Exploring Query Execution Strategies for JIT, Vectorization and SIMD

Tim Gubner, Peter Boncz

CWI

September 1, 2017

Motivation

- What will future DB engines look like?
- New hardware increasingly heterogenous: GPUs, FPGAs on the same chip, specialized CPU instructions
- For performance: exploit hardware features
- → Multi-dimensional design space

Contribution

- Shed some light into the design space
- Compare multiple implementations of TPC-H Q1:
  - Tuple-at-a-time / block-at-a-time
  - Regular / compact data types
  - Overflow checking / prevention
  - Different aggregation techniques
  - Varying aggregation table layout

SIMD (Single Instruction Multiple Data)

- One instruction processes 512 bits (AVX-512)
- Less bits per element = more elements processed per instruction

Data type width	**Parallelism**
64-bit	8
32-bit	16
16-bit	32
8-bit	64
1-bit	512

Data types

- Derived from schema:
  DECIMAL(15,2) → 64-bit integer ($32 < log_2(10^{15}) < 64$)
- Thinner data types → more data items per clock cycle

Compact data types by example

- Reduce size of data types based on actual data
- Example from TPC-H:
  - Schema: $\mathit{tax} \in \mathit{DECIMAL}(15, 2) \rightarrow$ 64-bit integer
  - Data: $\mathit{tax} \in \{0.0, 0.1, \ldots, 0.8\} \rightarrow [0, 80] \rightarrow$ 8-bit integer
  - Computation: $\mathit{tax} \cdot \mathit{tax} \rightarrow$ 8-bit integer $\cdot$ 8-bit integer $=$ 16-bit integer
- Restrict data types using domain minimum & maximum

Overflow handling

- Overflow handling is required to guarantee correctness

Detection	**Prevention**
Check via overflow CPU flag or specific code	Larger data types, overflow cannot (realistically) happen

- Overflow checking code is inefficient
- Overflow CPU flag is not SIMD friendly
- → Overflow prevention

Case study: TPC-H Q1

```
SELECT
 l_returnflag, l_linestatus,
 count(*)                            AS count_order
 sum(l_quantity)                     AS sum_qty,
 avg(l_quantity)                     AS avg_qty,
 avg(l_discount)                     AS avg_disc,
 avg(l_extendedprice)                AS avg_price,
 sum(l_extendedprice)                AS sum_base_price,
 sum(l_extendedprice*(1-l_discount)) AS sum_disc_price,
 sum(l_extendedprice*(1-l_discount)*(1+l_tax))
                                     AS sum_charge
FROM
 lineitem
WHERE
 l_shipdate <= date ’1998-12-01’ - interval ’90’ day
GROUP BY
 l_returnflag, l_linestatus
ORDER BY
 l_returnflag, l_linestatus
```

Compact data types in Q1

Expression	**Regular**	**Compact**
l_tax	64	8
l_returnflag	8	8
l_linestatus	8	8
l_extendedprice	64	32
1+l_tax	64	8
(1-l_discount)*(1+l_tax)	64	16
l_extendedprice*(1-l_discount)	64	32
l_extendedprice*(1-l_discount)*(1+l_tax)	64	64

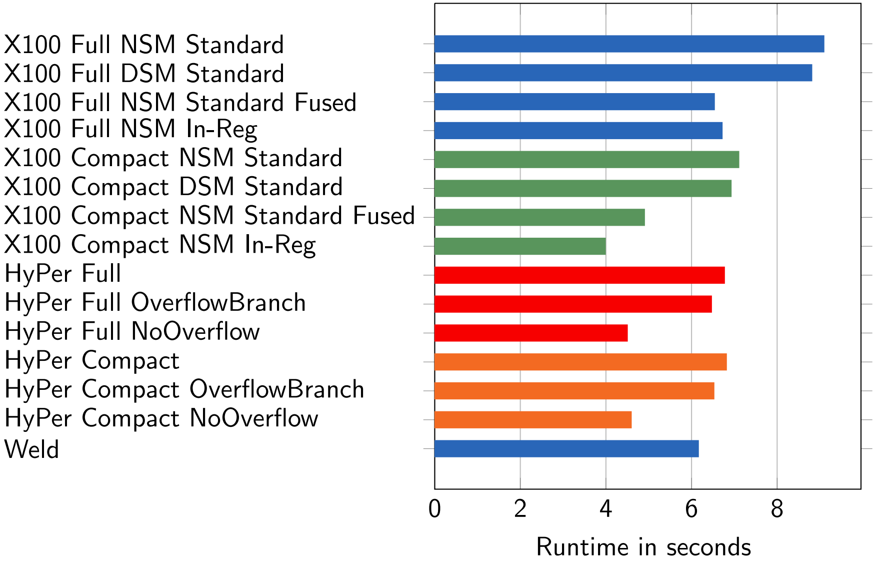
Q1 flavors

Base flavor	**Intermediate storage**	**Selection strategy**
X100	CPU cache	Sel. vector
HyPer	Registers	Branching
Handw. AVX-512	Registers	Bit mask

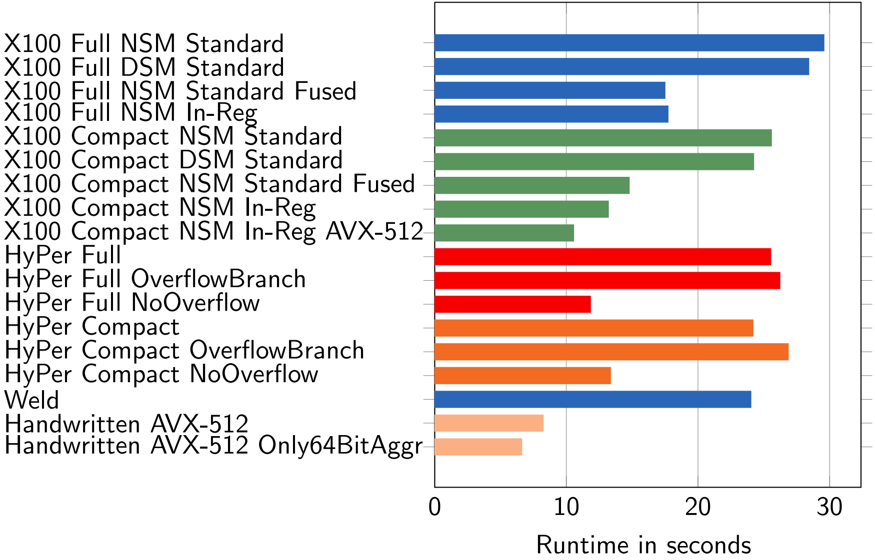
Design space explored

- X100
  - Data types: Full, Compact
  - Aggregate table layout: NSM (row-wise), DSM (column-wise)
  - Aggregation algorithm: Standard, Standard Fused, In-Reg
- HyPer
  - Data types: Full, Compact
  - Overflow: Default (set flag), OverflowBranch (branching), NoOverflow (prevention)

Q1 flavors on Sandy Bridge



Q1 flavors on Knights Landing



Takeaways

- Do not rely on schema!
- Exploit data statistics instead
- Ingredients for performance:
    - Compact data types
    - Overflow prevention
    - Adapt algorithms to take advantage of compact data types

Future work

- Even thinner data types / different representations → compressed execution
- System that generates such a query

Questions?