FUTURE VECTOR MICROPROCESSOR EXTENSIONS FOR DATA AGGREGATIONS

Timothy Hayes, Oscar Palomar, Osman Unsal, Adrian Cristal, Mateo Valero

timothy.hayes@bsc.es
Motivation

- Data generation growing at an exponential rate
- Increasing demand to summarise/aggregate data quickly
- Since ~2005, frequency scaling no longer viable
- Explicit forms of parallelism must be used
- Data-level parallelism (DLP) is excellent when available
  - Vector SIMD ISAs are highly efficient
  - Compact representation – Implicit parallelism – Scalable
  - Energy-efficient hardware implementations
- Vector SIMD ISA perfect when DLP is regular
- Many algorithms need transformations to be regular
- Transformations often hurt performance
Contributions

- We examine applicability of vector SIMD to aggregations
- Propose and evaluate different algorithms
  1. With transformations to use typical vector instructions
  2. Vectorise directly using our novel vector instructions
- Evaluate with many datasets
  - Five unique distributions
  - Twenty-two cardinalities
- Speedups between 2.7x and 7.6x over scalar baseline

⚠️ This work is an extension to our HPCA-21 article
  - I will skip many things due to time constraints
Presentation Contents

I. Motivation

II. What is Data Aggregation?

III. Experimental Setup

IV. Algorithms
   1. Scalar Baseline
   2. Polytable
   3. Sorted Reduce
   4. Monotable
   5. Partially Sorted Monotable
   6. Summary

V. Conclusions
What is a Data Aggregation?

- Frequently occurring operation found in
  - SQL GROUP BY queries
  - MapReduce
  - Statistical Languages
  - OLAP Cubes
- Reduction of key-value pairs
- Aggregation function, e.g.
  - SUM
  - MINIMUM
  - MAXIMUM
  - AVERAGE
What is a Data Aggregation?

Future Vector Microprocessor Extensions for Data Aggregations
Presentation Contents

I. Motivation

II. What is Data Aggregation?

III. Experimental Setup

IV. Algorithms
   1. Scalar Baseline
   2. Polytable
   3. Sorted Reduce
   4. Monotable
   5. Partially Sorted Monotable
   6. Summary

V. Conclusions

Future Vector Microprocessor Extensions for Data Aggregations
Query and Algorithms

Scalar baseline – no vector instructions

Regular DLP – Typical vector instructions
A. Polytable
B. Standard Sorted Reduce [not in presentation] ▲

Irregular DLP – Novel vector instructions
A. Advanced Sorted Reduce
B. Monotable
C. Partially Sorted Monotable

```sql
SELECT key, COUNT(*), SUM(value)
FROM table GROUP BY key
```
Datasets: Five Distributions

1. **Uniform**

```
4 8 2 3 6 7 4 4 1 6 6 7 1 5 2 4 8 1 3 1 2 2 3 3 7 8 5 5 7 6 5 8
```

2. **Sorted**

```
1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4 5 5 5 5 6 6 6 6 7 7 7 7 8 8 8 8
```

3. **Sequential**

```
1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8
```

4. **Heavy Hitter**

```
3 8 2 3 6 7 3 4 1 3 6 3 3 5 3 4 8 3 3 1 3 2 3 3 7 3 3 5 7 3 5 8
```

5. **Zipfian**

```
6 8 4 3 2 1 2 2 3 4 8 7 6 1 2 7 3 1 2 5 4 1 5 1 3 1 1 1 1 2 1 5
```
Datasets: Cardinalities

- Number of unique keys within dataset, e.g.

- \( N = 10,000,000 \)
- \( C = 4 \rightarrow 10,000,000 \)
- Grouped into four cardinality divisions
  1. Low cardinalities – many repeated keys
  2. Low-normal cardinalities
  3. High-normal cardinalities
  4. High cardinalities – many unique keys
Simulation Framework

- Custom Simulation Framework
  - PTLsim – 32 nm Westmere microarchitecture
  - DRAMSim2 – DDR3-1333

- Extended vector SIMD support
  - Heavily influenced from classical vector machines, e.g. CRAY-1
  - Emphasis on integer operations
  - 16x vector registers with 64x 64bit elements
  - Pipelined functional units with 4x lockstepped parallel lanes
  - Masked operations
  - Indexed memory operations, i.e. gather/scatter
  - Integrated in out-of-order superscalar pipeline
Presentation Contents

