ISCA-43 (2016)

Future Vector Microprocessor Extensions for Data Aggregations

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

timothy.hayes@bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

UPC

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 <u>regular</u>
- Many algorithms need <u>transformations</u> 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
  - *VSR Sort: A Novel Vectorised Sorting Algorithm. (2015) Hayes et al.*
- I will skip many things due to time constraints

Presentation Contents

I. Motivation

II. What is Data Aggregation?

III. Experimental Setup

IV. Algorithms

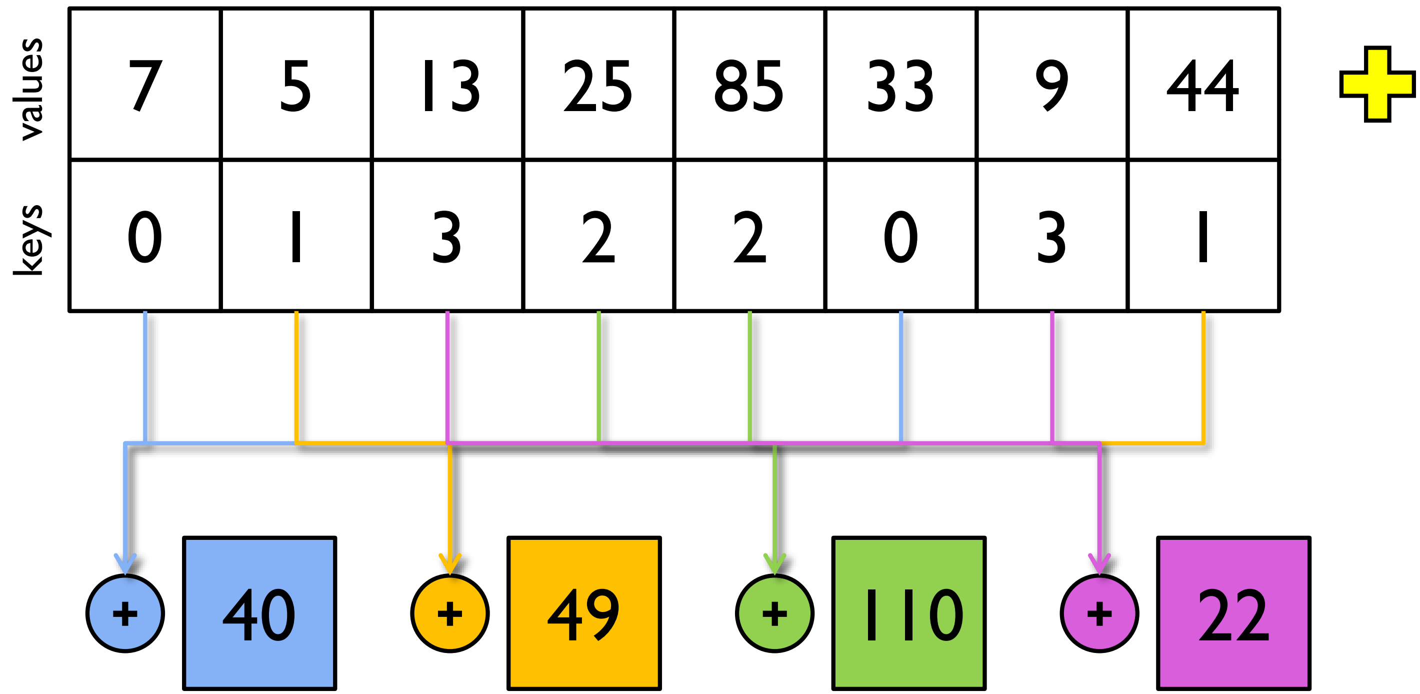
1. Scalar Baseline
2. Polytable
3. Sorted Reduce
4. Monotable
5. Partially Sorted Monotable
6. Summary

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?



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

Query and Algorithms

```
SELECT key, COUNT(*), SUM(value)
FROM table GROUP BY key
```

- 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

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.

1	1	1	1	1	1	1	1	⟵	N=8, C=1
1	1	2	1	2	2	2	1	⟵	N=8, C=2
2	1	1	3	4	2	4	1	⟵	N=8, C=4
3	2	8	5	4	6	1	7	⟵	N=8, C=8

- N = 10,000,000
- C = 4 → 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**

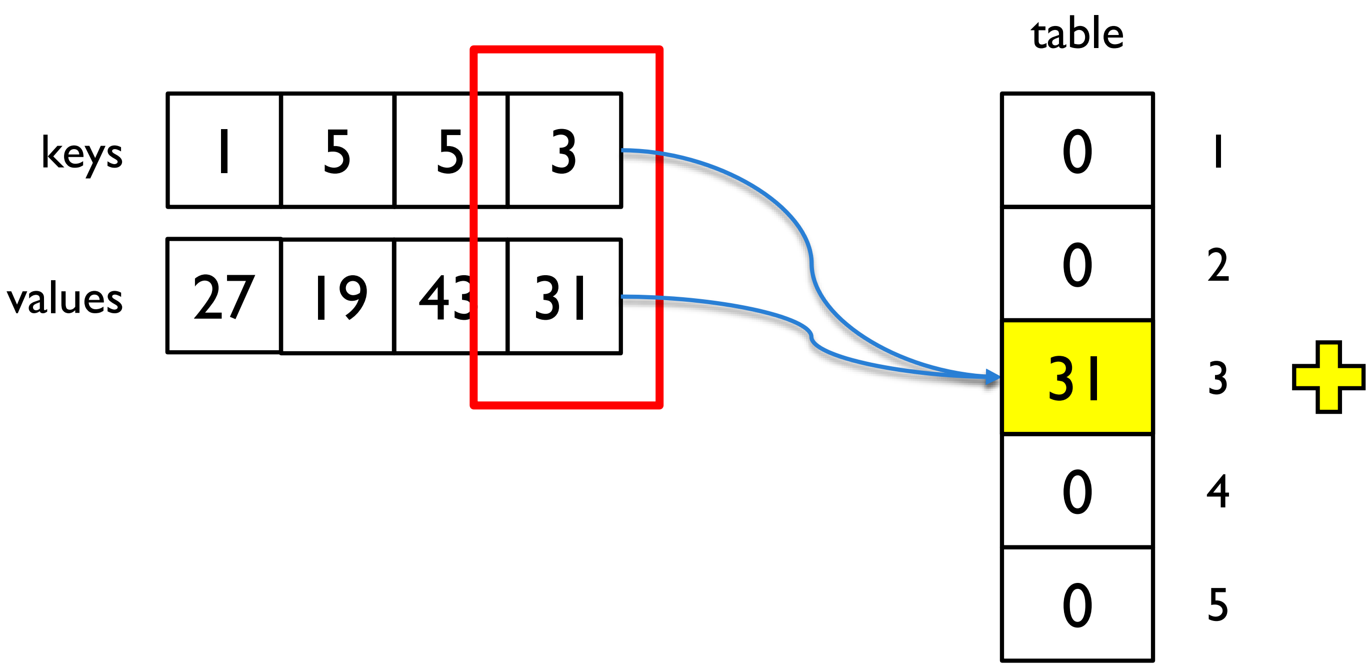
Simulation Framework

- 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

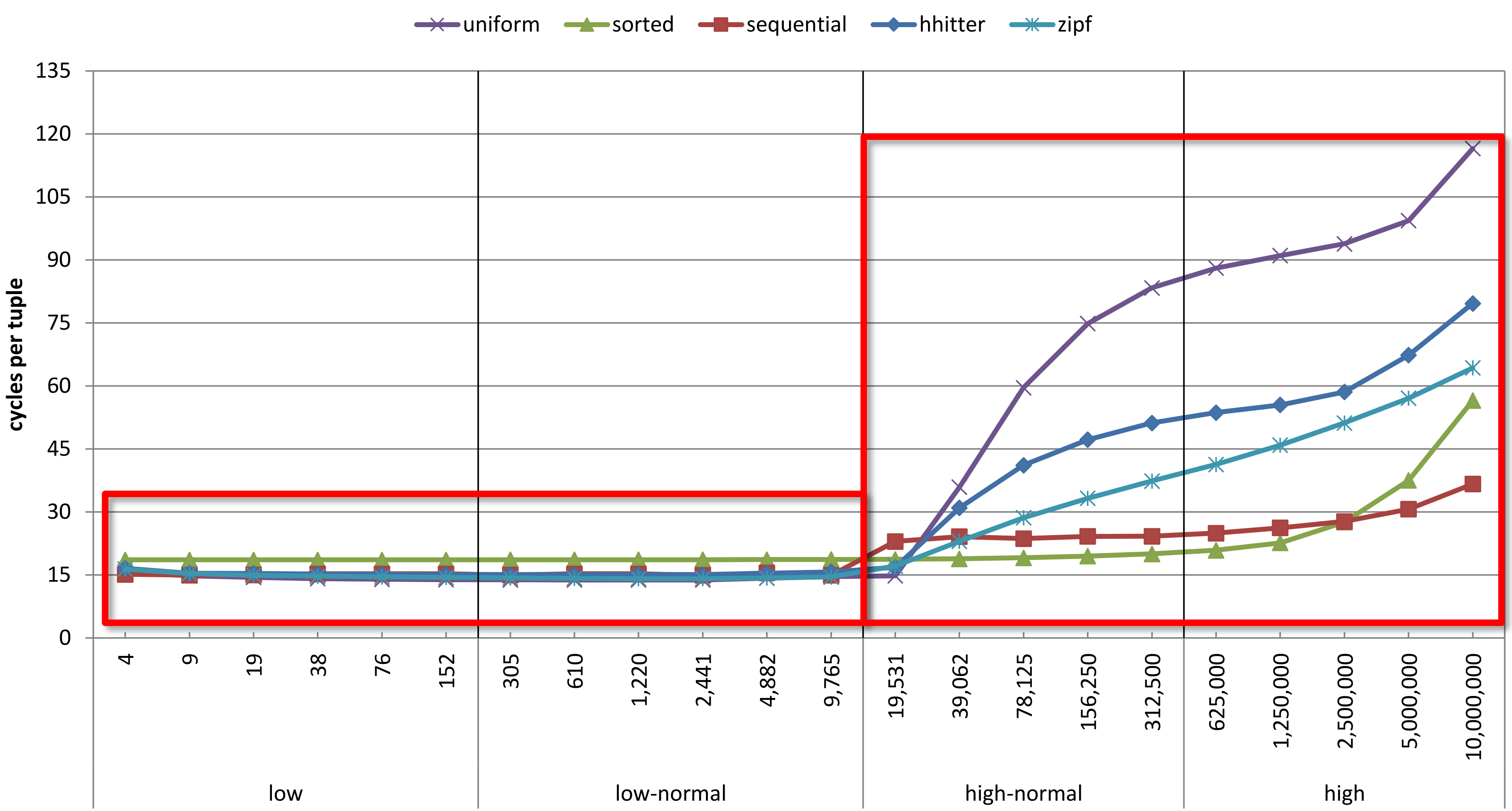
Presentation Contents

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



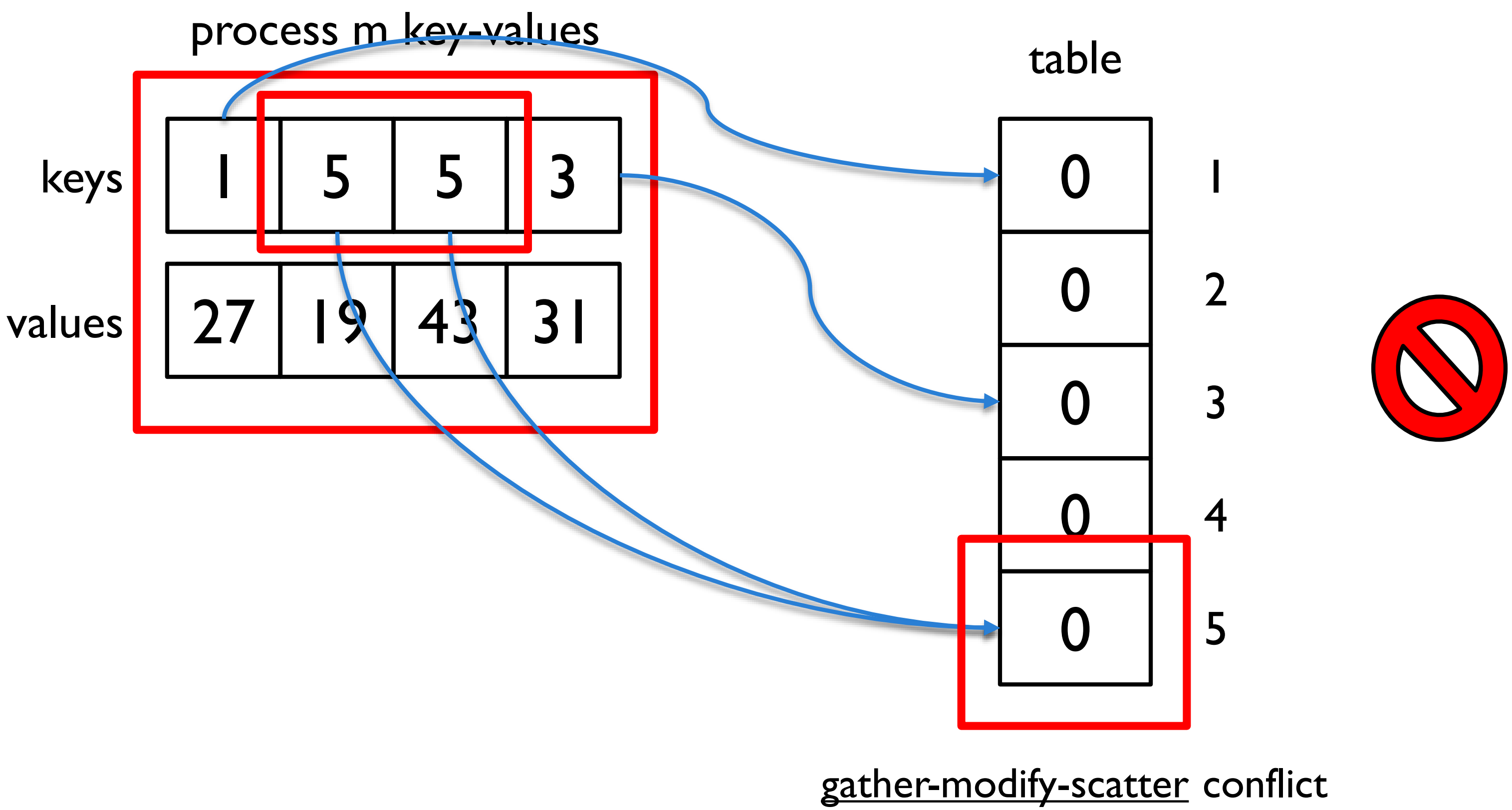
Scalar Baseline – Results



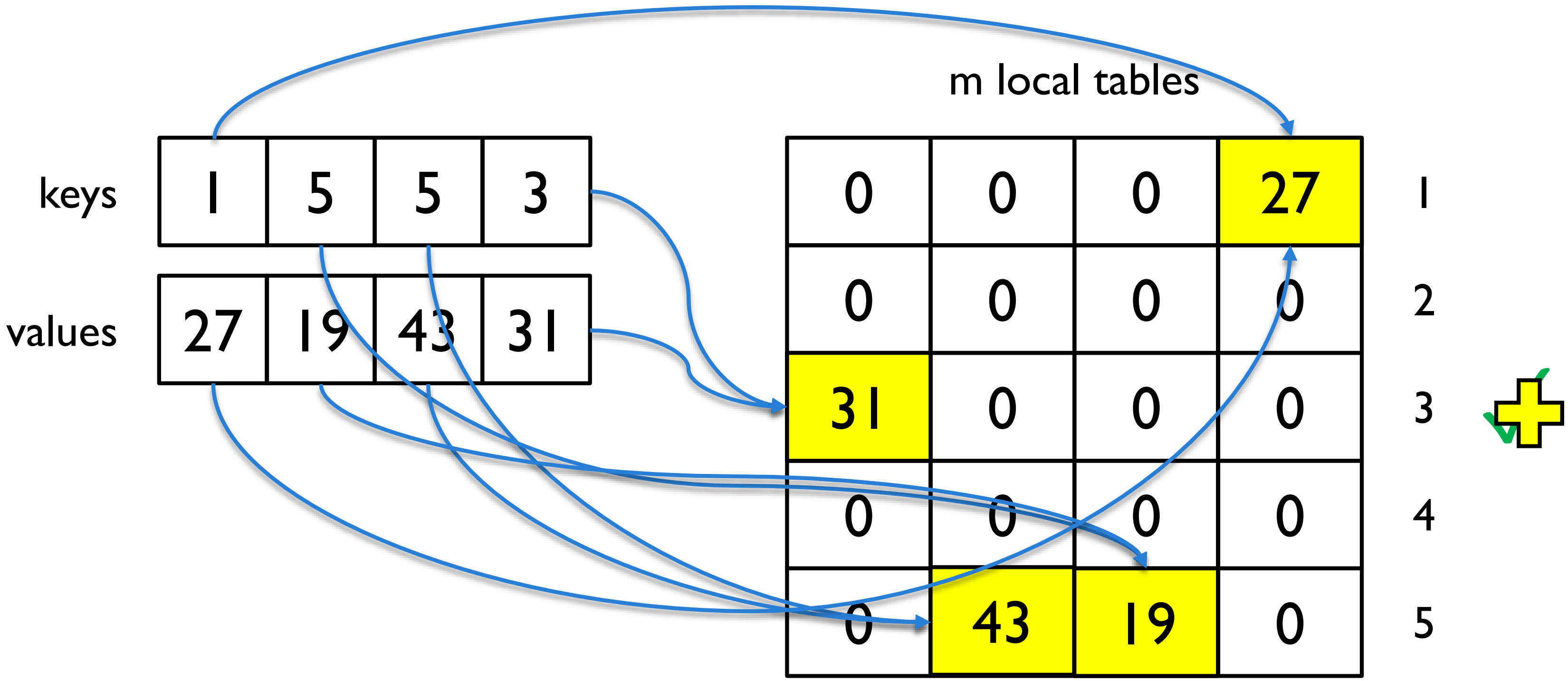
