CS564
BitWeaving: Fast Scans for Main Memory Data Processing

Yinan Li and Jignesh M. Patel
University of Wisconsin-Madison
The Problem

• Need interactive analysis (complex SQL queries) run on large volumes of data
• New world in which business decisions are made by analytical engines
  – Speed is king
A common approach

• Disks are slow (yes – even flash is slow), but memory is fast
• Memory densities are increasing and price is dropping

Hence the rise of “in-memory” data processing for data analytics
Research Problem

• We know that for data analytics, using column stores is faster.
• But, can we go even faster than traditional column stores?
• Insights: Need to think about how the CPU sees the “data” and run data operations at the speed of the CPU
  — Recall CPU is the fastest component in the system
Column-store scans: Naïve method

• An example SQL scan query:

```
SELECT COUNT(*)
FROM Customer
WHERE age BETWEEN 20 AND 24
```

• An naïve implementation:

```
count = 0;
FOR EACH value v in column age
  IF (20 <= v and v <= 24)
    count++;
```

• Complexity: O(n). Need to run O(n) CPU instructions.

• Better method?
Encoded column values

- Domain size of column is typically small
  - Gender: Male / Female
  - Age: 0-122
  - States: 50 states
- DBMSs converts native column values to codes.
- Codes only use as many bits as are needed for fixed-length encoding.
Motivation

```
SELECT count(*)
FROM Customer
WHERE age BETWEEN 20 AND 24
```

```
count = 0;
FOR EACH value v in column age
  IF (20 <= v and v <= 24)
    count++;
```
Column-store scans: BitWeaving method

• An example SQL scan query:

```
SELECT COUNT(*)
FROM Customer
WHERE age BETWEEN 20 AND 24
```

• An BitWeaving implementation:

```
count = 0;
FOR EACH group of codes v in column age
   Evaluate 20 <= v <= 24 in parallel
   Update count;
```

• Complexity: \( O(n/w) \). \( w \): group size.
BitWeaving

• In this lecture, we introduce BitWeaving
  – A fast scan method
  – for column stores

• Fully exploits intra-cycle parallelism

• How: By “gainfully” using every bit in every processor word.
BitWeaving: Two Flavors

BitWeaving/H (Horizontal bit organization)

BitWeaving/V (Vertical bit organization)
Framework

• Targets single-table scans

• Column-scalar scan: scan on a single column
  - produce a **result bit vector**, with **one** bit for each input tuple to indicate the matching tuples

• Complex predicates in the scan: logical AND and OR operations on these **result bit vectors**
SELECT COUNT(*) FROM Customer
WHERE age BETWEEN 20 AND 24
    AND gender = Male
    AND state = Wisconsin
Framework – Example

SELECT COUNT(*) FROM Customer
WHERE age BETWEEN 20 AND 24
  AND gender = Male
  AND state = Wisconsin
SELECT COUNT(*) FROM Customer
WHERE age BETWEEN 20 AND 24
    AND gender = Male
    AND state = Wisconsin
Outline

- Motivation & Overview
- BitWeaving/H
- BitWeaving/V
- Conclusion
BitWeaving/H

• Storage layout
  – Packs codes “horizontally” into processor words
  – Uses an extra bit (delimiter bit) in each code
  – Staggers codes across words inside a segment

• Column-scalar scan
  – Parallel predicate evaluation on packed codes
BitWeaving/H - Example

Code size: 3 + 1 bits (add an extra bit, called delimiter bit)

Predicate evaluation is done on the 4 codes in parallel

Word size (16 bits)
**BitWeaving/H Example:**

**Less Than Predicate (**\( < 5 \))**

\[ X = (c_1 c_5 c_9 c_{13}) \]

\[ (Y + (X \oplus M1)) \land M2 \]

- \( M1 = 0111 \ 0111 \ 0111 \ 0111 \)
- \( M2 = 1000 \ 1000 \ 1000 \ 1000 \)

Word size (16 bits)

**Works for arbitrary code sizes & word sizes!**

**Curious about why? See our paper!**

Only use 3 instructions!

Regular integer plus
BitWeaving/H – Example...

Segment 1

- c1  c2  c3  c4  c5  c6  c7  c8  c9  c10  c11  c12  c13  c14  c15  c16  c17  c18  c19

Word size (16 bits)

Efficient production of the result bit vector with this layout!

Code size (3+1 bits)

Word size

Code size (3+1 bits)
Outline

• Motivation & Overview
• BitWeaving/H
• BitWeaving/V
• Conclusion
BitWeaving/V

• Storage layout
  – Bit-level columnar data organization, i.e. it's like a **bit-level columnar store**.

• Column-scalar scan
  – Predicate evaluation is converted to logical computation on these “words of bits”
BitWeaving/V – Storage Layout

Codes: 10, 12, 3, 6, 9, 7, 1, 0

- The first (most significant) bits of the 8 codes
- The second bits of the 8 codes
- The third bits of the 8 codes
- The last (least significant) bits of the 8 codes

Segment 1

Codes: c1, c2, c3, c4, c5, c6, c7, c8

Word 1: 1 1 0 0 1 0 0 0
Word 2: 0 1 0 1 0 1 0 0
Word 3: 1 0 1 1 0 1 0 0
Word 4: 0 0 1 0 1 1 1 0
BitWeaving/V – Column-scalar Scan

Column codes:
10, 12, 3, 6, 9, 7, 1, 0

Constant
5

α < 5?

The layout of the segment exactly matches the access pattern of column-scalar scans
BitWeaving/V – Early pruning

Column codes: 10, 12, 3, 6, 9, 7, 1, 0

Constant 5

α < 5?

Early Pruning: terminate the predicate evaluation on a segment, when all results have been determined.
Outline

• Motivation & Introduction
• BitWeaving/V
• BitWeaving/H
• Conclusions
Conclusions

BitWeaving: A new method to use all the bits in a processor word gainfully.

Two flavors: BitWeaving/H and BitWeaving/V.

BitWeaving are faster than state-of-the-art scan methods, in some cases by an order of magnitude.
Resource

• Blog article:

• Paper: