Adaptive Aggregation on Chip Multiprocessors

John Cieslewicz - Kenneth A. Ross
Columbia University

Motivation

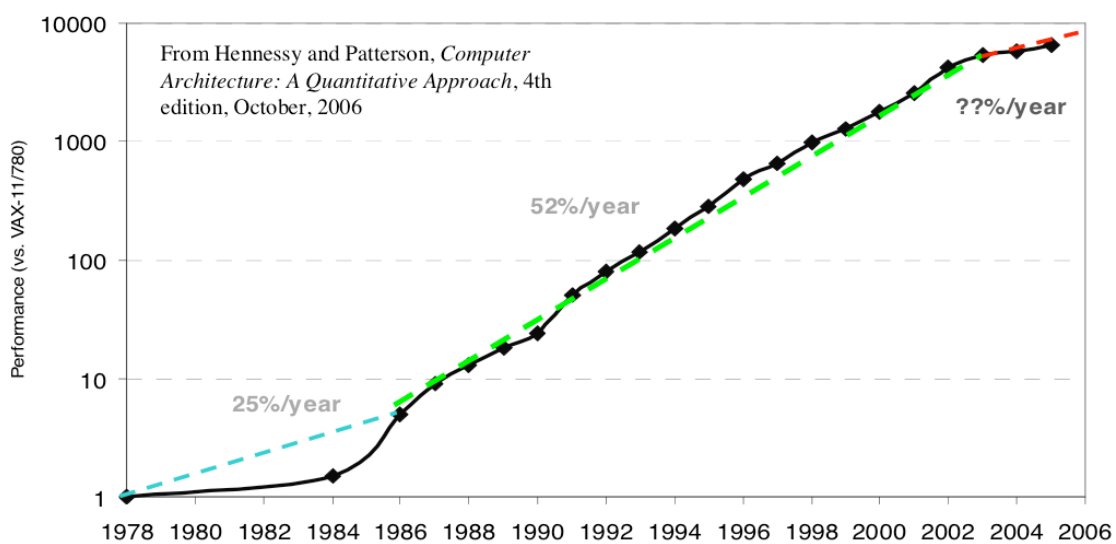
- Aggregation is a well understood database operation
- Seemingly easy to implement
- Chip multiprocessors introduce new challenges and opportunities
- Picking the wrong aggregation technique can result in a performance penalty of more than an order of magnitude

Outline

- Chip Multiprocessors
- Aggregation Strategies
- Modeling Performance
- Adaptive Aggregation
- Experimental Results

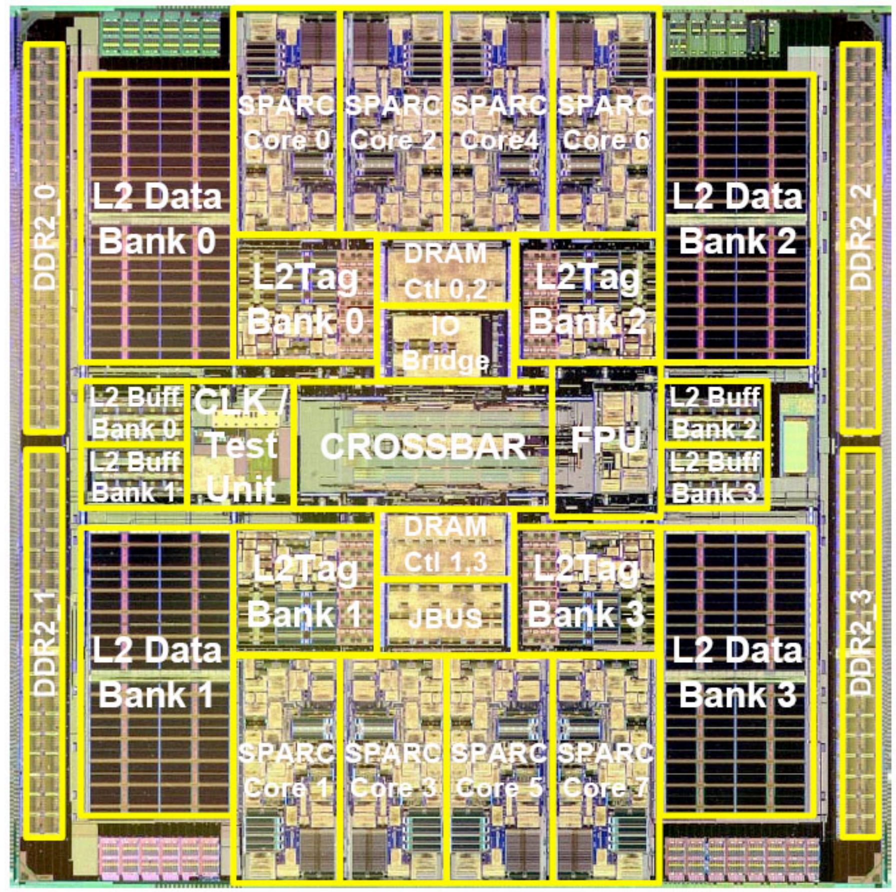
The Multicore Future

- ILP is tapped out
- Heat dissipation and power consumption problems
- Thread Level Parallelism (TLP) is the future of performance gains



Sun UltraSPARC T1

- 1 GHz
- 8 cores
- 4 threads / core
- 8 KB L1 D$ / core
- 16 KB L1 I$ / core
- 3 MB Shared L2
- Simple cores



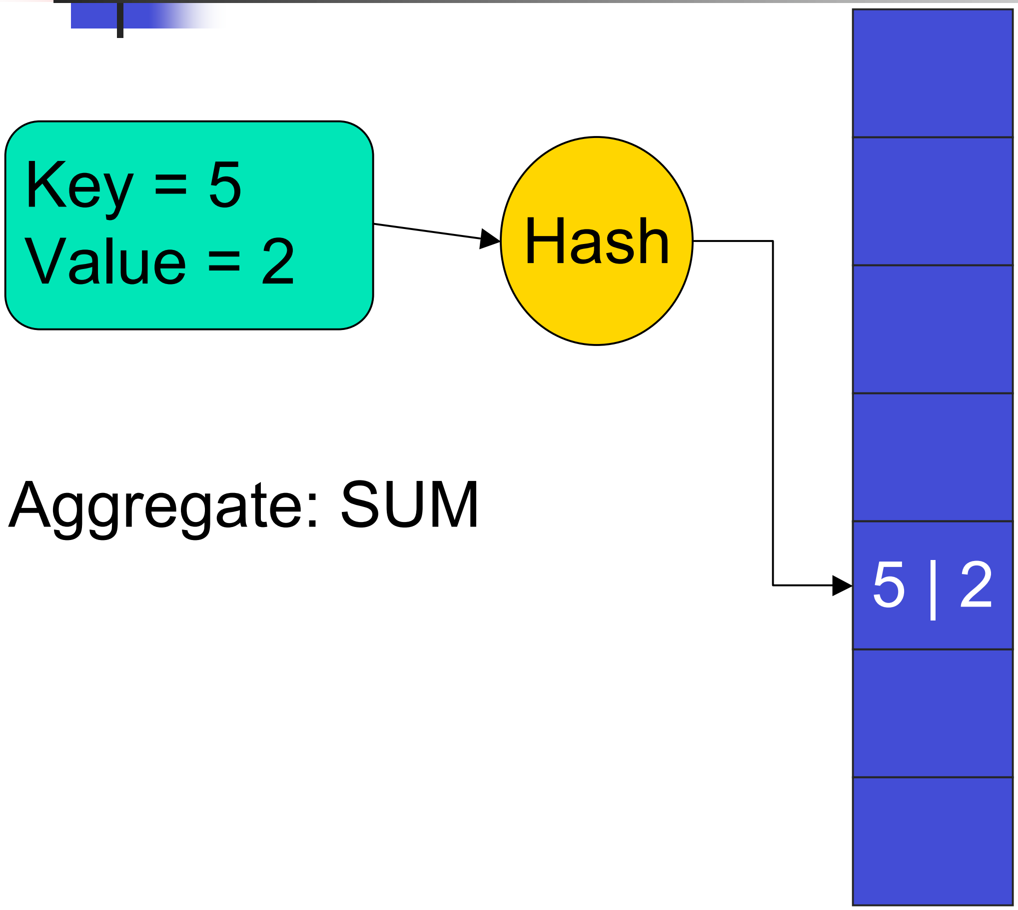
Aggregation Overview

- Group tuples by zero or more attributes
- Compute an aggregate for each group
  - SQL standards: COUNT, SUM, AVERAGE, MIN, MAX
- Two common strategies:
  - Sorting
  - Hashing

Hashing, not sorting

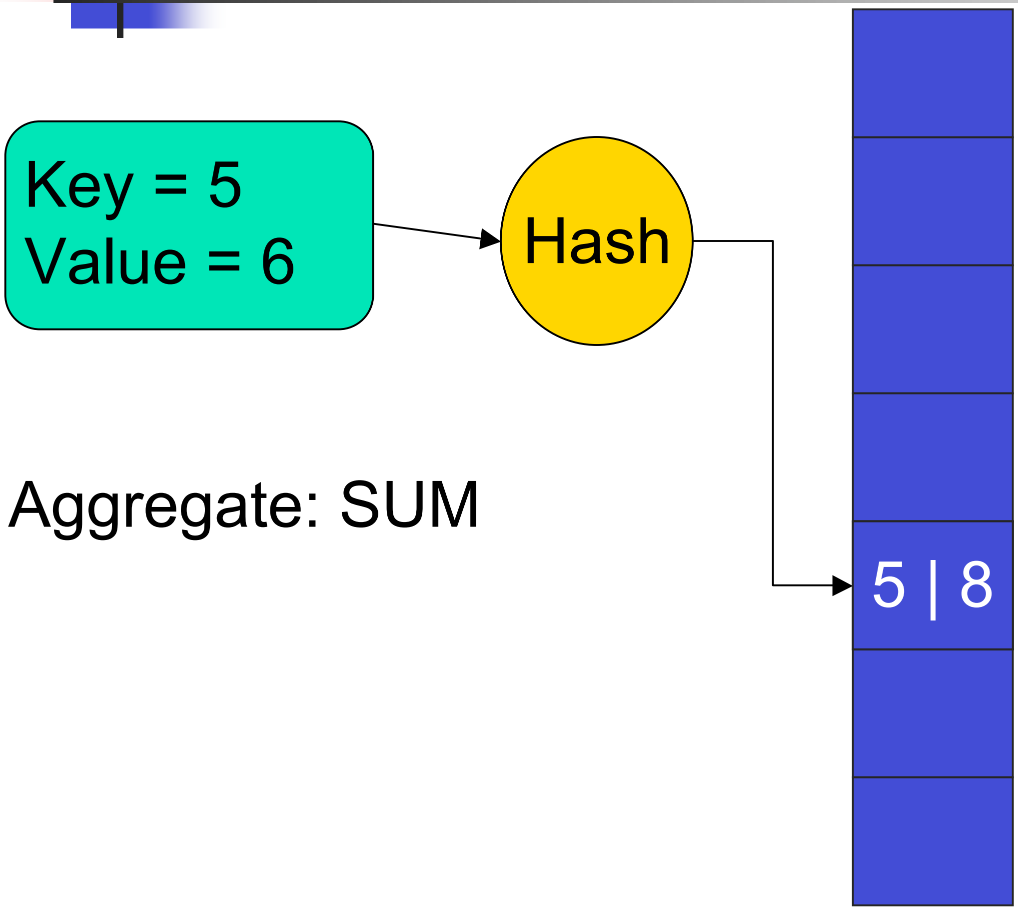
- Sort based aggregation is a blocking operation
  - All input must be materialized
- Hashing allows for better pipelining and arbitrary partitioning of the input
- We focus only on hashing for the rest of the presentation

Hash Aggregation



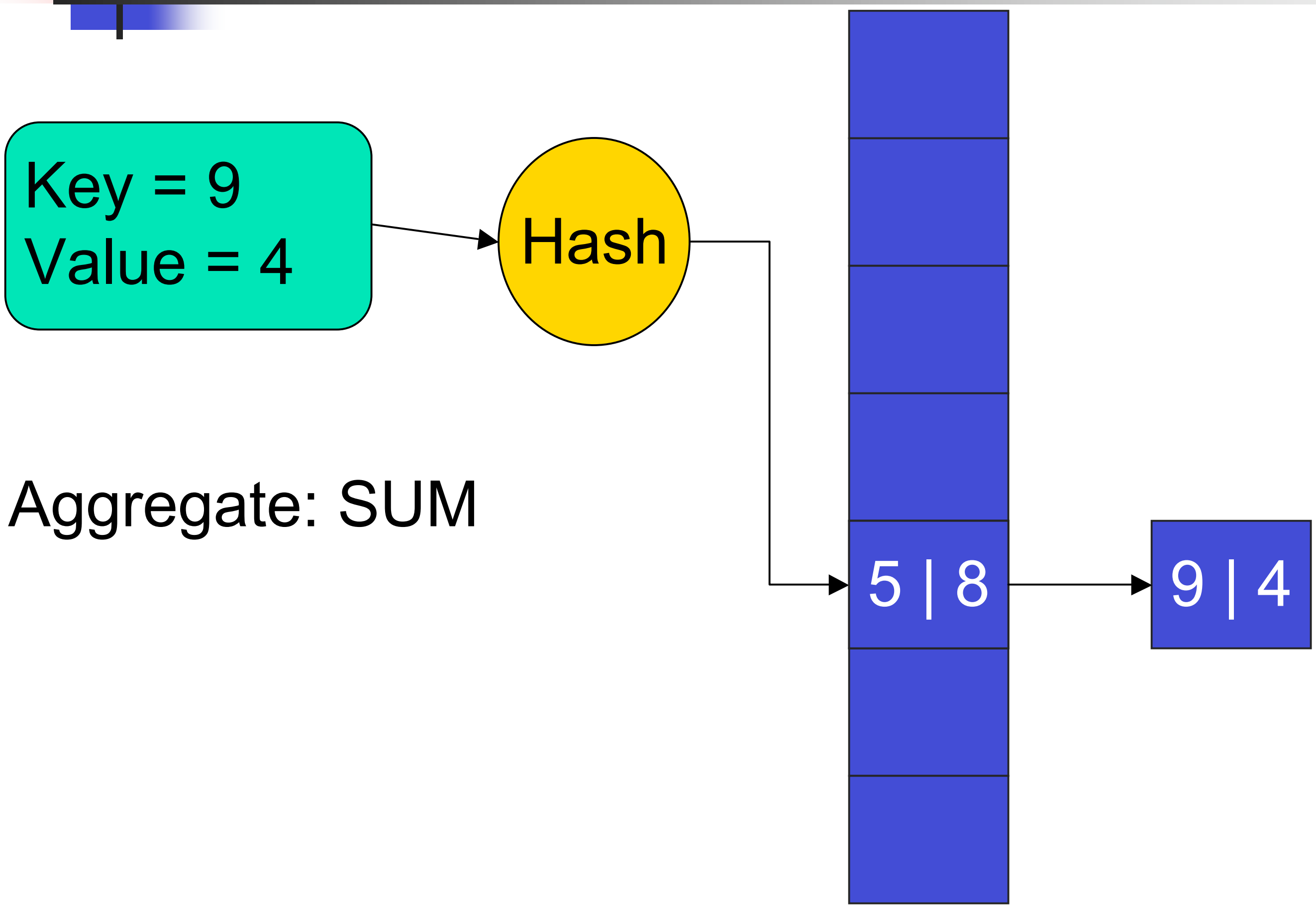
Aggregate: SUM

Hash Aggregation



Aggregate: SUM

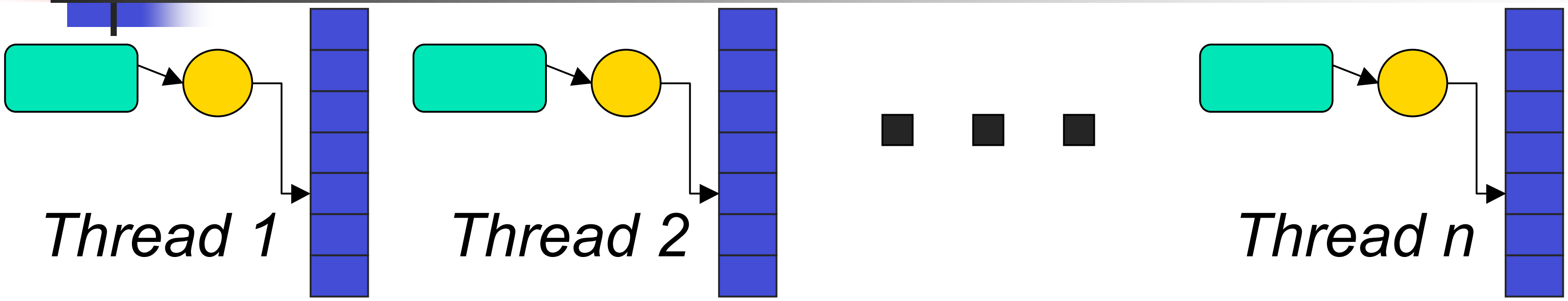
Hash Aggregation



Aggregation Implementation

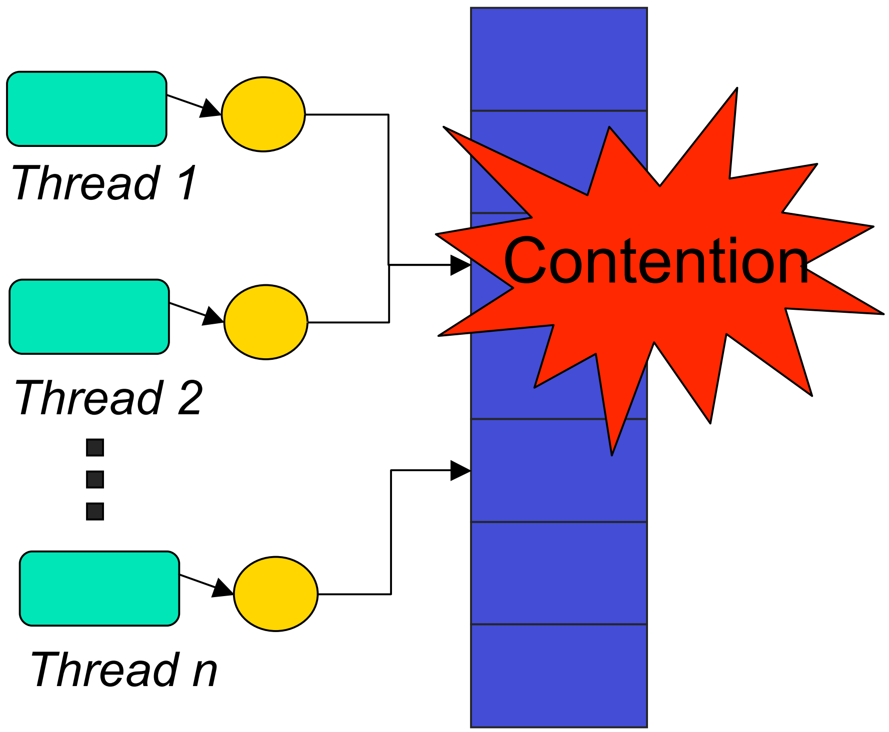
- All threads share input stream
  - Read contiguous chunks
- Execute same operation
- Intra-operator sharing and conflicts are easier to reason about than inter-operator
