HIGH THROUGHPUT HEAVY HITTER AGGREGATION FOR SIMD PROCESSORS

Orestis Polychroniou

Kenneth A. Ross

Columbia University

AN OVERVIEW

- In a Glimpse
- Our Motivation
- Problem Definition
- Algorithmic Design
- Implementation & SIMD
- Experimental Results
- Closing Remarks

ADAPTED SCREENPLAY

Manager: *“I want a* plot *of our sales per product.”*

Employee: *“*All *products ?”*

Manager: *“Yes* all *products.”*

Employee: *“But* most *of our income comes from X, Y, Z products.”*

Manager: *“Well tell me about the* top *products then.”*

Employee: *“Ok* wait*…”*

…………..

Manager: *“Not ready* yet *?”*

Employee: *“Well the system does most* work *for* all *products.”*

Manager: *“Is this* necessary *?”*

Employee: *“Well maybe… Could it do* better *?”*

IN A GLIMPSE

- What do you do ?
  - Best effort aggregation for heavy hitters

- What is so special about it ?
  - We do it only for heavy hitters and it is fast

- Why do you do it ?
  - People see and use top results
  - Hardware is faster on smaller working sets

HUMAN MOTIVATION

- Analytics & Business Intelligence
  - Big data are available everywhere
  - Results used for human decisions

- Common Properties
  - Very large input handled by machine
  - Small output handled by humans

- Observation # 1

  *No matter how **big** the **data**, a **small** part of the **output** will be considered **by** the **human** factor (analysts, …).*

SOFTWARE MOTIVATION

- Common Analytics
  - Select – Project – Join: large intermediate results
  - Aggregate – Sort (Rank): use top results
- Aggregation Step
  - May produce few results / groups
  - If not, top results will be seen anyway
- Observation # 2

  *The DBMS will aggregate before returning any results. It will **work** for **1,000,000,000** groups, even if you **use 100** groups.*

HARDWARE MOTIVATION

- Caches are fast
  - Faster than RAM by 1-2 orders of magnitude
  - Can still fit thousands of groups
  - Private caches allow shared nothing parallelism
- Caches levels have variable speeds
  - L1 is 2-4 cycles, L3 is 25-40 cycles
  - Tradeoff between speed & capacity
- Observation # **3**

  *The smaller the **working** (result) **set**,*
  the faster the scan/probe phase.
  *But there are many **tradeoffs**.*

DATA MOTIVATION

- Skewed data are common
  - Zipf distribution is important
  - Skewed distributions for synthetic data
  - Strategic real data exhibit skew
  - Importance of items by rank (frequency)

- Sampling can estimate result
  - Top-K items will be in a sample
  - A verification step is required
  - Avoid going over the data multiple times

PROBLEM DEFINITION

- Heavy hitter groups
  - Aggregate top K groups by tuple cardinality
  - Defined by higher input frequency
  - Hopefully important groups for analysis

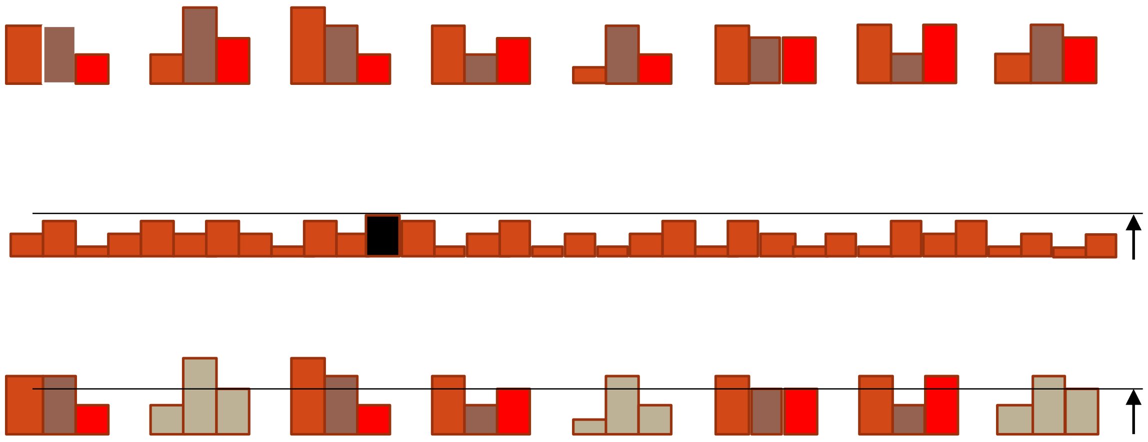
- Example query

```
select product_id, count(*)
from sales
group by product_id
order by count(*) desc
limit 1000;
```