Presentation Contents

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

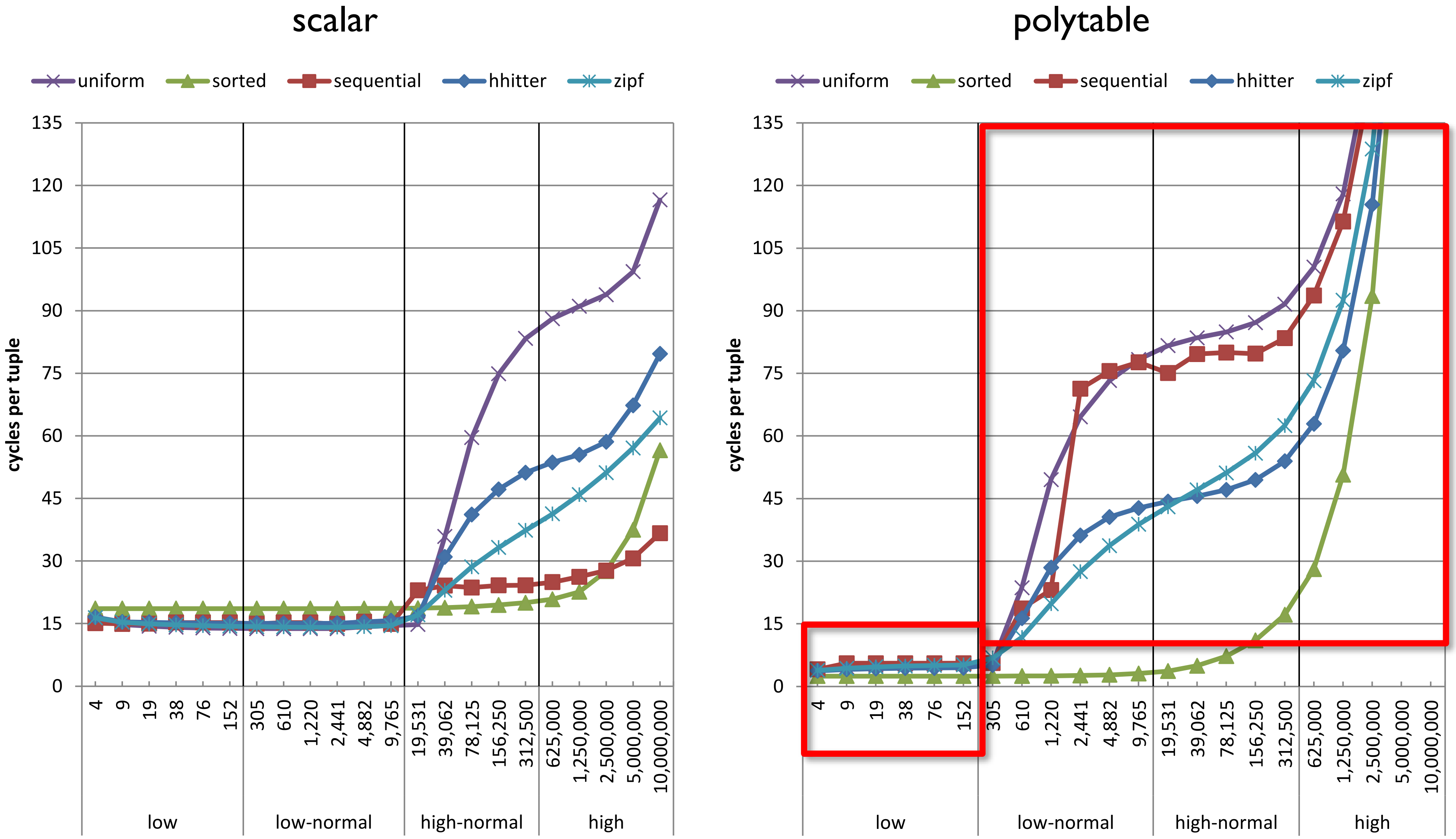

process m key-values
keys
1
5
5
3
values
27
19
43
31
table
0
0
0
0
0
1
2
3
4
5
gather-modify-scatter conflict

Polytable



m local tables
keys
1
5
5
3
values
27
19
43
31
0
0
0
27
1
0
0
0
0
2
31
0
0
0
3
0
0
0
0
4
0
43
19
0
5