- Instructions shared by threads
  - Instruction cache misses are expensive

Option 1: Independent Tables



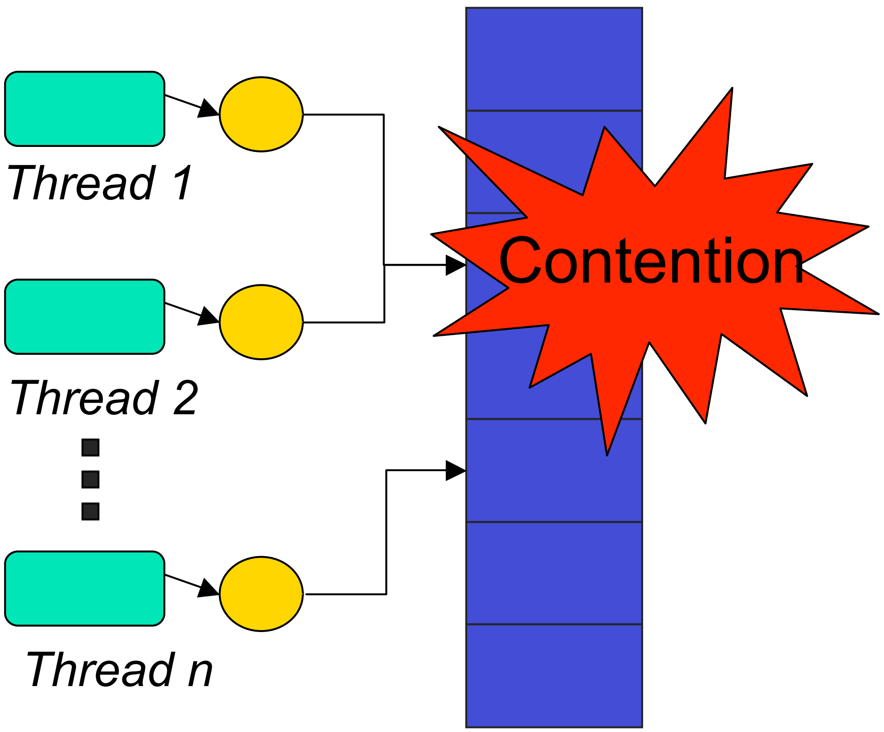
- Each thread has its own hash table
- Advantages:
  - No coordination between threads
- Disadvantages:
  - No sharing
  - *Capacity* and *conflict* cache misses
  - Huge memory requirement