OUR SOLUTION

- Identify possible heavy hitters
  - Sample input randomly
  - Extract heavy hitter candidates
  - Configure & build a hash table
- Scan over input data & probe
  - Update candidates found in the table
  - Increment non-candidate counts
- Verify heavy hitter groups
  - Max non-candidate is threshold
  - Like an 1D count sketch

VERIFICATION VISUALIZED



- Candidate Aggregates
- Non-Candidate Counts
- Verified Aggregates

■ counts ■ sum(X) ■ max(Y)

TRADEOFF ASPECTS

- Candidate aggregates
  - Store the whole incomplete aggregate
  - If smaller then higher in cache & faster
  - If larger more candidates & more accurate
- Non-candidate counts
  - Store only a count
  - Less counts make it faster
  - More counts more accurate
- Goal
  - Choose fastest configuration
  - Accurate enough to verify top K

WHERE AND HOW TO USE

- Conventional Aggregation
  - Small group-by cardinality

- Optimization Step
  - Loop over configurations
  - Estimate configurations using sample
  - Choose best configuration
  - Early failure detection

- Verified < K
  - On failure roll back to conventional
  - Fast enough to retry other configuration

FAILURE CASES

- Correct top K are not among the candidates
  - Sample size was small and inaccurate
  - Cannot distinguish top groups by sampling
- Cannot verify K candidates
  - Not enough non-candidate counts
  - High verification threshold
- Wrong table configuration
  - Not enough candidate aggregates
  - Not enough non-candidate counts

HASH TABLE

- Multiplicative hashing
  - Fast computation
  - Random multiplier
- Perfect hashing
  - No branching and branch mispredictions
  - Fast reply for " is key X in the table ? "
  - Birthday paradox explains small load factor
- Bucketized hashing
  - Load factor of perfect hashing increases
  - Fast branch free probe through SIMD

HASH TABLE CONFIGURATIONS

- Cuckoo hashing
  - Two choice probe without branching
  - 2X perfect hashing with larger load factor
  - Combine with bucketized hashing
- Hash configurations
  - Cuckoo or perfect ?
  - Bucket size ?
  - Cache level ?
  - # of non-candidate counts ?
  - More choices more tradeoffs