Polytable – Results



Presentation Contents

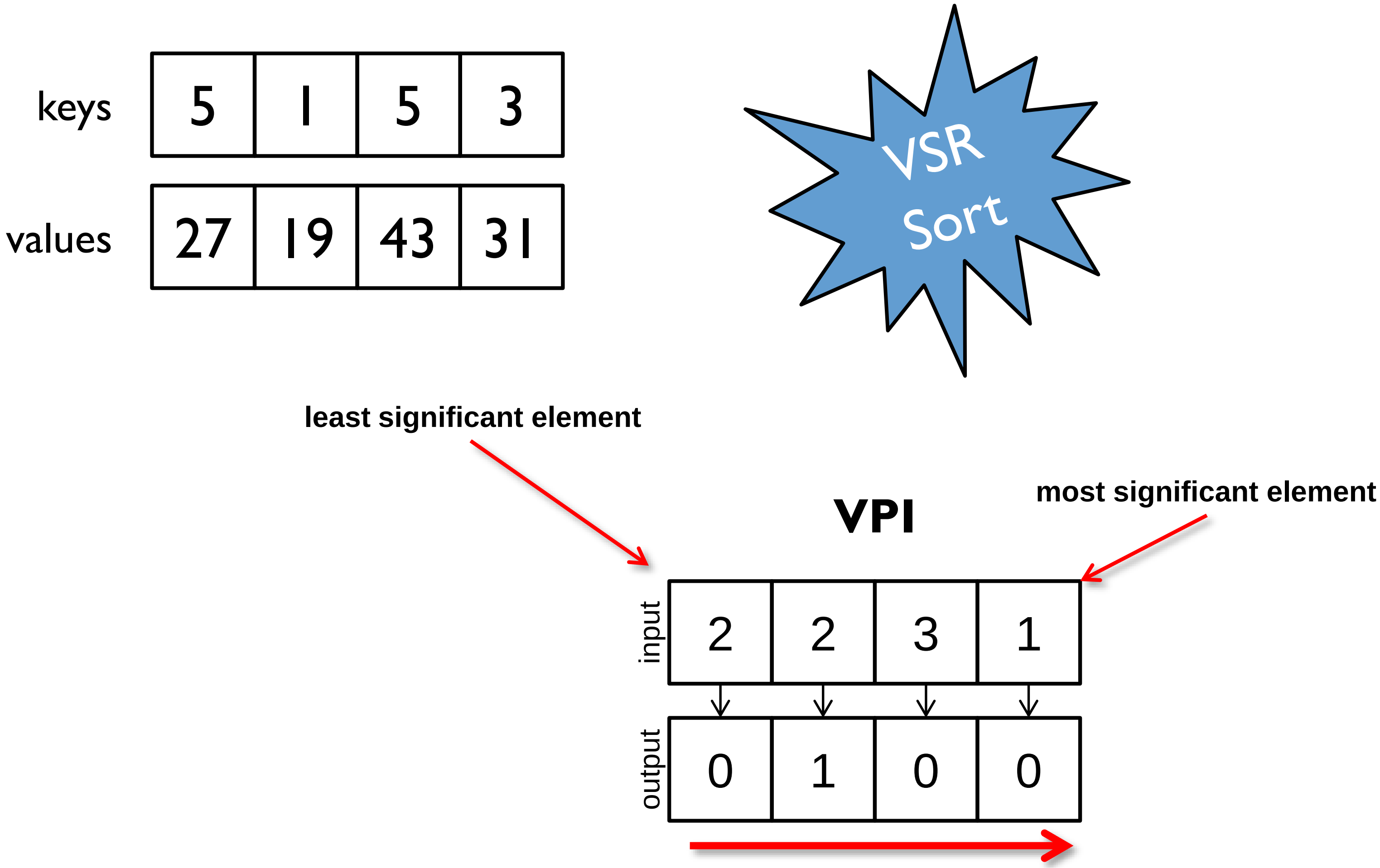
I. Motivation
II. What is Data Aggregation?
III. Experimental Setup
IV. Algorithms
 1. Scalar Baseline
 2. Polytable
 3. **Sorted Reduce**
 4. Monotable
 5. Partially Sorted Monotable
 6. Summary
V. Conclusions

