Implementing Database Operations Using SIMD Instructions

By: Jingren Zhou, Kenneth A. Ross

CSC2531: Advanced Topics in Database Systems, Fall 2011

Presented by: Ioan Stefanovici
The Problem

- Databases have become bottlenecked on CPU and memory performance
- Need to fully utilize available architectures’ features to maximize performance
  - Cache performance
    - e.g.: cache-conscious B⁺ trees, PAX, etc.
  - **Proposal**: use SIMD instructions
Single-Instruction, Multiple-Data (SIMD)
Single-Instruction, Multiple-Data (SIMD)

Let $S = \#$operands (degree of parallelism)

X0 \quad X1 \quad X2 \quad X3

Y0 \quad Y1 \quad Y2 \quad Y3

OP \quad OP \quad OP \quad OP

X0 \quad OP \quad Y0 \quad X1 \quad OP \quad Y1 \quad X2 \quad OP \quad Y2 \quad X3 \quad OP \quad Y3

Same Operation
Single-Instruction, Multiple-Data (SIMD)

- **Focus**
- **Goal**
  - Achieve speed-ups close to (or higher!) than $S$ (the degree of parallelization)
Outline

- Motivation & Problem Statement
- SIMD Instructions and Implementation Details
- Algorithm Improvements:
  - Scan algorithms
  - Index traversals
  - Join algorithms
A few points...

- Compiler auto-parallelization is difficult
  - Explicit use of SIMD instructions

- SIMD data alignment
  - Column-oriented storage

- Targets
  - Scan-like operations
  - Index traversals
  - Join algorithms
Want to perform: $X < Y$

Comparison Result Example
Comparison Result Example

- Want to perform: \( X < Y \)

```
\[
\begin{array}{cccc}
X & 0x00000001 & 0x00000003 & 0x00000004 & 0x00000007 \\
Y & 0x00000002 & 0x00000003 & 0x00000005 & 0x00000006 \\
\end{array}
\]
```

- Result:
  - \( X < Y \) for the first and third elements.
  - \( Y < X \) for the second and fourth elements.

```
\[
\begin{array}{cccc}
0xFFFFFFFF & 0x00000000 & 0xFFFFFFFF & 0x00000000 \\
\end{array}
\]
```

- SIMD_bit_vector:
  - 0x1010
Typical scan:

```plaintext
for i = 1 to N{
    if (condition(x[i])) then
        process1(y[i]);
    else
        process2(y[i]);
}
```

<table>
<thead>
<tr>
<th>x (condition)</th>
<th>y (data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>x1</td>
<td>y1</td>
</tr>
<tr>
<td>x2</td>
<td>y2</td>
</tr>
<tr>
<td>x3</td>
<td>y3</td>
</tr>
<tr>
<td>x4</td>
<td>y4</td>
</tr>
<tr>
<td>x5</td>
<td>y5</td>
</tr>
<tr>
<td>x6</td>
<td>y6</td>
</tr>
</tbody>
</table>
Typical SIMD scan:

```plaintext
for i = 1 to N step S {
    Mask[1..S] = SIMD_condition(x[i..i+S-1]);
    SIMD_Process(Mask[1..S], y[i..i+S-1]);
}
```

For $S=4$

```
x (condition)     y (data)
```

```
...  x1  x2  x3  x4  x5  x6  ...
```

```
...  y1  y2  y3  y4  y5  y6  ...
```

...
SIMD Return First Match

SIMD_Process(mask[1..S], y[1..S]){
    V = SIMD_bit_vector(mask);
    /* V = number between 0 and 2^S-1 */
    if (V != 0){
        for j = 1 to S
            if ( (V >> (S-j)) & 1 ) /* jth bit */
                { result = y[j]; return; }
    }
}
Scan: Return All Matches

- **SIMD All Matches Alternative 1**

  ```
  SIMD_Process(mask[1..S], y[1..S]){
    V = SIMD_bit_vector(mask);
    /* V = number between 0 and 2^S-1 */
    if (V != 0){
      for j = 1 to S
        if ( (V >> (S-j)) & 1 ) /* jth bit */
          { result[pos++] = y[j]; }
    }
  }
  ```

- **SIMD All Matches Alternative 2**

  ```
  SIMD_Process(mask[1..S], y[1..S]){
    V = SIMD_bit_vector(mask);
    /* V = number between 0 and 2^S-1 */
    if (V != 0){
      for j = 1 to S
        tmp = (V >> (S-j)) & 1 /* jth bit */
        result[pos] = y[j];
        pos += tmp; }
  }
  ```
Scan: Return All Matches Performance

Table Scan

- Elapsed time (milliseconds)
- Table Cardinality (million)

- Original All-Match
- SIMD Alternative 1
- SIMD Alternative 2

Table Scan

- Elapsed time (milliseconds)
- Selectivity

- Original All-Match
- SIMD Alternative 1
- SIMD Alternative 2

Searching a table with 1 million records.
Predicate selectivity 0.2

- Elapsed Time (milliseconds)

- Original All-Match
- SIMD Alternative 1
- SIMD Alternative 2

- Other Cost
- Branch Misprediction Penalty
Index Structures ($B^+$ trees)

Log₂ (n) Height

Example of a $B^+$-tree internal node

(Source: Wikipedia)
Internal Node Search

- 5 Ways to Search
  - Binary Search (SISD)
  - SIMD Binary Search
  - SIMD Sequential Search 1
  - SIMD Sequential Search 2
  - Hybrid Search
**Internal Node Search**

- Naive SIMD Binary Search (looking for “4”)

| 1 | 3 | 4 | 5 | 7 | 8 | 10 | 13 | 14 | 17 | 19 | 20 | 23 | 24 | 25 | 32 |
Internal Node Search