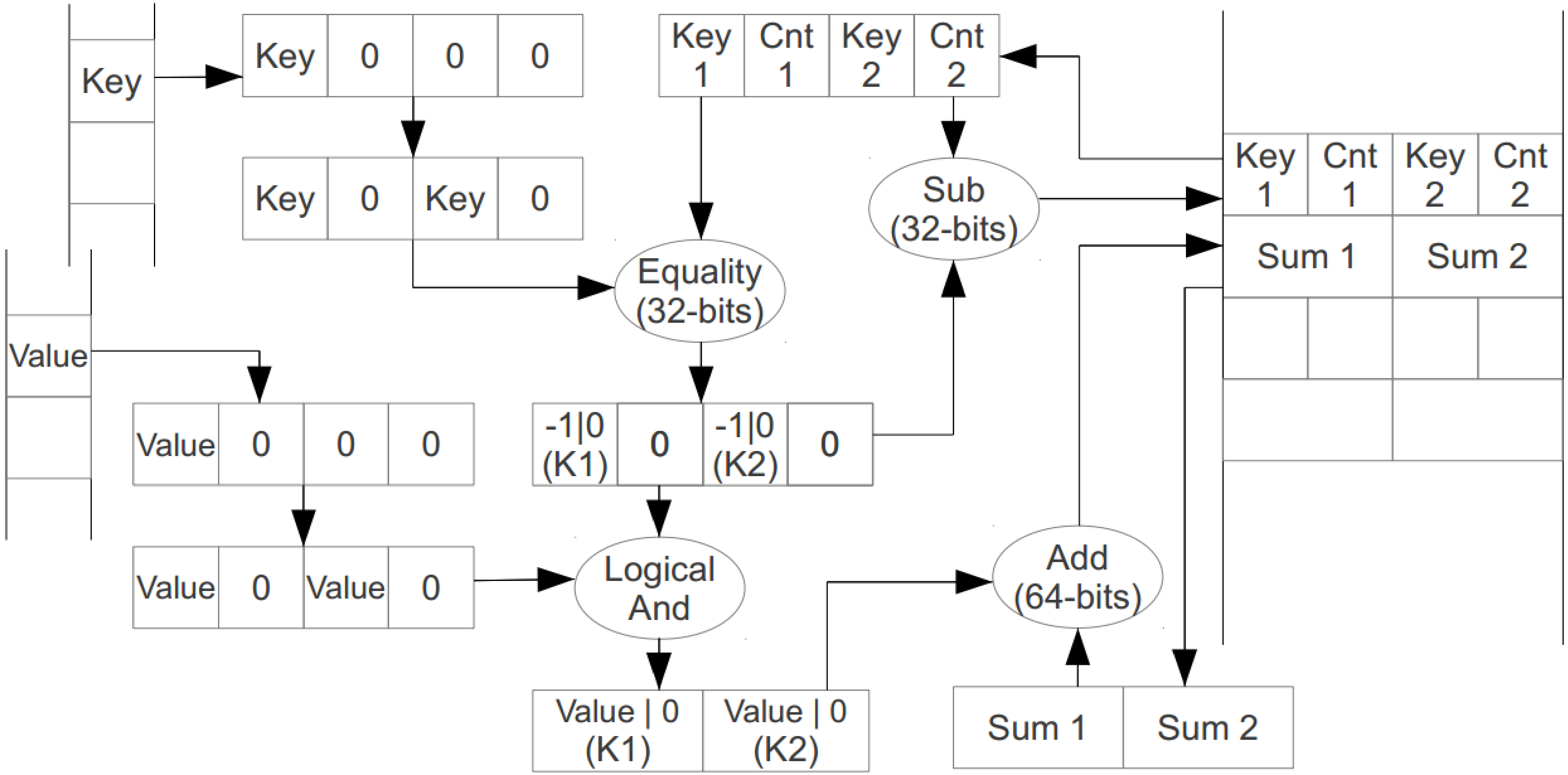
HASH TABLE UPDATES

- Branch free update
  - Updates nullified if keys do not match
  - Non candidate counts updated offline
- Why SIMD ?
  - Scalar code uses slower control flags
  - Transform to data dependencies
- SIMD where ?
  - To batch compare keys
  - To update & nullify faster

