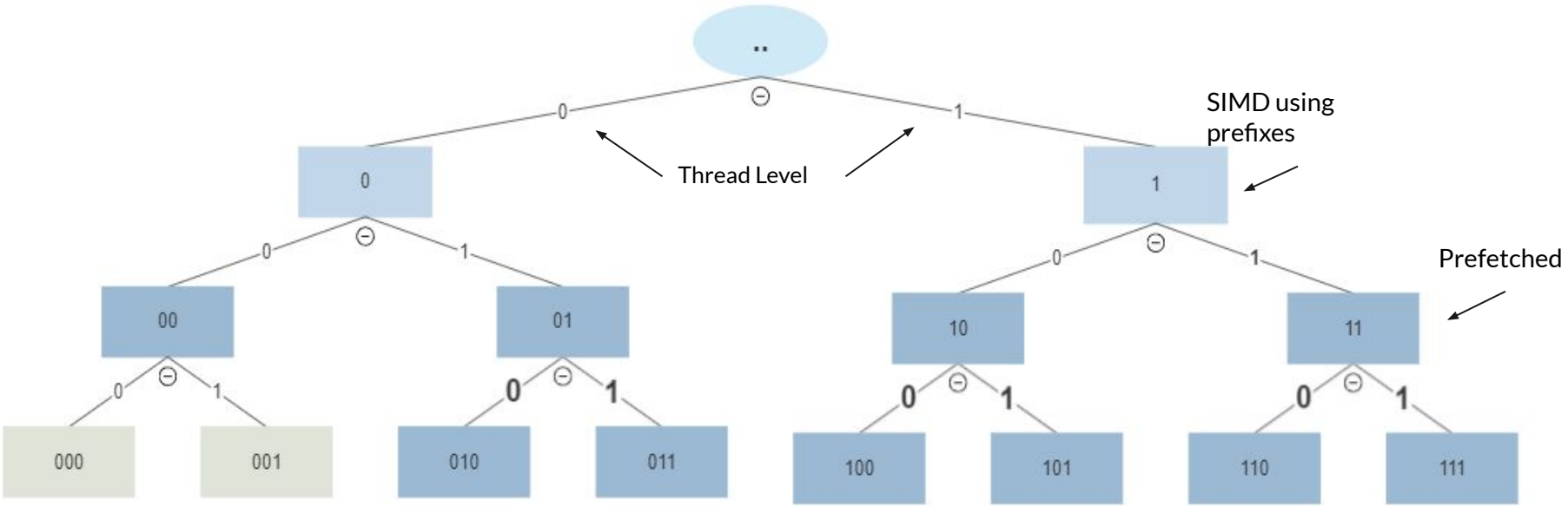
Leis, V., Kemper, A., & Neumann, T. (2013). The adaptive radix tree: Artful indexing for main-memory databases. *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. https://doi.org/10.1109/icde.2013.6544812

Vectorization

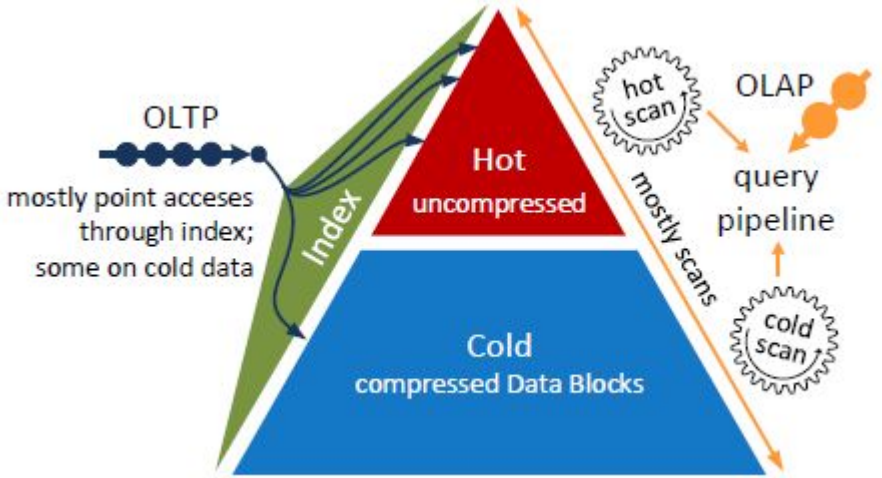
1. ART makes use of **hardware prefetching** by using the *_mm_prefetch instruction* provided by Intel's SIMD instruction set. (A node can fit into cache).
2. In terms of SIMD parallelism, **ART can take advantage of SIMD instructions** to compare multiple keys in parallel during a range query.
3. In terms of thread-level parallelism, ART can be easily **parallelized by splitting the tree into multiple subtrees and processing each subtree in parallel using multiple threads.**

