Charting the Design Space of Query Execution using VOILA

Tim Gubner

Peter Boncz

CWI
Database Architectures

(Some) Design Choices for HTAP/OLAP Systems

- Execution paradigm
  - Data-centric
  - Vectorized
  - Mixes (e.g. Relaxed Operator Fusion)
- Selective processing
  - None
  - Selection vector (indirection)
  - Bitmask (SIMD friendly)
  - Mixes
- Prefetching
  - Naive
  - State-machine-based (AMAC, IMV)
- Buffering
  - None
  - Intra-operator
  - Inter-operator
- Adaptivity
  - None
  - Micro (operation level)
  - Macro (operator/plan level)
- Memory layout
  - Columnar
  - Row-wise
  - Mixes (PAX)
- Granularity
  - Column
  - Vector
  - Block
  - Value
  - Partial value
- Compression
  - None
  - “Compressed Execution”
  - Storage
- Different algorithms
- `NULL` handling
- Overflow handling
- ...

(Some) Design Choices for HTAP/OLAP Systems

- Execution paradigm
  - Data-centric
  - Vectorized
  - Mixes (e.g. Relaxed Operator Fusion)
- Selective processing
  - None
  - Selection vector (indirection)
  - Bitmask (SIMD friendly)
  - Mixes
- Prefetching
  - Naive
  - State-machine-based (AMAC,)
- Buffering
  - None
  - Intra-operator
  - Inter-operator
- Adaptivity
  - None
  - Micro (operation level)
  - Macro (operator/plan level)

- Memory layout
  - Columnar
  - Row-wise
  - Mixes (PAX)
- Granularity
  - Column
  - Vector
  - Block
  - Value
  - Partial value
- Compression
  - None
  - “Compressed Execution”
  - Storage
- Different algorithms
- `NULL` handling
- Overflow handling
- ...

A Glimpse into our Knowledge

Data-centric (compiled tuple-at-a-time, e.g. Hyper)

+ Latency (of single tuple)
+ Computation
- Compilation time

Vectorized (column-slice-at-a-time, e.g. Vectorwise)

+ Parallel memory access
+ Adaptivity
+ Profiling

A Glimpse into our Knowledge

Data-centric (compiled tuple-at-a-time, e.g. Hyper)

+ Latency (of single tuple)
+ Computation
- Compilation time

Vectorized (column-slice-at-a-time, e.g. Vectorwise)

+ Parallel memory access
+ Adaptivity
+ Profiling

Interaction with Features/Techniques?

- Prefetching always good?
- SIMD always good?
- Hybrids?
- Memory layout?
- Selective processing?
- ...

Interaction with Hardware?

- Huge L3 (> 100MB)?
- Slower cores (< 2 GHz)?
- ARM? RISC-V?
- “Crazy” design decisions (e.g. no L3)?
- Accelerators?
- ...

The State-of-the-Art Exploration Process

Domain Experts

Design Choices

Implementers

Experiment

Knowledge

The State-of-the-Art Exploration Process

Design Choices

Experiment

Domain Experts

Implementers

Knowledge

Months - Years

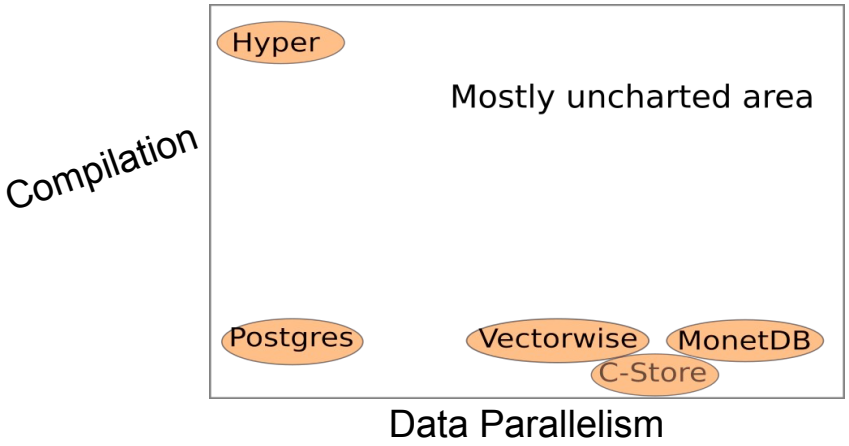
Seeking Gold in the Design Space

Bad risk/reward trade-off

- High initial investment
- Low success rate
- Vast highly dimensional space
- *Some* good points already discovered