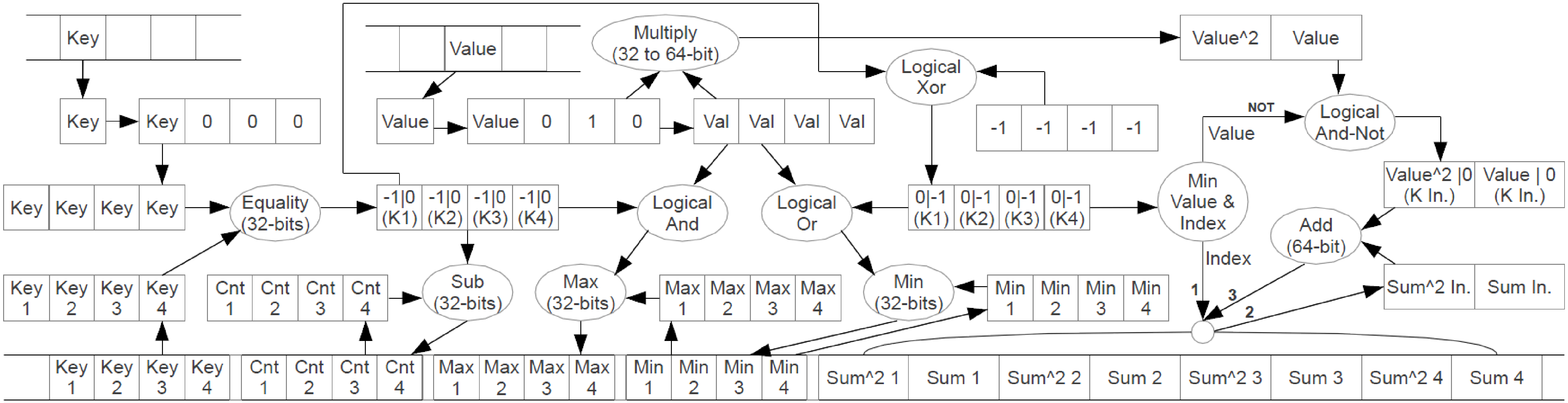
SCREENSHOT OF A CONFIGURATION

- Perfect hashing
- 2-wide bucket

select count(*), sum(value)
from table
group by key
order by count(*) desc
limit …