Leis, V., Kemper, A., & Neumann, T. (2013). The adaptive radix tree: Artful indexing for main-memory databases. *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. https://doi.org/10.1109/icde.2013.6544812

Range Search Using ART



Datablocks



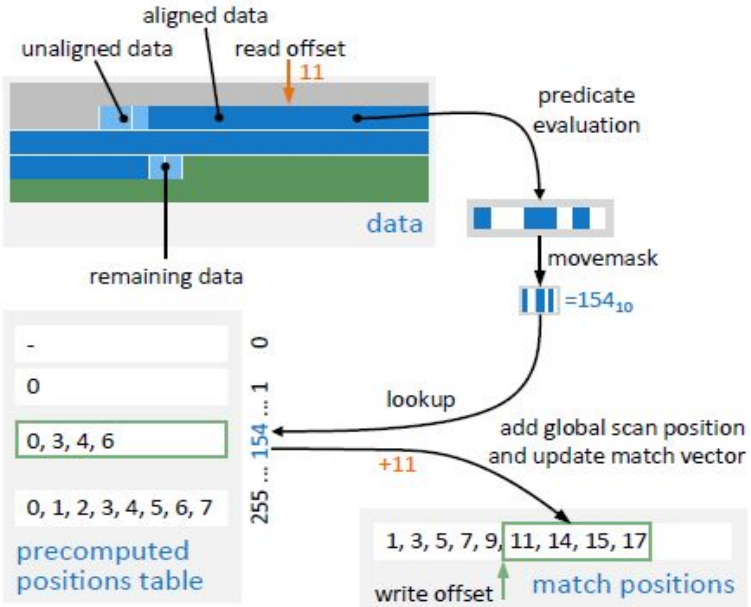
- Data Blocks is a novel compressed columnar storage format for hybrid database systems working both for OLAP and OLTP workloads. A high level idea:

Source: Harald Lang, Tobias Mühlbauer, Florian Funke, Peter A. Boncz, Thomas Neumann, and Alfons Kemper. 2016. Data Blocks: Hybrid OLTP and OLAP on Compressed Storage using both Vectorization and Compilation. In Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16). Association for Computing Machinery, New York, NY, USA, 311–326. https://doi.org/10.1145/2882903.2882925

Vectorization

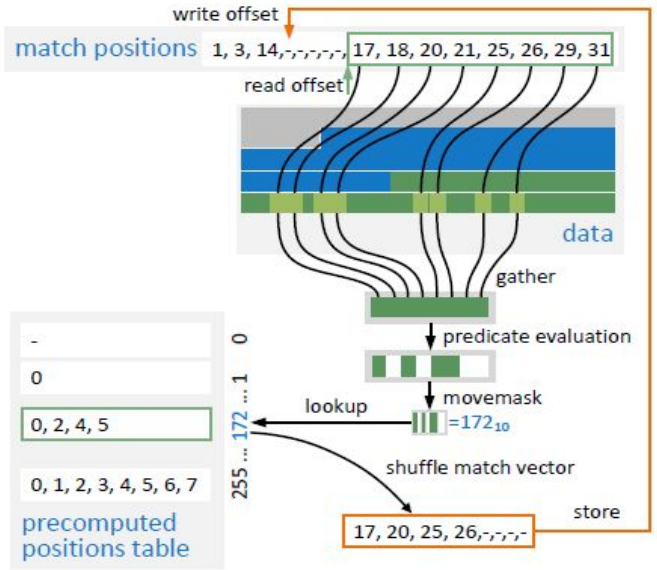
- Datablocks leverages SIMD optimized algorithms for predicate evaluation, and
- It integrates multiple storage layout combinations in a compiling tuple-at-a-time query engine by using vectorization.