Option 2: Global Tables with Mutexes



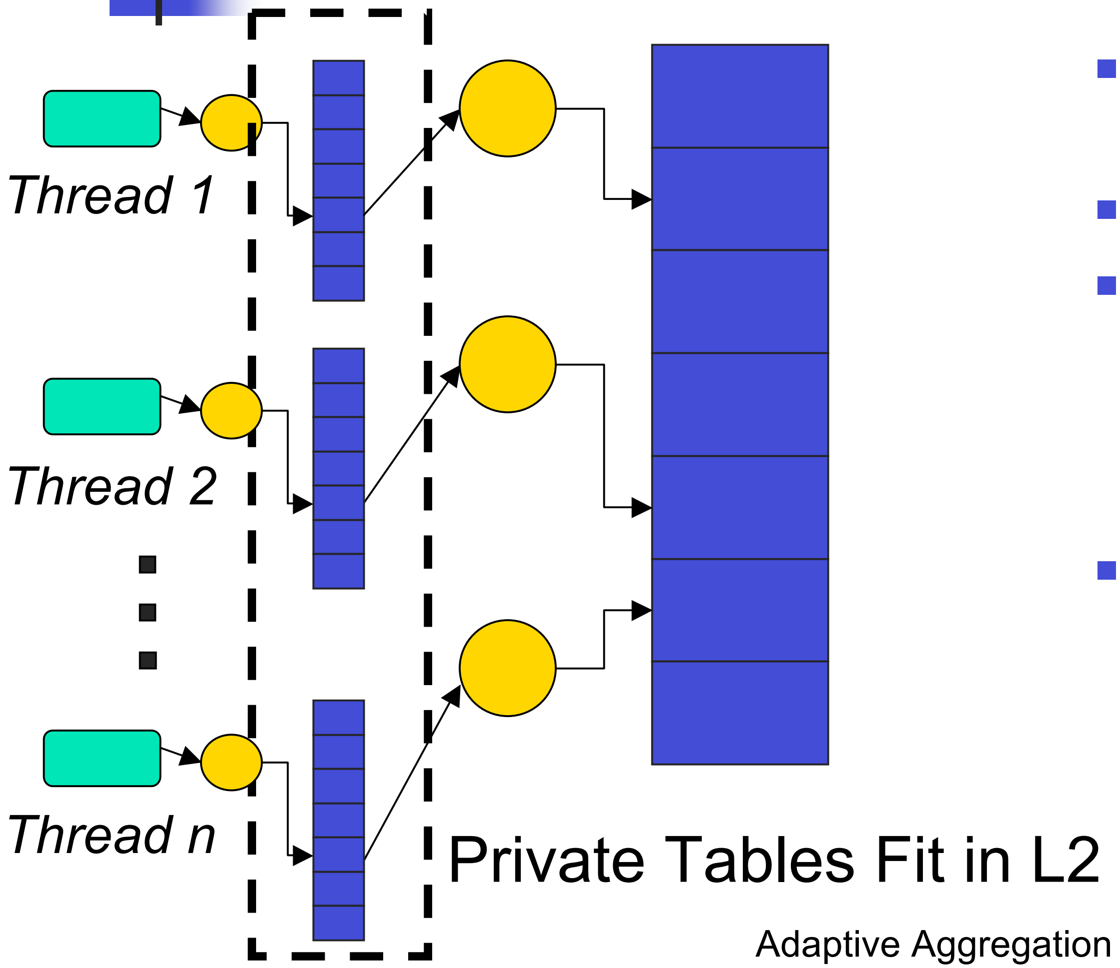
- Threads share 1 table
- Advantage:
  - Shared table means more unique aggregate values fit in the cache
- Disadvantage:
  - Hash buckets must be locked to prevent race conditions
  - Contention for common keys

Option 2: Global Tables with Atomic Instructions



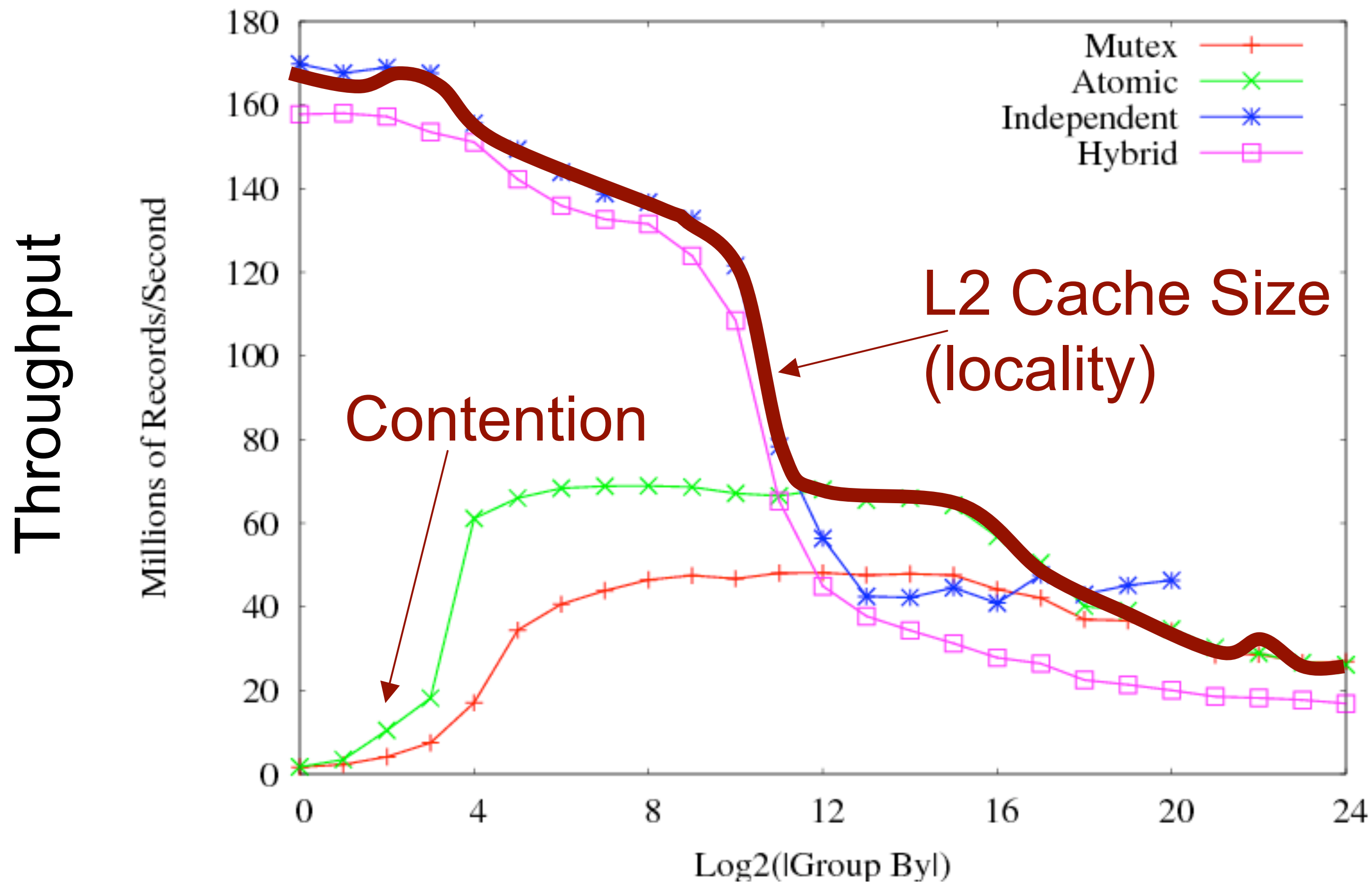
- Atomicity like a mutex, but...
- No locking, use *atomic operations* for updates
- Provided by many microarchitectures
- More efficient than locking, longer latency than comparable non-atomic operation