A MORE COMPLICATED QUERY

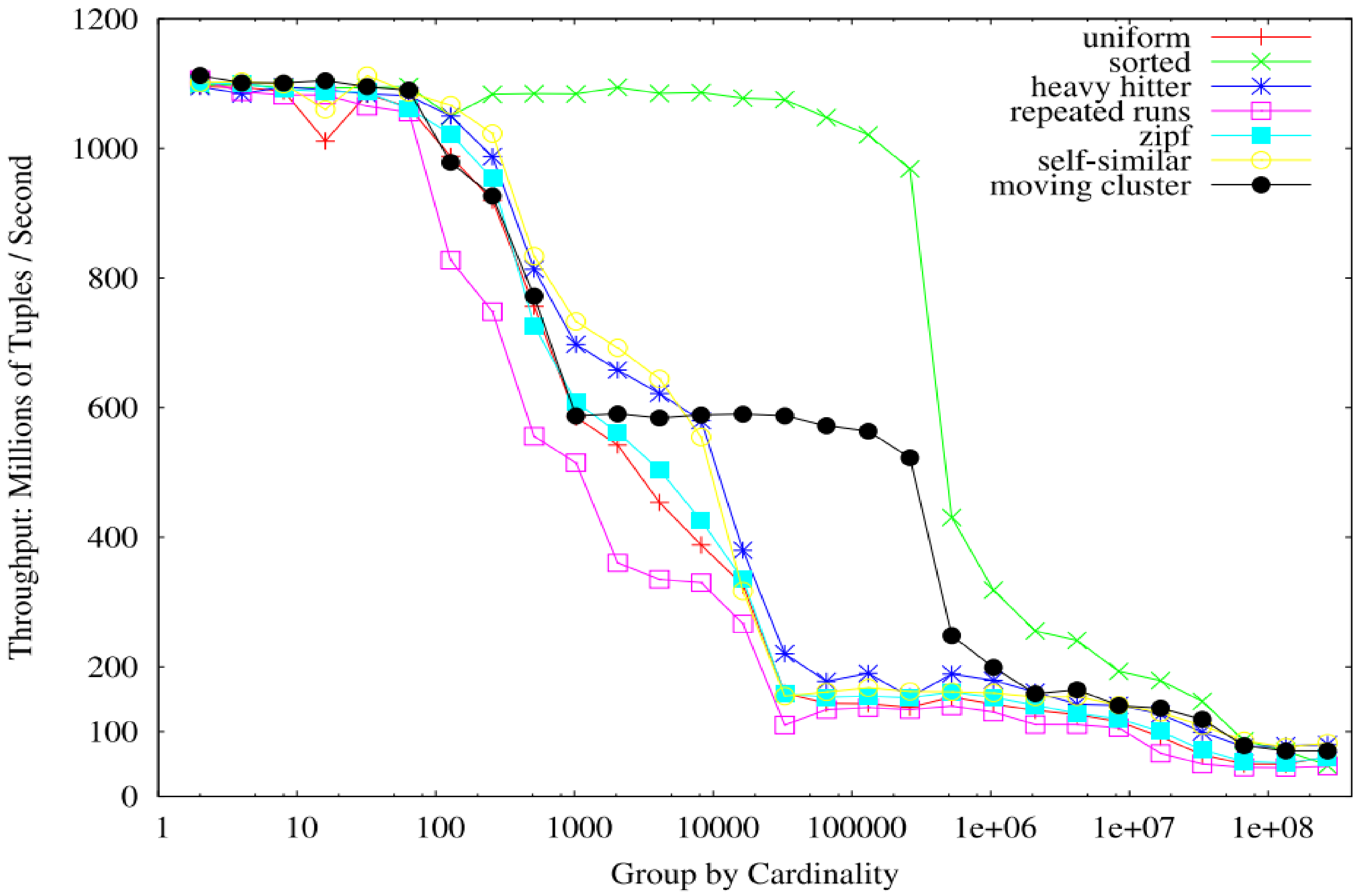


- Perfect hashing
- 4-wide bucket

select count(*), sum(X),
*max(X), min(X), sum(X*X)*
from table group by key
order by count(*) desc …