Consequences:

- Underexplored
- Understanding = Rules of Thumb
- Vicious cycle of small improvements



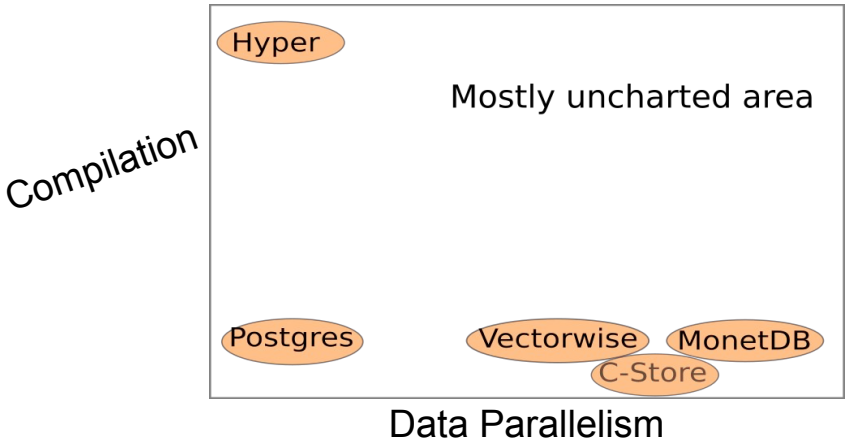
Seeking Gold in the Design Space

Bad risk/reward trade-off

- High initial investment
- Low success rate
- Vast highly dimensional space
- *Some* good points already discovered

Consequences:

- Underexplored
- Understanding = Rules of Thumb
- Vicious cycle of small improvements



Time for a Change!

The Rise of the Machines

Specification

Design Choices

Code Generator

Experiment

Knowledge

The Rise of the Machines

Specification

Design Choices

Code Generator

Experiment

Knowledge

Seconds - Minutes
(500,000x - 2,500,000x faster)

Challenges

How can we factor specific details out?

How can we synthesize them, later?

⇨Paper

VOILA

= *Variable Operator Implementation LAnguage*

Idea:

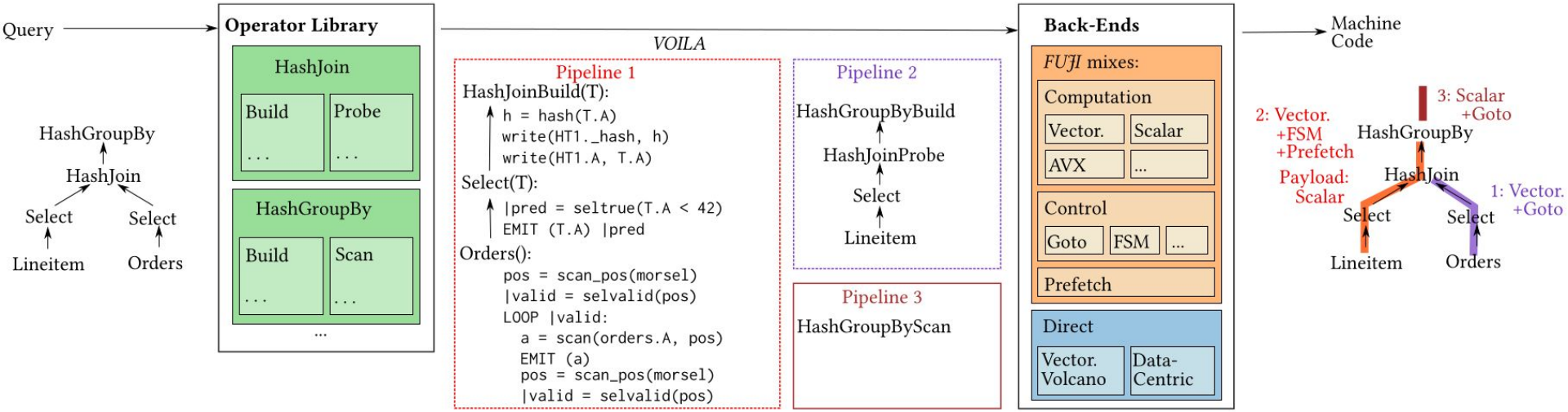
- Performance-focussed, not necessarily elegant
- Data-parallelism via algorithmic patterns

Features:

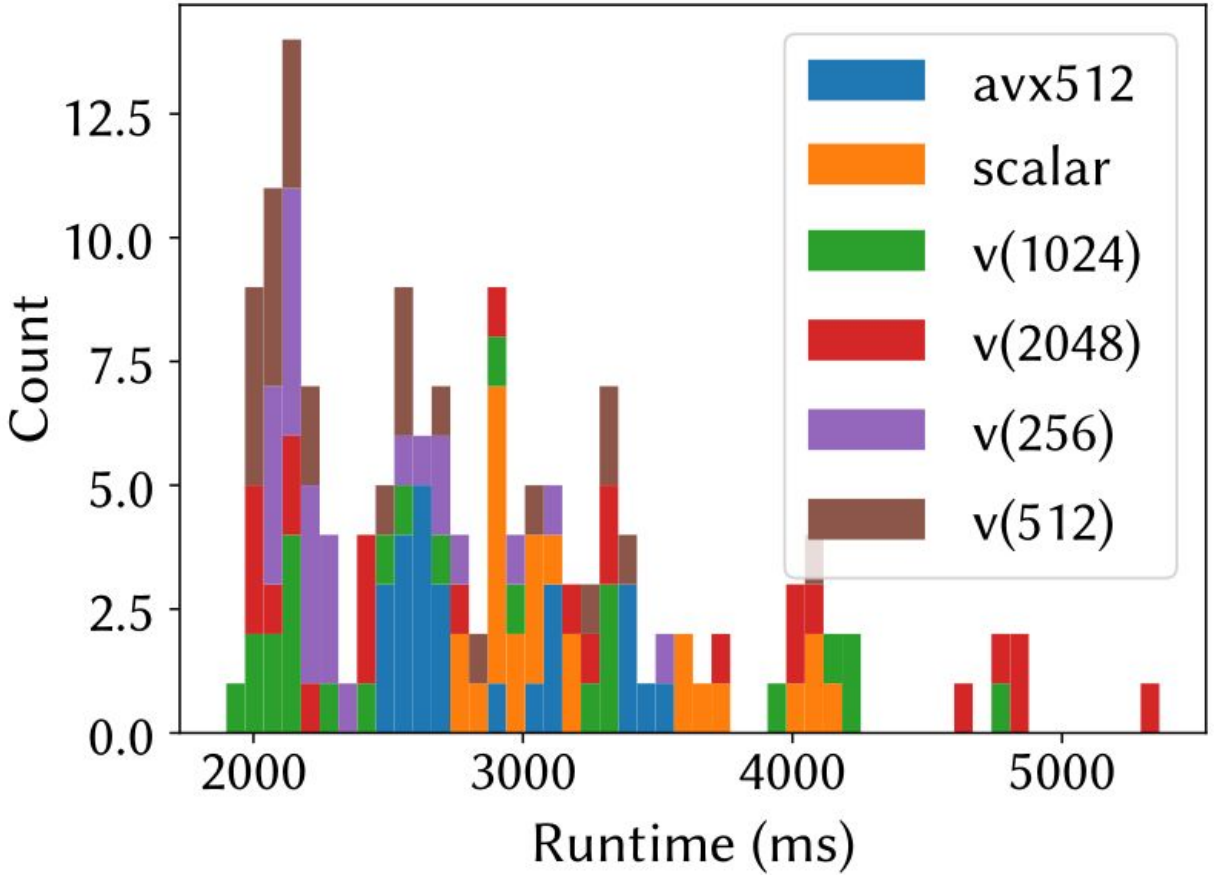
- Predicates (instead of branches)
- Fix-pointer iteration (`LOOP`)
- Special statements to move data (`EMIT`)
- Tuples (`[]` and `()`)

```
hashjoin_probe(child):
  key = child[0]
  hash = hash(key)
  bucket = bucket_lookup(HT, hash)
  hit = seltrue(ne(bucket, 0))
  LOOP |hit:
    k = gather(HT.key, bucket)
    found = seltrue(eq(k, key))
    v = gather(HT.value, bucket) |found
    EMIT (k, v) |found
    bucket = gather(HT.next, bucket)
    hit = seltrue(ne(bucket, 0))
```

VOILA-based Synthesis



Q9 (Computation)



Takeaways

With VOILA, we can:

- Encode commonly used operators
- Synthesize **many** different flavors ⇨ semi-automatic exploration
- Get top-notch performance

Future Work:

- More elegant VOILA?
- WCOJs in VOILA?
- More exploration?