Sorted Reduce

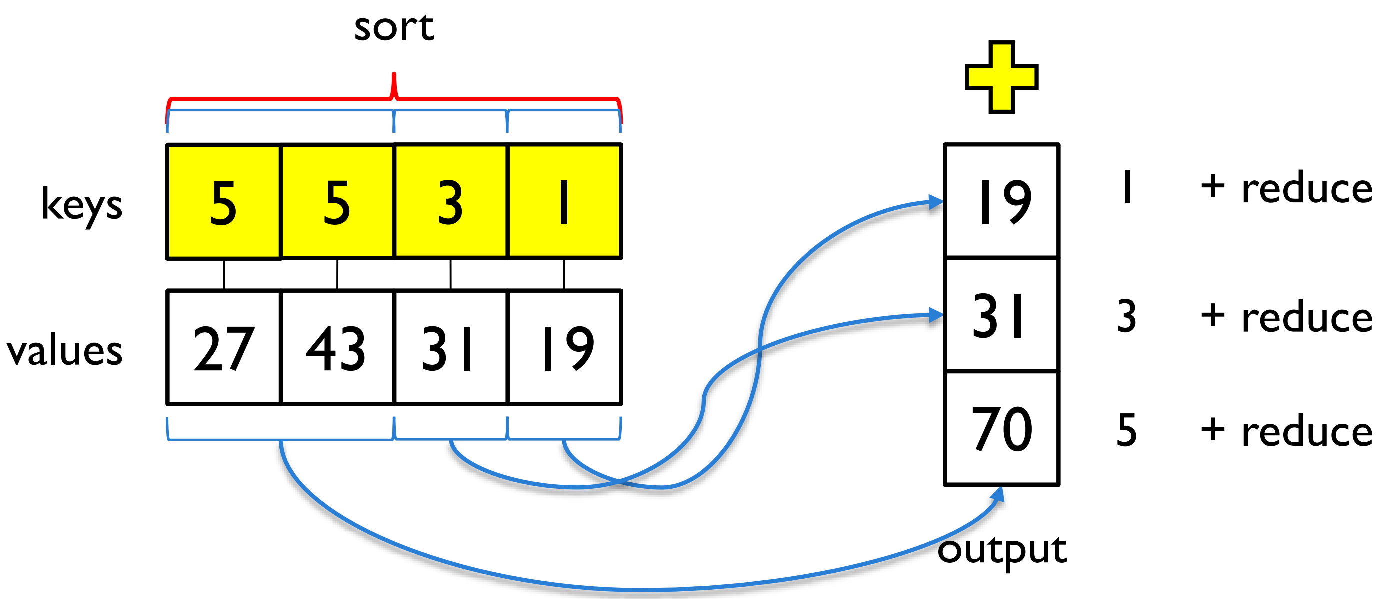
keys	5	1	5	3
values	27	19	43	31

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

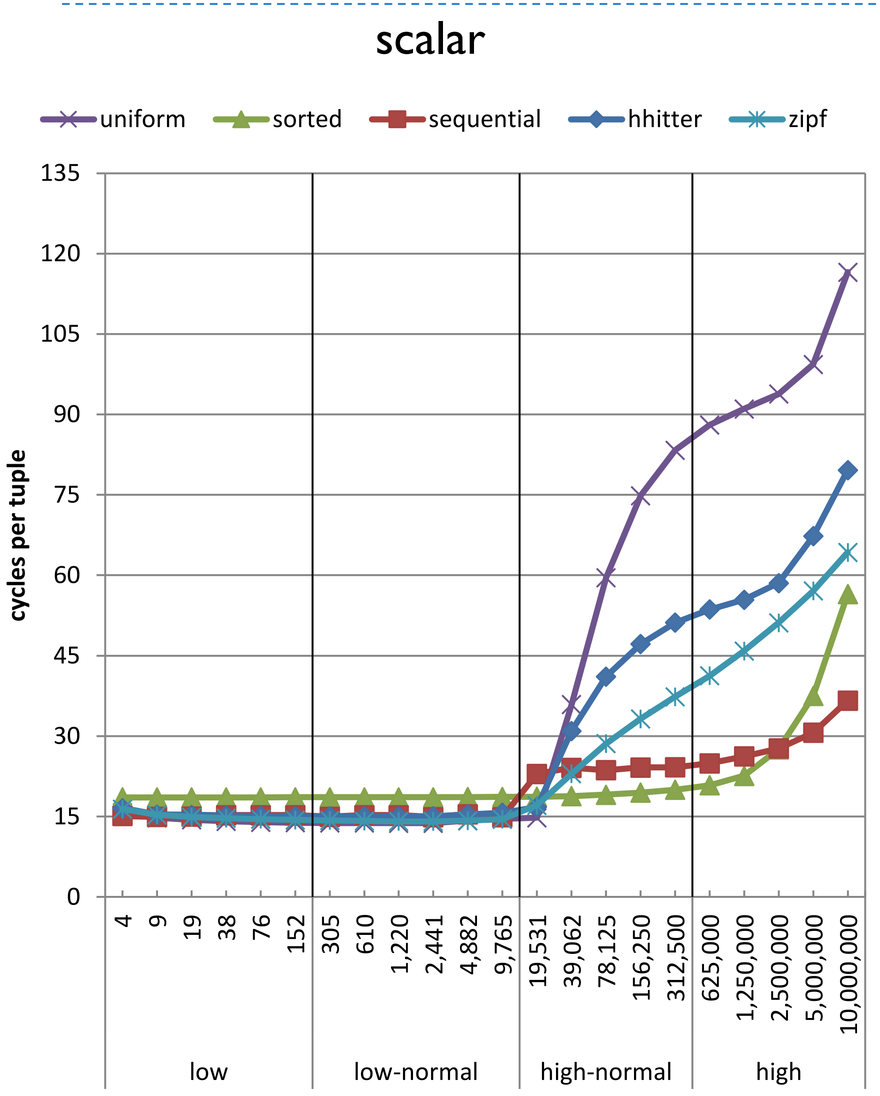
Sorted Reduce

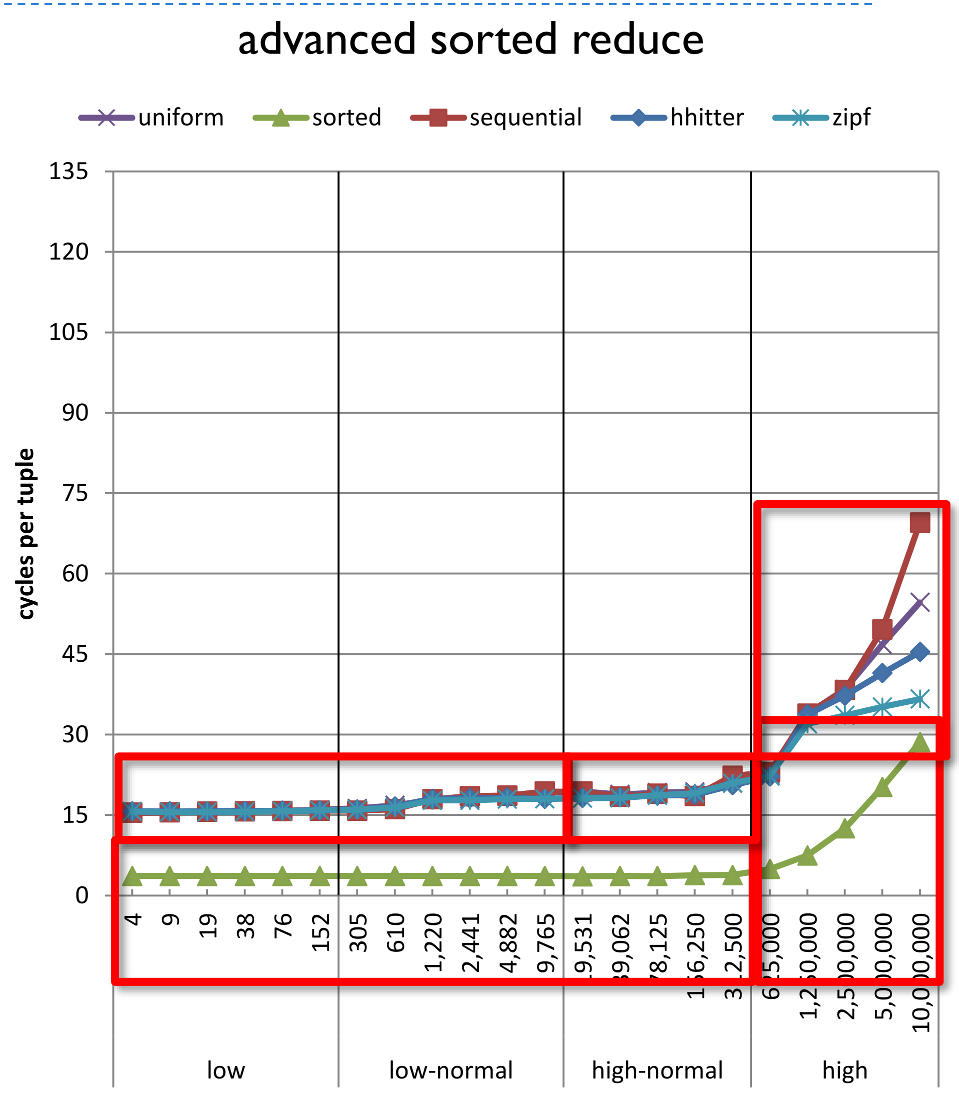


Sorted Reduce



Sorted Reduce – Results





Presentation Contents

I. Motivation

II. What is Data Aggregation?

III. Experimental Setup

IV. Algorithms

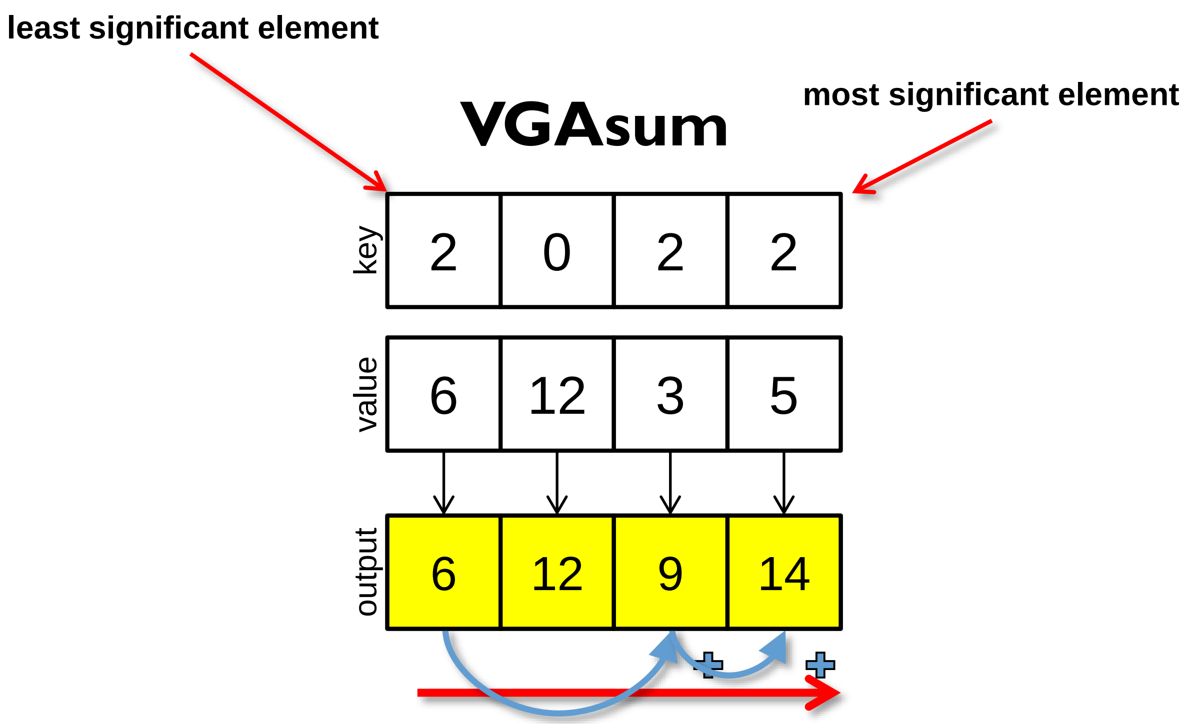
1. Scalar Baseline
2. Polytable
3. Sorted Reduce
4. **Monotable**
5. Partially Sorted Monotable
6. Summary