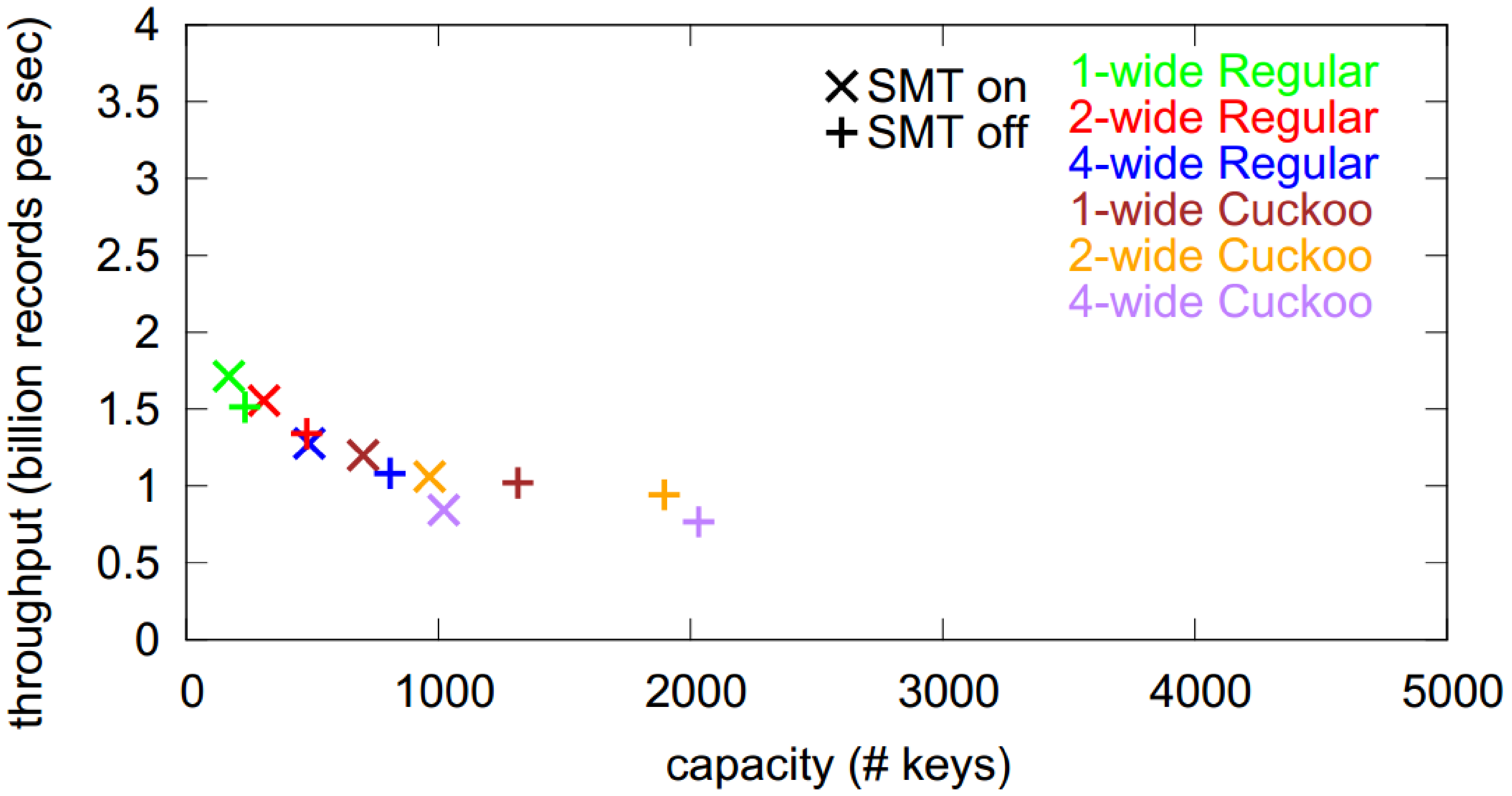
CONVENTIONAL AGGREGATION

- Single pass
  - Large hash table for aggregates with random hits on RAM
  - PLAT method used for cross-core cache invalidations due to heavy hitters [Ye, DaMoN11]
- Multiple passes
  - Bound by RAM throughput
  - Hash tables on cache