Source: Harald Lang, Tobias Mühlbauer, Florian Funke, Peter A. Boncz, Thomas Neumann, and Alfons Kemper. 2016. Data Blocks: Hybrid OLTP and OLAP on Compressed Storage using both Vectorization and Compilation. In Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16). Association for Computing Machinery, New York, NY, USA, 311–326. https://doi.org/10.1145/2882903.2882925



Example on how matches are found without restrictions.

Source: Harald Lang, Tobias Mühlbauer, Florian Funke, Peter A. Boncz, Thomas Neumann, and Alfons Kemper. 2016. Data Blocks: Hybrid OLTP and OLAP on Compressed Storage using both Vectorization and Compilation. In Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16). Association for Computing Machinery, New York, NY, USA, 311–326. https://doi.org/10.1145/2882903.2882925



Example on how matches are found with restrictions.

Source: Harald Lang, Tobias Mühlbauer, Florian Funke, Peter A. Boncz, Thomas Neumann, and Alfons Kemper. 2016. Data Blocks: Hybrid OLTP and OLAP on Compressed Storage using both Vectorization and Compilation. In Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16). Association for Computing Machinery, New York, NY, USA, 311–326. https://doi.org/10.1145/2882903.2882925

Conclusion

In summary, both ART and Datablocks leverage various forms of vectorization to perform on par with their counterparts meanwhile expanding their domain.

ART can perform better than the hash tables when the data is skewed wrt the keys and can give tree like performance for range lookups and is highly memory-efficient and provides fast point lookup operations

Datablocks can provide a format that can be used for both OLTP and OLAP workloads even though it is highly efficient when the data is in integer format so that it can make use of vectorization.

Database Algorithms that utilize SIMD for Vectorization

Vectorizing Bloom Filtering Probing

1. One of the major applications of vectorization is the probing of bloom filters.
2. They are relevant in the context of database engines because they are an essential data structure for applying selective conditions across tables before joining them (semi joins).
3. To test if an item is in the set, it is hashed by each of the hash functions and the corresponding bits are checked. If all of them are set to 1, the item is probably in the set.

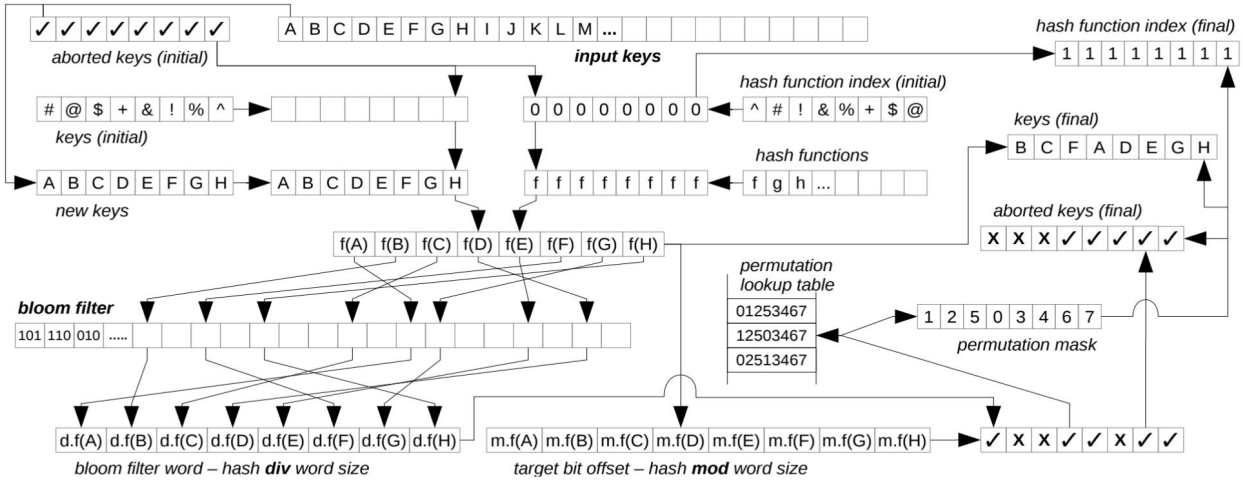
Source: Orestis Polychroniou and Kenneth A. Ross. 2014. Vectorized Bloom filters for advanced SIMD processors. In Proceedings of the Tenth International Workshop on Data Management on New Hardware (DaMoN '14). Association for Computing Machinery, New York, NY, USA, Article 6, 1–6. https://doi.org/10.1145/2619228.2619234

How is this implemented in a scalar fashion?