V. Conclusions

Monotable

- 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

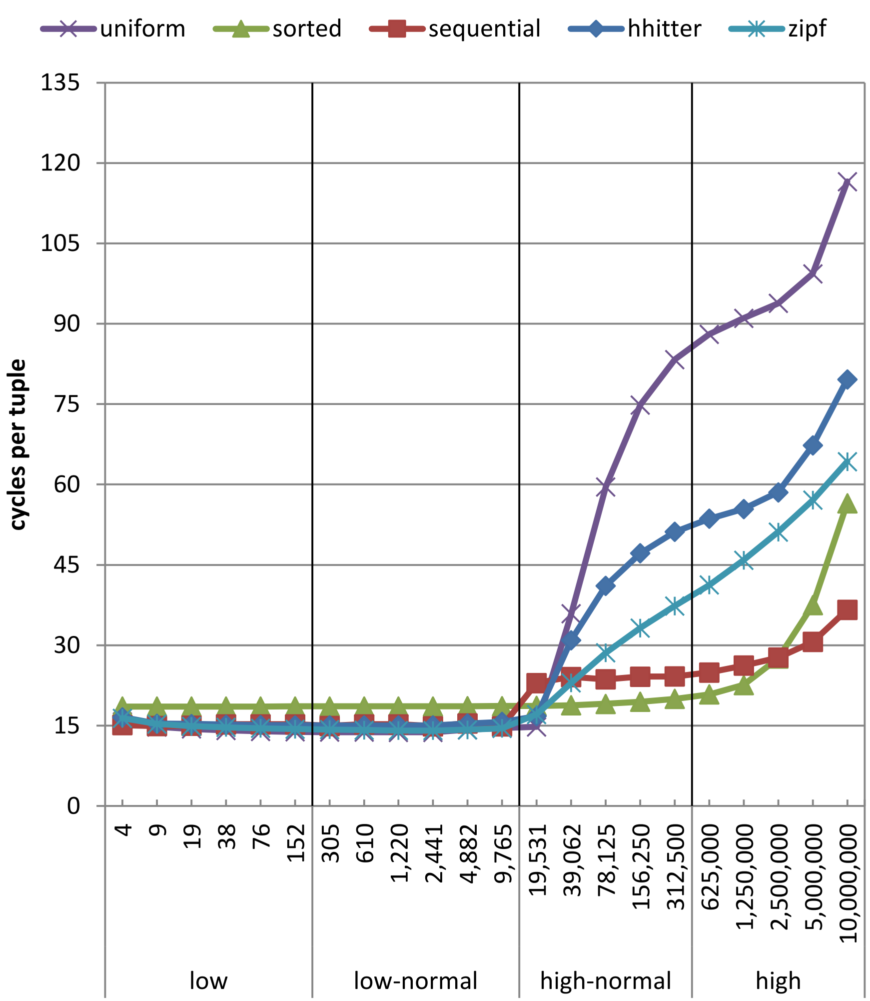
Monotable



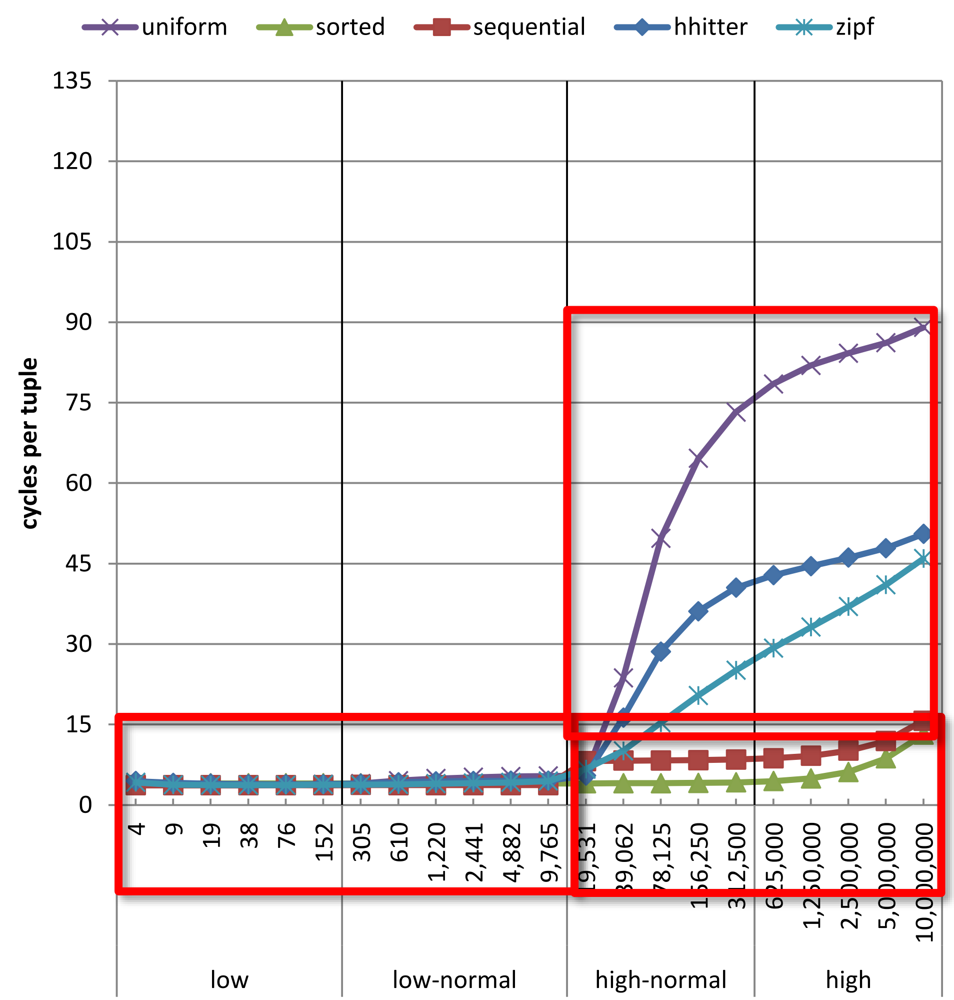
least significant element
VGAsum
most significant element
key
2
0
2
2
value
6
12
3
5
output
6
12
9
14