Option 3: Hybrid



- Independent, fixed size tables fit in L2 cache
- "Spill" to global table
- Advantage:
  - locality
  - contention free
  - lower memory needs
- Disadvantage
  - Pure overhead if global table is better

Aggregation Performance

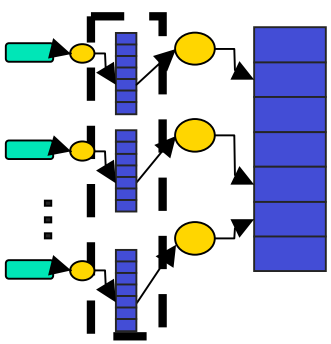


What influences performance?

- Runs
  - Consecutive tuples with same key can be aggregated directly
- Locality
  - If keys repeat with temporal locality, bucket will be in the cache
- Contention
  - If keys repeat too often in multiple threads, contention may occur for shared hash buckets

Modeling Performance

- Sample the input stream
- Add statistics gathering to hybrid approach
- Each thread gathers independent statistics
  - No coordination overhead
  - Each thread proceeds as fast as possible
  - Local decision may not be globally optimal
    E.g., "If every thread saw input like mine, there would be contention."



Why can’t the optimizer choose?

- Statistics might be wrong or inadequate
- Static choice cannot adapt to change in distribution
- Multiple operators to choose from versus one that works well; reduces plan space
- Aggregation occurs late in plans, other operators may have introduced skew