I. Motivation

II. What is Data Aggregation?

III. Experimental Setup

IV. Algorithms

   1. Scalar Baseline
   2. Polytable
   3. Sorted Reduce
   4. Monotable
   5. Partially Sorted Monotable
   6. Summary

V. Conclusions
Scalar Baseline

keys
1 5 5 3

values
27 19 43 31

table
0
0
31
0
0

Future Vector Microprocessor Extensions for Data Aggregations
Scalar Baseline – Results
Presentation Contents

I. Motivation

II. What is Data Aggregation?

III. Experimental Setup

IV. Algorithms
   1. Scalar Baseline
   2. Polytable
   3. Sorted Reduce
   4. Monotable
   5. Partially Sorted Monotable
   6. Summary

V. Conclusions
Polytable

process m key-values

keys
1 5 5 3
values
27 19 43 31
table

0
0
0
0
0

gather-modify-scatter conflict
### Polytable

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27</td>
</tr>
<tr>
<td>5</td>
<td>19</td>
</tr>
<tr>
<td>5</td>
<td>43</td>
</tr>
<tr>
<td>3</td>
<td>31</td>
</tr>
</tbody>
</table>

$m$ local tables

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>27</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>43</td>
<td>19</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Polytable – Results

### Scalar

- uniform
- sorted
- sequential
- hhitter
- zipf

### Polytable

- uniform
- sorted
- sequential
- hhitter
- zipf

---

Future Vector Microprocessor Extensions for Data Aggregations
Presentation Contents

I.  Motivation
II. What is Data Aggregation?
III. Experimental Setup
IV. Algorithms
   1. Scalar Baseline
   2. Polytable
   3. Sorted Reduce
   4. Monotable
   5. Partially Sorted Monotable
   6. Summary
V.  Conclusions
Sorted Reduce

Our new sorting algorithm from HPCA-21
- Based on vectorised radix sort
- Uses novel vector SIMD instructions
- Avoids gather-modify-scatter conflicts
- Vector Prior Instances (VPI)

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 1 5 3</td>
<td>27 19 43 31</td>
</tr>
</tbody>
</table>
Sorted Reduce

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 1 5 3</td>
<td>27 19 43 31</td>
</tr>
</tbody>
</table>

least significant element

most significant element

VPI

Future Vector Microprocessor Extensions for Data Aggregations
Sorted Reduce

- **sort**
- **keys**: 5, 5, 3, 1
- **values**: 27, 43, 31, 19
- **output**: 19, 31, 70
  - 1 + reduce
  - 3 + reduce
  - 5 + reduce

Future Vector Microprocessor Extensions for Data Aggregations
Sorted Reduce – Results

**scalar**

<table>
<thead>
<tr>
<th>cycles per tuple</th>
<th>low</th>
<th>low-normal</th>
<th>high-normal</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>19</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>38</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>76</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>152</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>305</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>610</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>1,220</td>
<td>4</td>
<td>9</td>
<td>19</td>
<td>38</td>
</tr>
<tr>
<td>2,441</td>
<td>4</td>
<td>9</td>
<td>19</td>
<td>38</td>
</tr>
<tr>
<td>4,882</td>
<td>4</td>
<td>9</td>
<td>19</td>
<td>38</td>
</tr>
<tr>
<td>9,765</td>
<td>4</td>
<td>9</td>
<td>19</td>
<td>38</td>
</tr>
<tr>
<td>19,531</td>
<td>4</td>
<td>9</td>
<td>19</td>
<td>38</td>
</tr>
<tr>
<td>39,062</td>
<td>4</td>
<td>9</td>
<td>19</td>
<td>38</td>
</tr>
<tr>
<td>78,125</td>
<td>4</td>
<td>9</td>
<td>19</td>
<td>38</td>
</tr>
<tr>
<td>156,250</td>
<td>4</td>
<td>9</td>
<td>19</td>
<td>38</td>
</tr>
</tbody>
</table>