Monotable – Results

scalar



monotable

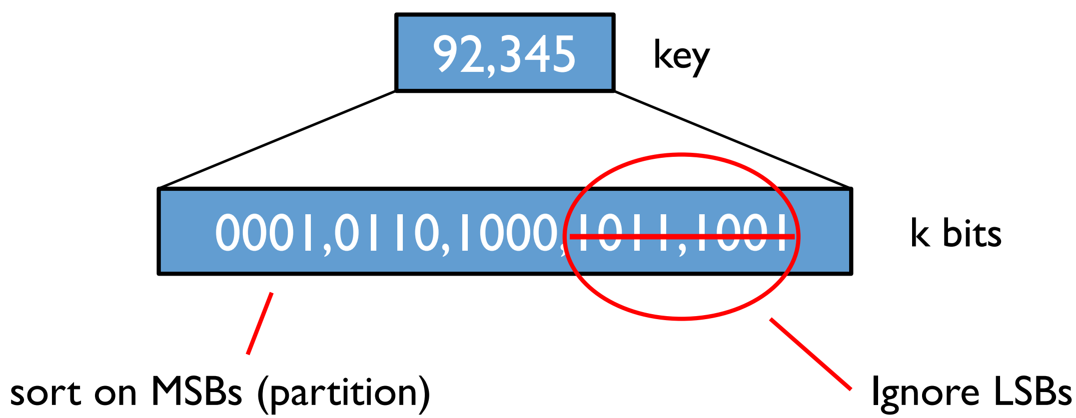


Presentation Contents

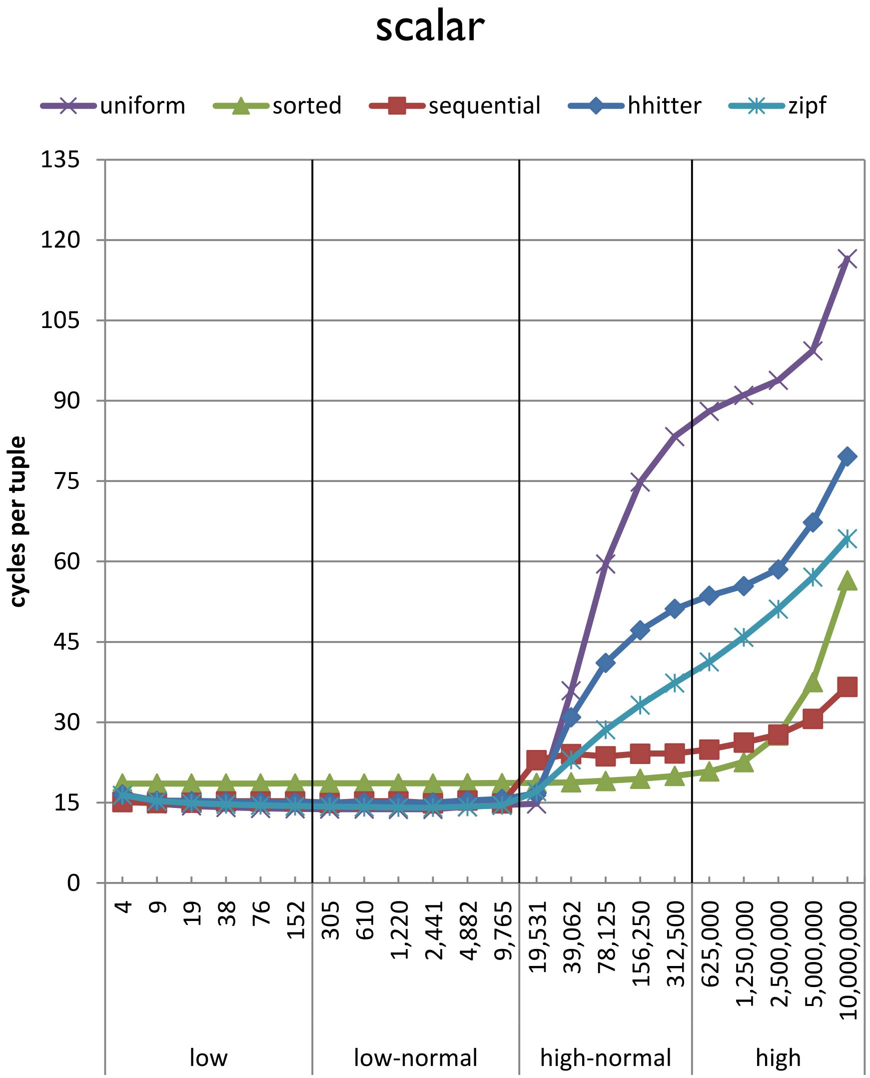
I. Motivation
II. What is Data Aggregation?
III. Experimental Setup
IV. Algorithms
   1. Scalar Baseline
   2. Polytable
   3. Sorted Reduce
   4. Monotable
   5. **Partially Sorted Monotable**
   6. Summary
V. Conclusions