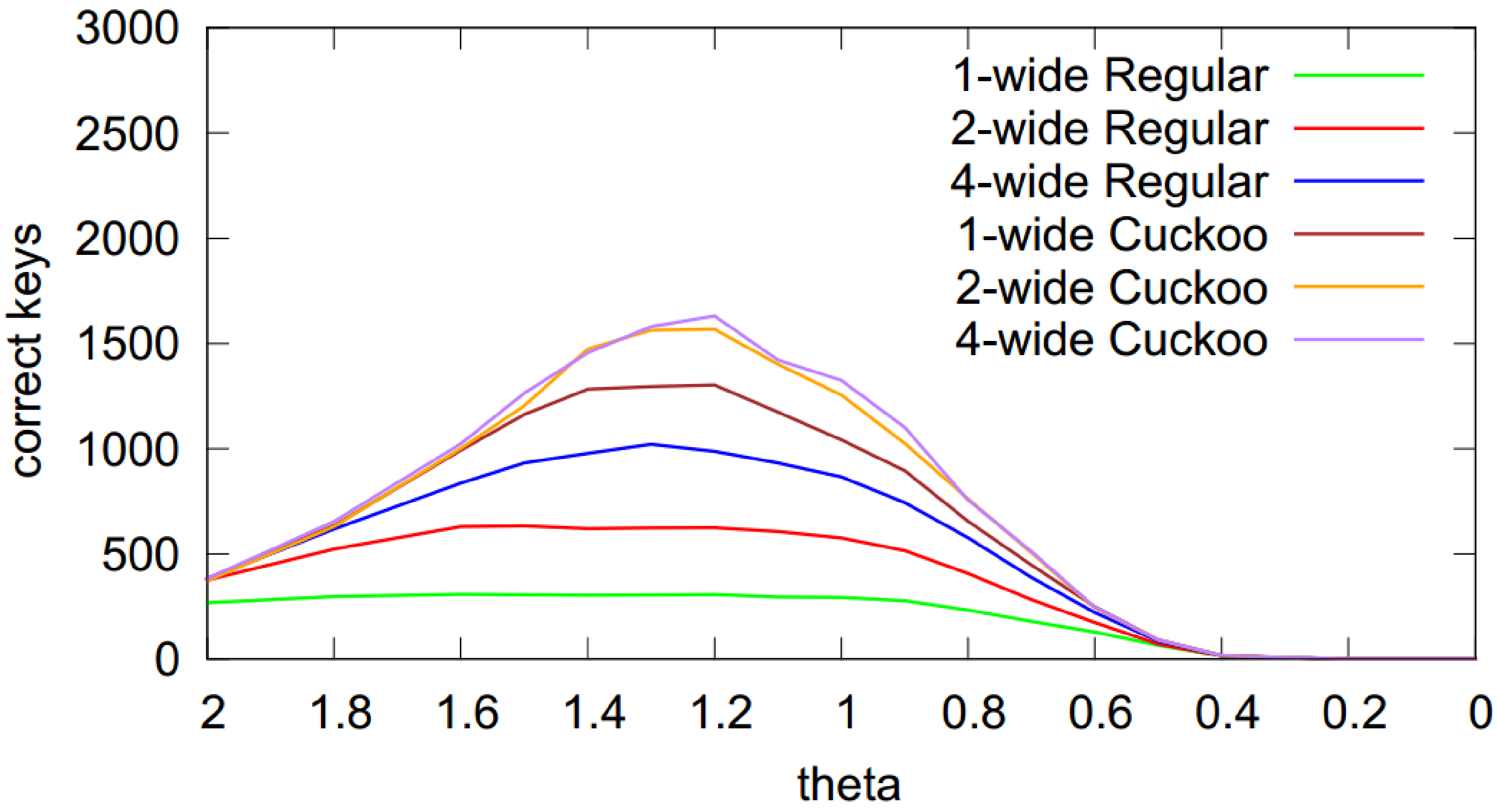
PERFORMANCE TRADEOFF

- 2 CPUs @ 2.4 GHz
- Intel Nehalem Appeared 2008
- 4 physical cores / CPU
- 2 hardware threads / core