1. In the scalar implementation, we load **one key at a time** and we iteratively check over all the hash functions.
2. If the bits of all the hash functions are set, then we can conclude that the key probably belongs to the set.
3. If not, the key is discarded

Source: Orestis Polychroniou and Kenneth A. Ross. 2014. Vectorized Bloom filters for advanced SIMD processors. In Proceedings of the Tenth International Workshop on Data Management on New Hardware (DaMoN '14). Association for Computing Machinery, New York, NY, USA, Article 6, 1–6. https://doi.org/10.1145/2619228.2619234

How is this vectorized?



Source: Orestis Polychroniou and Kenneth A. Ross. 2014. Vectorized Bloom filters for advanced SIMD processors. In Proceedings of the Tenth International Workshop on Data Management on New Hardware (DaMoN '14). Association for Computing Machinery, New York, NY, USA, Article 6, 1–6. https://doi.org/10.1145/2619228.2619234

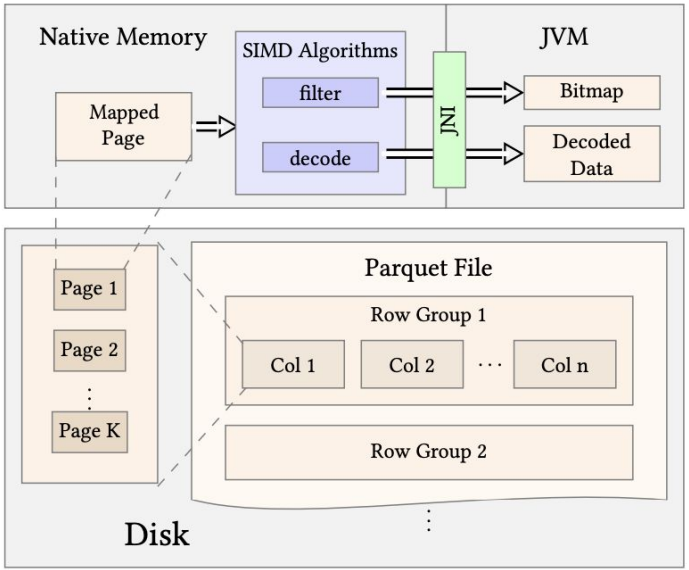
Vectorized Filtering of Encoded Data

1. Data in columnar databases are stored in an encoded format in order to save storage space.
2. In typical open-source columnar data formats, the encoded data needs to be decoded first to memory because queries can be executed on that data.
3. The proposed solution is **SBoost** - A columnar data store which leverages SIMD algorithms to execute filter operators directly on encoded data.

Source: Hao Jiang and Aaron J. Elmore. 2018. Boosting data filtering on columnar encoding with SIMD. In Proceedings of the 14th International Workshop on Data Management on New Hardware (DAMON '18). Association for Computing Machinery, New York, NY, USA, Article 6, 1–10. https://doi.org/10.1145/3211922.3211932

System Design

1. The Parquet file consists of multiple row groups consisted of fix sizes pages.
2. When a filter or decode operation is to the executed on these pages, they are mapped to a off-heap memory.
3. The corresponding SIMD operations are invoked through the JNI to process all the data items in the page and the result is then passed back to the JVM.



Source: Hao Jiang and Aaron J. Elmore. 2018. Boosting data filtering on columnar encoding with SIMD. In Proceedings of the 14th International Workshop on Data Management on New Hardware (DAMON '18). Association for Computing Machinery, New York, NY, USA, Article 6, 1–10. https://doi.org/10.1145/3211922.3211932

Vectorized Filtering in Action for Less-than and Equal-to Operations for Bit-Packed Encoding:

Let **X** = a SIMD vector of **n** entries
Let **M** = a mask where the MSB of each entry is 1, the remaining are 0.
Let **A** = a SIMD vector where the scalar to be matched **(a)** is replicated **n** times.

$$D = X \oplus A$$
$$R = D \mid ((D \,\&\, \sim M) + \sim M)$$

Equality Check

$$U = (X \mid M) - (A \,\&\, \sim M)$$
$$R = (\sim A \,\&\, (X \mid U)) \mid (X \,\&\, U)$$