**advanced sorted reduce**

<table>
<thead>
<tr>
<th>cycles per tuple</th>
<th>low</th>
<th>low-normal</th>
<th>high-normal</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>19</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>38</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>76</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>152</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>305</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>610</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>1,220</td>
<td>4</td>
<td>9</td>
<td>19</td>
<td>38</td>
</tr>
<tr>
<td>2,441</td>
<td>4</td>
<td>9</td>
<td>19</td>
<td>38</td>
</tr>
<tr>
<td>4,882</td>
<td>4</td>
<td>9</td>
<td>19</td>
<td>38</td>
</tr>
<tr>
<td>9,765</td>
<td>4</td>
<td>9</td>
<td>19</td>
<td>38</td>
</tr>
<tr>
<td>19,531</td>
<td>4</td>
<td>9</td>
<td>19</td>
<td>38</td>
</tr>
<tr>
<td>39,062</td>
<td>4</td>
<td>9</td>
<td>19</td>
<td>38</td>
</tr>
<tr>
<td>78,125</td>
<td>4</td>
<td>9</td>
<td>19</td>
<td>38</td>
</tr>
<tr>
<td>156,250</td>
<td>4</td>
<td>9</td>
<td>19</td>
<td>38</td>
</tr>
</tbody>
</table>

**Future Vector Microprocessor Extensions for Data Aggregations**

23
Presentation Contents

I. Motivation

II. What is Data Aggregation?

III. Experimental Setup

IV. Algorithms
   1. Scalar Baseline
   2. Polytable
   3. Sorted Reduce
   4. **Monotable**
   5. Partially Sorted Monotable
   6. Summary

V. Conclusions
The **Polytable** algorithm needs to replicate tables
- Avoids **gather-modify-scatter** conflicts
- Hurts performance

The **Sorted Reduce** algorithm uses VSR Sort
- VSR Sort uses VPI to resolve **gather-modify-scatter** conflicts
- Could VPI also be used to optimise **Polytable**?

**VPI** is not sufficient, but…
- Hardware could be reused
- Create similar-style but different instruction

**Vector Group Aggregate:** **SUM (VGAsum)**
- Similar to VPI but uses second vector of values
- Vectorise scalar baseline without transformations
### Monotable

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>14</td>
</tr>
</tbody>
</table>

- **Least significant element**: 2
- **Most significant element**: 2

**VGAsum**

- The sum of the most significant element and the least significant element.
- The output is the sum of the key and the value.

**Notes**:
- VGAsum is the sum of the key and the value.
- The output is the result of the sum.

---

*Future Vector Microprocessor Extensions for Data Aggregations*
Monotable – Results

scalar

- uniform
- sorted
- sequential
- hhitter
- zipf

monotable

- uniform
- sorted
- sequential
- hhitter
- zipf

cycles per tuple

Future Vector Microprocessor Extensions for Data Aggregations
Presentation Contents

I. Motivation
II. What is Data Aggregation?
III. Experimental Setup
IV. Algorithms
   1. Scalar Baseline
   2. Polytable
   3. Sorted Reduce
   4. Monotable
   5. Partially Sorted Monotable
   6. Summary
V. Conclusions

Future Vector Microprocessor Extensions for Data Aggregations
Partially Sorted Monotable

- Losing locality hurts performance
- Fully sorting can have a high overhead
- VSR Sort has $O(k.n)$ complexity
- If we reduce the ‘$k$’, we reduce the overhead

![Diagram showing key and k bits with sorting on MSBs (partition) and ignoring LSBs.]

Future Vector Microprocessor Extensions for Data Aggregations
Partially Sorted Monotable – Results

Future Vector Microprocessor Extensions for Data Aggregations
Presentation Contents

I. Motivation

II. What is Data Aggregation?

III. Experimental Setup

IV. Algorithms
   1. Scalar Baseline
   2. Polytable
   3. Sorted Reduce
   4. Monotable
   5. Partially Sorted Monotable

6. Summary

V. Conclusions
Summary – Best Speedups Overall

Future Vector Microprocessor Extensions for Data Aggregations
Summary – Best Speedups Overall

Future Vector Microprocessor Extensions for Data Aggregations
Presentation Contents

I. Motivation

II. What is Data Aggregation?

III. Experimental Setup

IV. Algorithms
   1. Scalar Baseline
   2. Polytable
   3. Sorted Reduce
   4. Monotable
   5. Partially Sorted Monotable
   6. Summary

V. Conclusions
Conclusions

- Aggregating data quickly is important
- DLP & SIMD is an attractive way to accelerate it
- Aggregation algorithms are simple but DLP is irregular
- We proposed various algorithms
  A. Use transformations and typical vector SIMD instructions
  B. Avoid transformations using our novel vector instructions
- Evaluated using many data distributions and cardinalities
- Speedups between 2.7x and 7.6x over scalar baseline
- Best solution is dependent on input characteristics