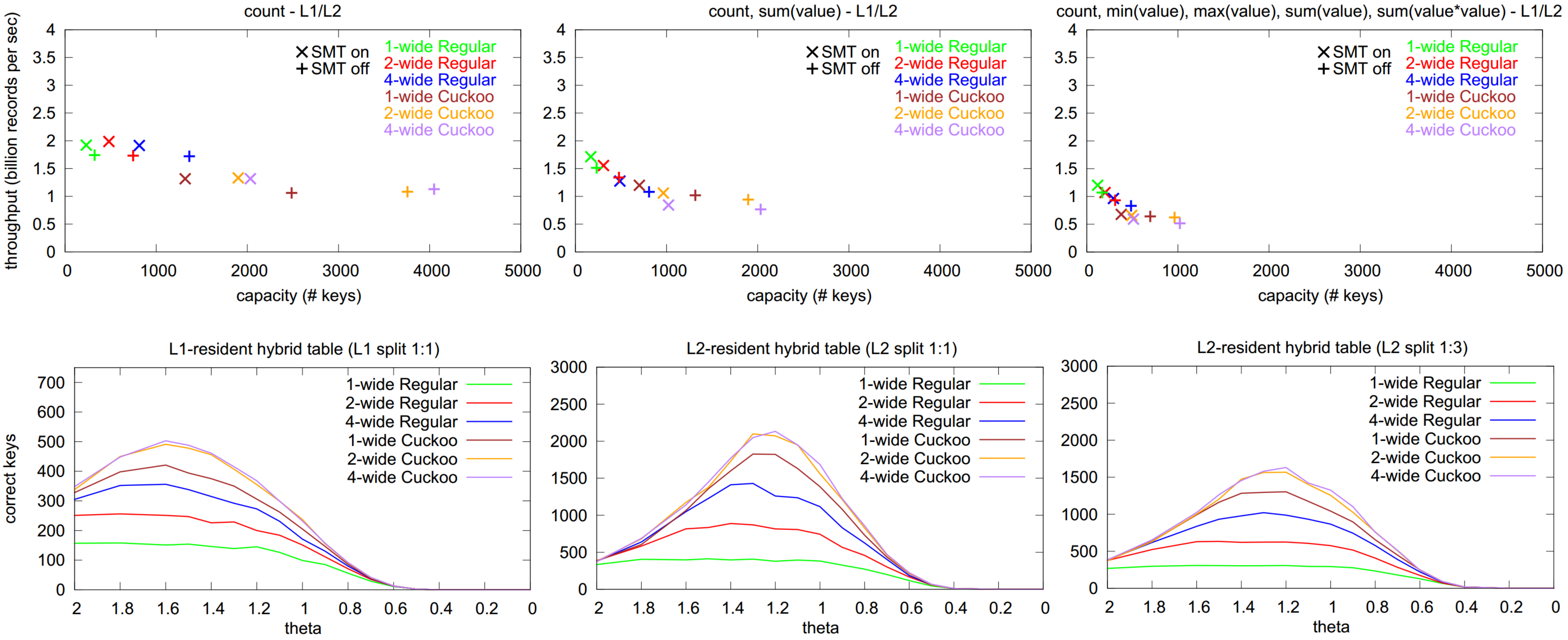


QUALITY TRADEOFF

- 32 KB L1 cache private / core
- 256 KB L2 cache private / core
- Version SIMD SSE 4.2



COMBINED RESULTS



REALISTIC EXPERIMENT

- Wikipedia
  - Hourly Wikipedia visits for January 2012
  - Group by URL & get average visit hour
- Skew
  - 3,463,321,585 visits
  - 102,216,378 distinct URLs
  - Top-3 URLs are 1.6 % of total
  - Top-100 URLs are 6.65 % of total
  - Top-10,000 URLs are 25.3 % of total

WIKIPEDIA DATASET

Candidates	Non-Cand.	Scheme	1-wide			2-wide			4-wide		
			Time	HH.	Freq.	Time	HH.	Freq.	Time	HH.	Freq.
L1 $\times 1/2$	L1 $\times 1/2$	Regular	2.32	9	3.62	3.07	10	3.62	3.47	10	3.62
		Cuckoo	3.41	12	3.62	3.93	12	3.39	4.78	12	3.45
L1 $\times 1/4$	L1 $\times 3/4$	Regular	2.15	14	3.39	2.55	14	2.95	3.28	16	2.72
		Cuckoo	3.47	15	2.78	3.73	16	2.78	4.59	16	2.70
L2 $\times 1/2$	L2 $\times 1/2$	Regular	3.59	92	1.00	3.67	145	0.75	4.11	187	0.69
		Cuckoo	4.49	217	0.63	4.67	260	0.61	5.72	273	0.57
L2 $\times 1/4$	L2 $\times 3/4$	Regular	2.77	103	0.95	2.98	146	0.78	3.68	187	0.67
		Cuckoo	3.92	215	0.62	4.28	260	0.59	5.38	268	0.57
L1	L2 $\times 3/4$	Regular	2.59	84	0.89	2.83	121	0.88	3.55	141	0.80
		Cuckoo	3.74	162	0.73	4.11	179	0.72	5.24	179	0.71

- # **verified** top groups
- min (K^{th} item) frequency (x 10^{-4})
- execution **time** (seconds)

select count(*) as visits,
avg(hour) as mean_visit_hour
from wikipedia
group by URL
order by count(*) desc;

FINAL REMARKS

- Usefulness
  - Applied on specific queries
  - Requires skew in data
  - Best effort approach
  - Useful for data exploration

- Quality
  - 5-20x faster than conventional aggregation
  - Get top 250 results out of > 25 GB in time < 5 sec
  - Smallest forms 0.006 % of total

ANY QUESTIONS