Lesser-than Check

In both cases, the condition is satisfied when R_i(MSB) == 0

Source: Hao Jiang and Aaron J. Elmore. 2018. Boosting data filtering on columnar encoding with SIMD. In Proceedings of the 14th International Workshop on Data Management on New Hardware (DAMON '18). Association for Computing Machinery, New York, NY, USA, Article 6, 1–10. https://doi.org/10.1145/3211922.3211932

Database Engines designed for Vectorized Execution

Quickstep: A Data Platform Based on the Scaling-Up Approach

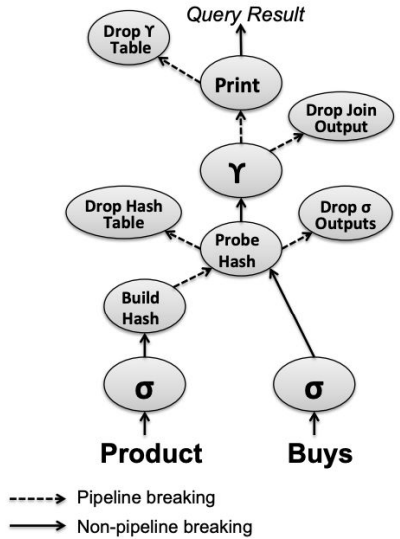
1. Quickstep is a database system that allows for intra-operator parallelism to execute its workloads, thereby aiming to exploit the full potential of the high level of hardware compute parallelism available.
2. Quickstep adopts a block-based storage design, and its scheduler generates independent workloads at the block level.
3. Another focus of Quick step is to reduce the number of semi-joins by using cache-efficient filter data structures and by “pushing down” complex disjunctive predicates.

Source: "Quickstep: a data platform based on the scaling-up approach" by Jignesh M. Patel et al. (2018): - https://dl.acm.org/doi/abs/10.14778/3184470.3184471

Using Hash Tables for Minimizing Costs

1. Quickstep employes Hash Tables in order to reduce materialization costs.
2. Quickstep has two hashing phases: Build phase and Probing phase
3. In the build phase, Quickstep stores relations in the hash tables with the join predicates as the key.
4. The probe hash table operator reads blocks of the probe relation, probes the hash table, and materializes joined tuples into in-memory blocks.
5. Both the build and probe operators take advantage of block-level parallelism, and use a latch-free concurrent hash table to allow multiple workers to proceed at the same time.

Source: "Quickstep: a data platform based on the scaling-up approach" by Jignesh M. Patel et al. (2018): - https://dl.acm.org/doi/abs/10.14778/3184470.3184471



DAG execution pipeline of sample Query.
Image sourced from paper.

1. The picture of the represents the process of how Quickstep processes a given query.
2. Each node in the graph can issue independent work orders.
3. We can see that the edge from Probe Hash to Aggregation allows for pipelining.
4. When an block is fully filled using the probe hash operator, it can be independently streamed to the aggregation operator thereby achieving pipelining.

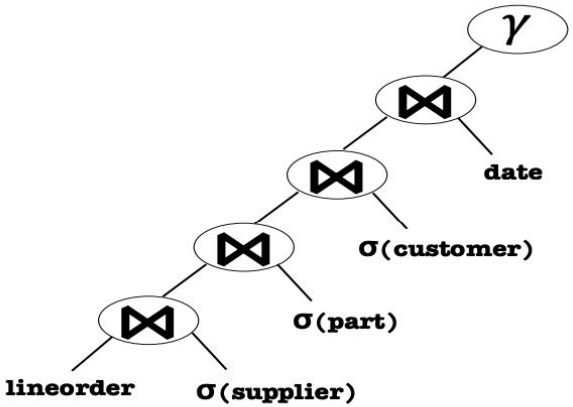
Source: "Quickstep: a data platform based on the scaling-up approach" by Jignesh M. Patel et al. (2018): - https://dl.acm.org/doi/abs/10.14778/3184470.3184471

Lookahead Information Passing (LIP)