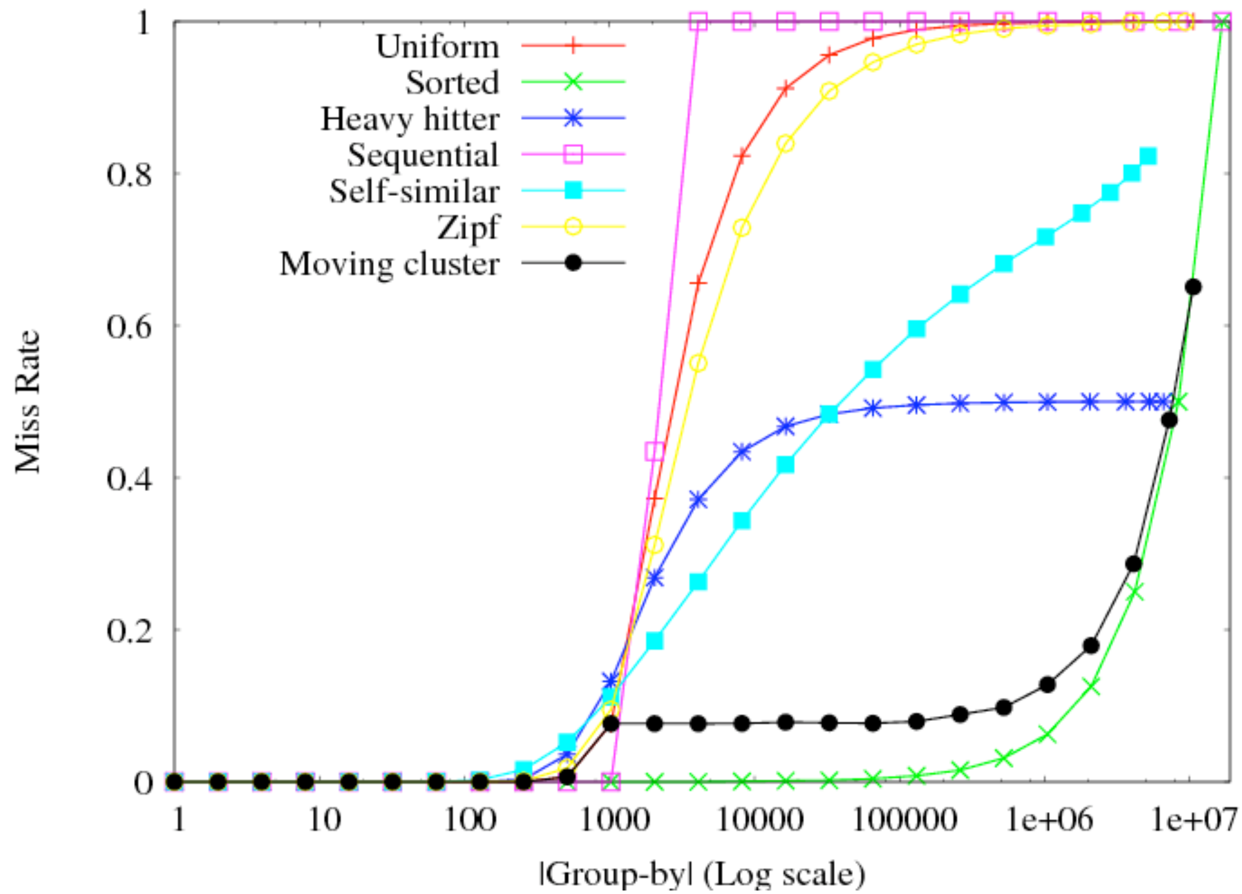
Sampling for Runs

- Count the number of runs seen during the sampling window, find average run length
- On uniform input, run optimization is beneficial up to |Group By| = 8
- Expected run length=$1+(1/8)^2+(1/8)^3...=8/7$
- Use run optimization if average run length exceeds 8/7

Sampling for Locality

- Count "hits" in the local table
  - A "hit" is when a key is found (no insertion)
  - Tables sized to fit in L2 (likely to be a $ hit)
- Avoid compulsory misses with "warm-up"
- Locality if miss rate is less than 50%
  - Derived by the relative cost of processing a tuple in the local table compared to the cost of using a global table

Miss rate



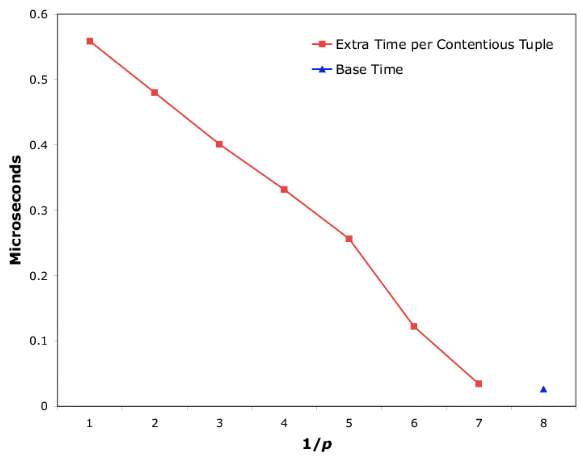
Uniform
Sorted
Heavy hitter
Sequential
Self-similar
Zipf
Moving cluster
Miss Rate
0
0.2
0.4
0.6
0.8
1
1
10
100
1000
10000
100000
1e+06
1e+07
|Group-by| (Log scale)

Contention

- Often subsumed by locality, except...
- Distributions with “heavy hitters” have contention without locality
- Global table must be avoided if there is contention because overhead dominates execution time

Modeling Contention

- Contention directly related to the proportion of the input with the same key.
  - Contention negligible below a measurable threshold
- See paper for model of contention
  - When contention is present, the penalty per contentious tuple is *linearly related* to the inverse of the key's frequency in the input



Sampling for Contention

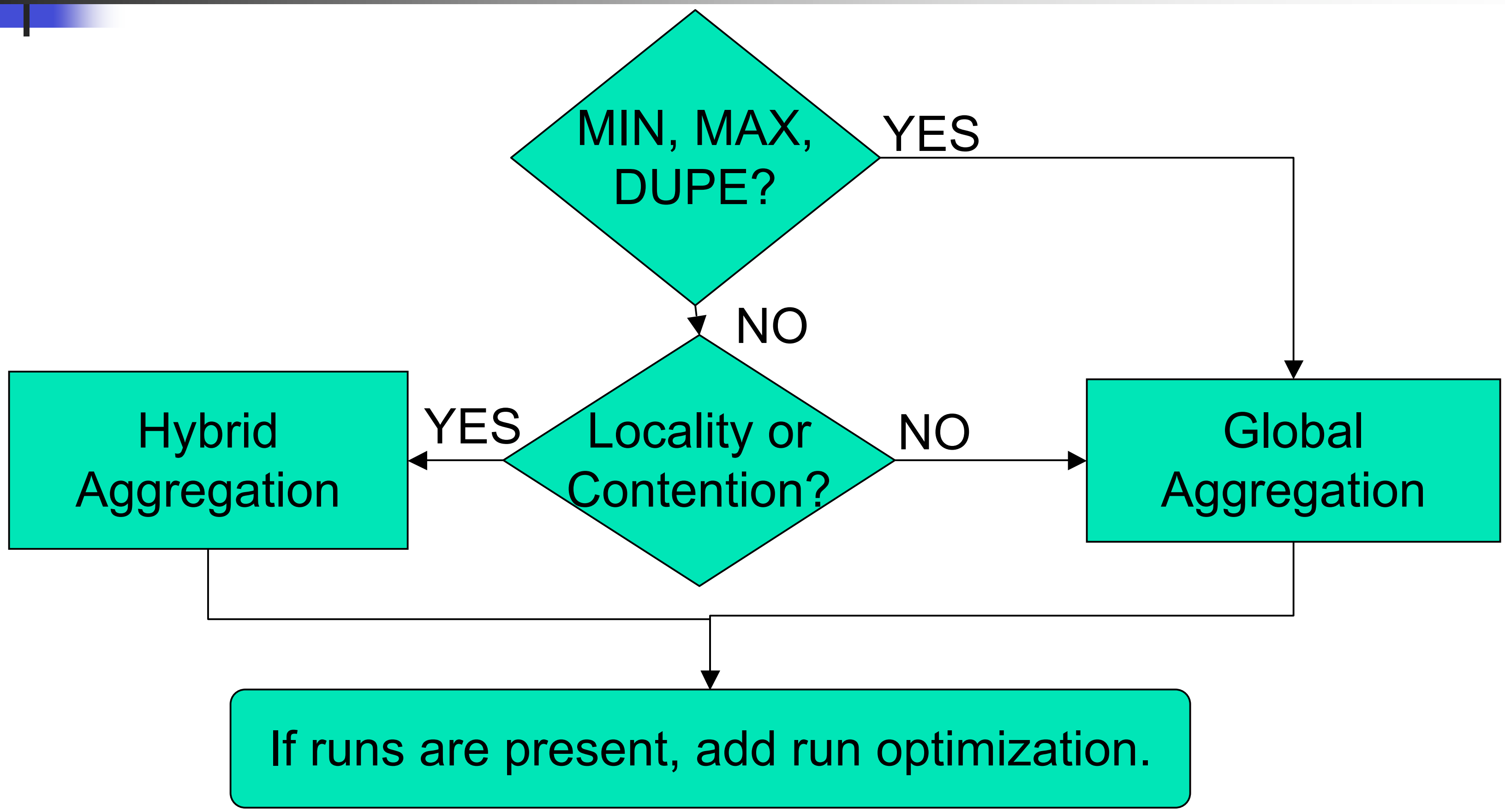
- Count accesses to each hash bucket
- If any bucket's access count exceeds a threshold, mark it as potentially contentious*
- Calculate the penalty due to contention of all marked contentious buckets*
- If the cumulative contention penalty is sufficiently high, flag the input as contentious*