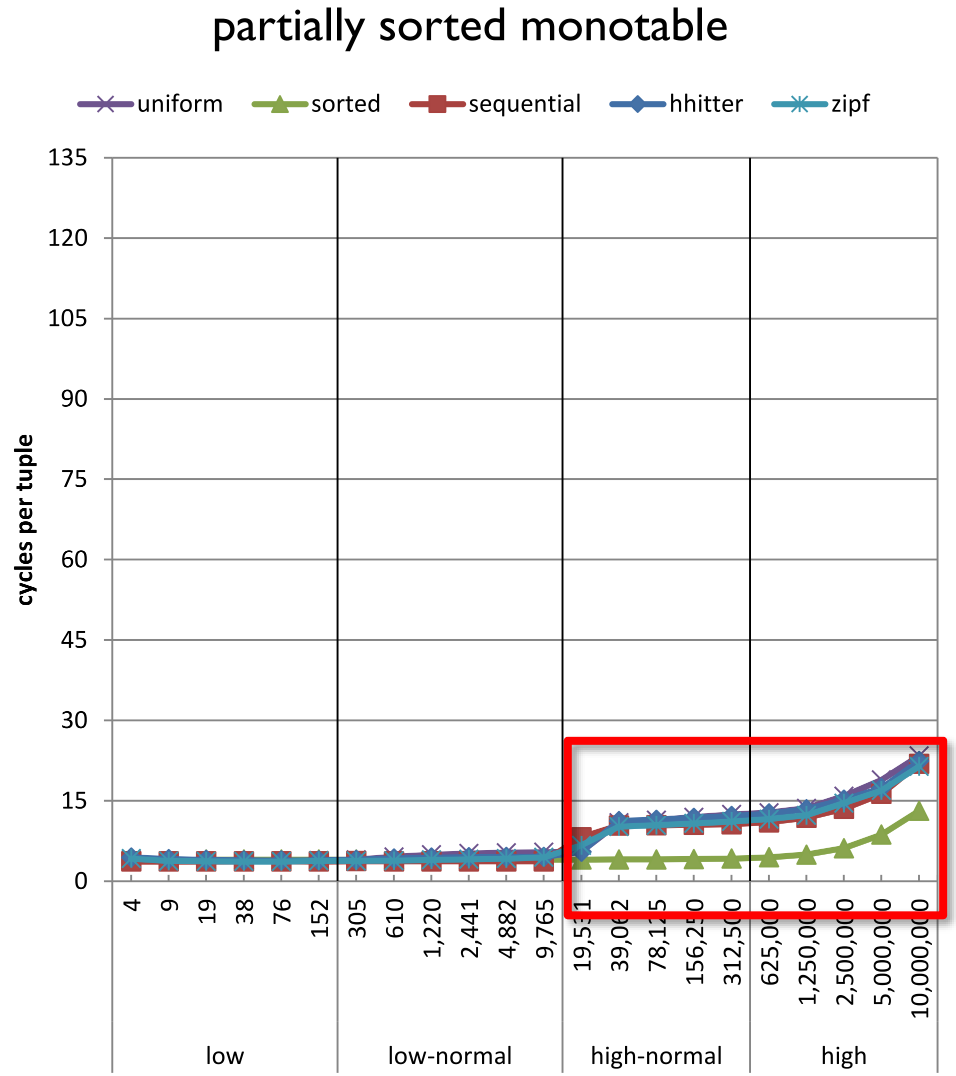
Partially Sorted Monotable

- 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



Partially Sorted Monotable – Results

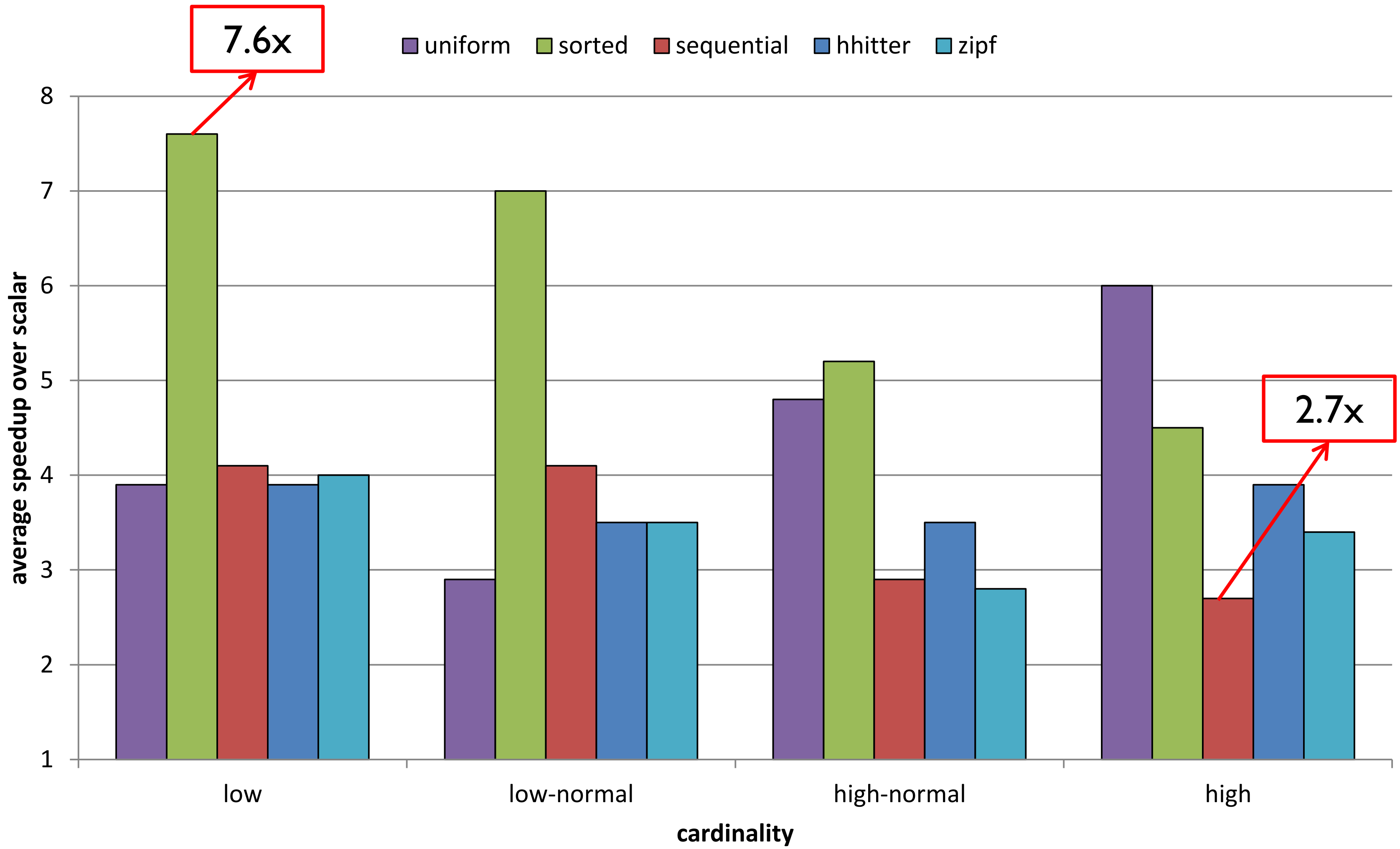




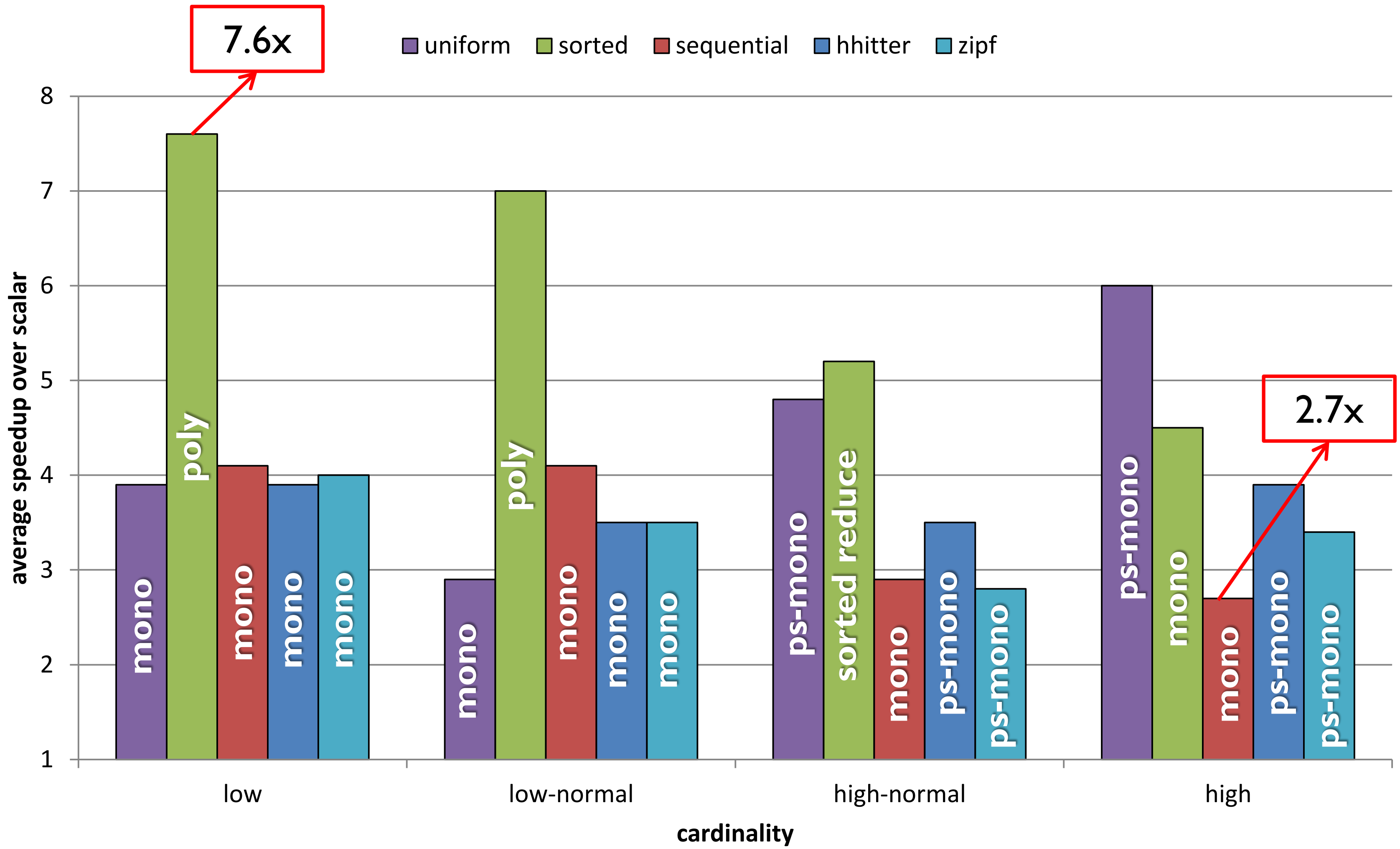
Presentation Contents

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

Summary – Best Speedups Overall



Summary – Best Speedups Overall



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