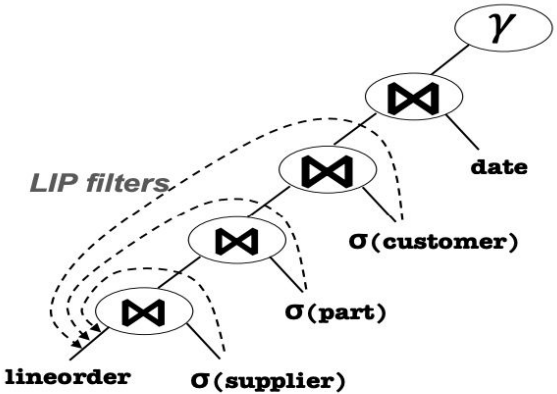
1. Quickstep optimizes query processing using something a technique called Lookahead Information Passing.
2. This measure is implemented through the use of **Bloom Filters.**
3. This enables Quickstep to embody the “drop early” and “drop fast” principles.
4. For each join in the join tree, during the hash-table build phase, we insert the build-side join keys into an LIP filter. Then, these filters are all passed to the probe-side table.

Source: "Quickstep: a data platform based on the scaling-up approach" by Jignesh M. Patel et al. (2018): - https://dl.acm.org/doi/abs/10.14778/3184470.3184471

Lookahead Information Passing (LIP)



(a) Original query plan



(c) Query plan using LIP (only)

Source: "Quickstep: a data platform based on the scaling-up approach" by Jignesh M. Patel et al. (2018): - https://dl.acm.org/doi/abs/10.14778/3184470.3184471

Quickstep: Data Compression

1. Quickstep allows for data storage in the row and columnar format, and with or without compression.
2. Quickstep employs an order-preserving dictionary compression (or encoding) scheme in order to reduce the storage overhead imposed by data.
3. Depending on the cardinality of values in a particular column within a particular block, such codes may require considerably less storage space than the original values.
4. Quickstep currently packs codes at 1, 2, and 4 byte boundaries.

Source: "Quickstep: a data platform based on the scaling-up approach" by Jignesh M. Patel et al. (2018): - https://dl.acm.org/doi/abs/10.14778/3184470.3184471

In Conclusion

1. Vectorized Algorithms serve a variety of use-cases when it comes to Database Systems.
2. Bloom filter probing for joining tables results in a significant performance improvement of 3.3x over scalar code when the Bloom filter is **cache-resident.**
3. Data Compression helps to reduce storage overhead, and processing queries directly on encoded data yields an improvement in performance as decode operations are not necessary.

References

1. Orestis Polychroniou, Arun Raghavan, and Kenneth A. Ross. 2015. Rethinking SIMD Vectorization for In-Memory Databases. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15). Association for Computing Machinery, New York, NY, USA, 1493–1508. https://doi.org/10.1145/2723372.2747645

2. Prashanth Menon, Todd C. Mowry, and Andrew Pavlo. 2017. Relaxed operator fusion for in-memory databases: making compilation, vectorization, and prefetching work together at last. Proc. VLDB Endow. 11, 1 (September 2017), 1–13. https://doi.org/10.14778/3151113.3151114

3. Harald Lang, Tobias Mühlbauer, Florian Funke, Peter A. Boncz, Thomas Neumann, and Alfons Kemper. 2016. Data Blocks: Hybrid OLTP and OLAP on Compressed Storage using both Vectorization and Compilation. In Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16). Association for Computing Machinery, New York, NY, USA, 311–326. https://doi.org/10.1145/2882903.2882925

4. Hao Jiang and Aaron J. Elmore. 2018. Boosting data filtering on columnar encoding with SIMD. In Proceedings of the 14th International Workshop on Data Management on New Hardware (DAMON '18). Association for Computing Machinery, New York, NY, USA, Article 6, 1–10. https://doi.org/10.1145/3211922.3211932

5. "Quickstep: a data platform based on the scaling-up approach" by Jignesh M. Patel et al. (2018): - https://dl.acm.org/doi/abs/10.14778/3184470.3184471

6. Orestis Polychroniou and Kenneth A. Ross. 2014. Vectorized Bloom filters for advanced SIMD processors. In Proceedings of the Tenth International Workshop on Data Management on New Hardware (DaMoN '14). Association for Computing Machinery, New York, NY, USA, Article 6, 1–6. https://doi.org/10.1145/2619228.2619234

Thanks