- Naive SIMD Binary Search (looking for “4”)

```
1 3 4 5 7 8 10 13 14 17 19 20 23 24 25 32
```

```
0 0 0 0
```
Internal Node Search

- Naive SIMD Binary Search (looking for “4”)

```
1 3 4 5 7 8 10 13 14 17 19 20 23 24 25 32
```

Got it!
Internal Node Search

- SIMD Sequential Search 1 (looking for “4”)
Internal Node Search

- SIMD Sequential Search 1 (looking for “4”)

```
1 3 4 5 7 8 10 13 14 17 19 20 23 24 25 32
```

Total ≤ 4:

```
1 1 1 0
3
```
Internal Node Search

- SIMD Sequential Search 1 (looking for “4”)

```
1 3 4 5 7 8 10 13 14 17 19 20 23 24 25 32
```

Total ≤ 4:
```
1 1 1 0
```

- SIMD Sequential Search 2 (looking for “4”)

```
1 3 4 5 7 8 10 13 14 17 19 20 23 24 25 32
```

Total ≤ 4:
```
0 0 0 0
```
Internal Node Search

- SIMD Sequential Search 1 (looking for “4”)

Total ≤ 4:
3

0 0 0 0
Internal Node Search

- SIMD Sequential Search 1 (looking for “4”)

| 1 | 3 | 4 | 5 | 7 | 8 | 10 | 13 | 14 | 17 | 19 | 20 | 23 | 24 | 25 | 32 |
|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|

Total ≤ 4:
3

Got it!
Internal Node Search

- SIMD Sequential Search 2 (looking for “4”)

| 1 | 3 | 4 | 5 | 7 | 8 | 10 | 13 | 14 | 17 | 19 | 20 | 23 | 24 | 25 | 32 |
Internal Node Search

- SIMD Sequential Search 2 (looking for “4”)

1 3 4 5 7 8 10 13 14 17 19 20 23 24 25 32

Total ≤ 4: 3

1 1 1 0

Is there a key > the search key in the SIMD unit? Yes! Got it!
Internal Node Search

- SIMD Sequential Search 2 (looking for “4”)

```
1 3 4 5 7 8 10 13 14 17 19 20 23 24 25 32
```

Total $\leq 4$: 3

- Is there a key $> \text{the search key}$ in the SIMD unit? Yes! Got it!

- Pro: processes fewer keys (50% fewer on average)
- Con: extra conditional test
Internal Node Search

- Hybrid Search (looking for “4”)  
  
  Pick some L (say L = 3)

1 3 4 5 7 8 10 13 14 17 19 20 23 24 25 32 ...
Internal Node Search

- Hybrid Search (looking for “4”)  

Pick some $L$ (say $L = 3$)

```
1 3 4   5 7 8   10 13 14   17 19 20   23 24 25 32 ...
```

Binary Search on last element of each “segment”
Hybrid Search (looking for “4”) Pick some L (say L = 3)

Binary Search on last element of each “segment”

Sequential SIMD scan inside the correct segment
Internal Node Search Performance

![Graph showing search performance for different methods. The x-axis represents the number of keys per node, and the y-axis represents the time taken for a 10K random search in milliseconds. The graph compares Sequential Search, Sequential Search 2, Binary Search, Naive SIMD Search, and Hybrid Search.]
Internal Node Search – Branch Misprediction

10K random search over a node with 128 keys

10K random search over a node with 512 keys
Nested Loop Join – $O(n^2)$

- Nested Loop

Outer Loop

Inner Loop

<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
<th>1</th>
<th>16</th>
<th>9</th>
<th>3</th>
<th>18</th>
<th>2</th>
<th>34</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td>80</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Nested Loop Join – $O(n^2)$

- SISD Algorithm

Outer Loop

Inner Loop

Iterate 1 at a time
Nested Loop Join – $O(n^2)$

- **SIMD Duplicate-Outer**

  Fix & duplicate $S$ times

  Outer Loop: 2, 4, 1, 16, 9, 3, 18, 2, 34, 80

  Inner Loop: 5, 4, 80, 8, 9, 7, 10

  Iterate $S$ at a time
Nested Loop Join – $O(n^2)$

- **SIMD Duplicate-Inner**

  Iterate $S$ at a time

  - Outer Loop
    - 2
    - 4
    - 1
    - 16
    - 9
    - 3
    - 18
    - 2
    - 34
    - 80
  
  - Inner Loop
    - 5
    - 4
    - 80
    - 8
    - 9
    - 7
    - 10

  Fix & duplicate $S$ times
Nested Loop Join – $O(n^2)$

- SIMD Rotate-Inner (Rotate & Compare S times)

Iterate S at a time

Outer Loop

Inner Loop

Iterate S at a time
Nested Loop Join – Performance

Queries

Q1. SELECT ... FROM R, S WHERE R.Key = S.Key (integer)
Q2. SELECT ... FROM R, S WHERE R.Key = S.Key (floating-point)
Q3. SELECT ... FROM R, S WHERE R.Key < S.Key < 1.01 * R.Key
Q4. SELECT ... FROM R, S WHERE R.Key < S.Key < R.Key + 5
Nested Loop Join Branch Misprediction

**Query 3**

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>Duplicate-outer</th>
<th>Duplicate-inner</th>
<th>Rotate-inner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed Time (seconds)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Other Cost
- Branch Misprediction Penalty

**Query 4**

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>Duplicate-outer</th>
<th>Duplicate-inner</th>
<th>Rotate-inner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed Time (seconds)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Other Cost
- Branch Misprediction Penalty
Conclusion

- Thank you!

Questions