*See paper for a full description

MIN, MAX, Duplicate Elimination

- Contention Free
- Why? Answer: Updates are rare
  - E.g., given a uniform input, after 99 inputs the running minimum is in the first percentile. The chance that the 100^{th} value will update the minimum is 1%
- Adversarial distributions exist, but can be handled with randomization

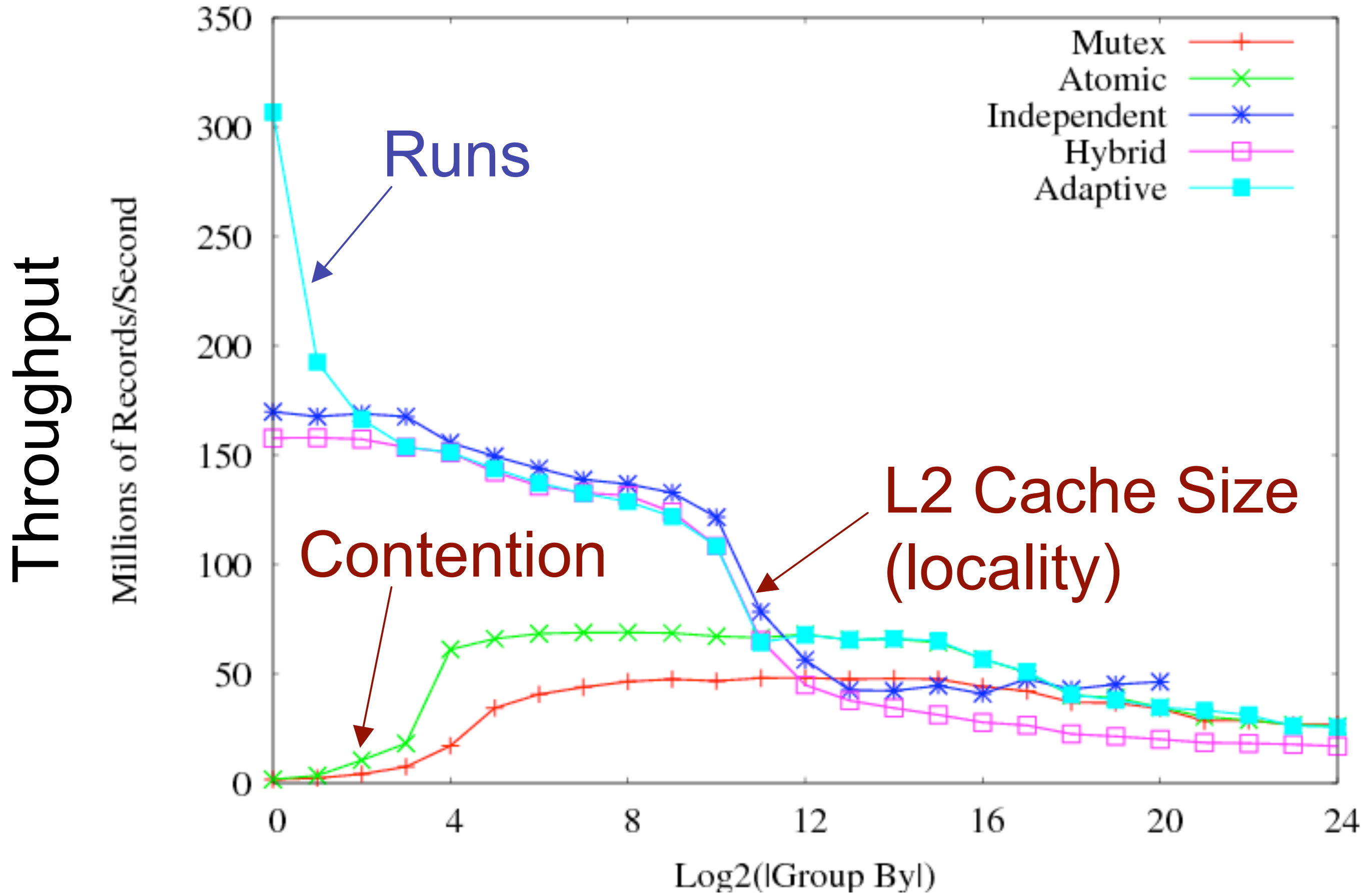
Adaptive Aggregation



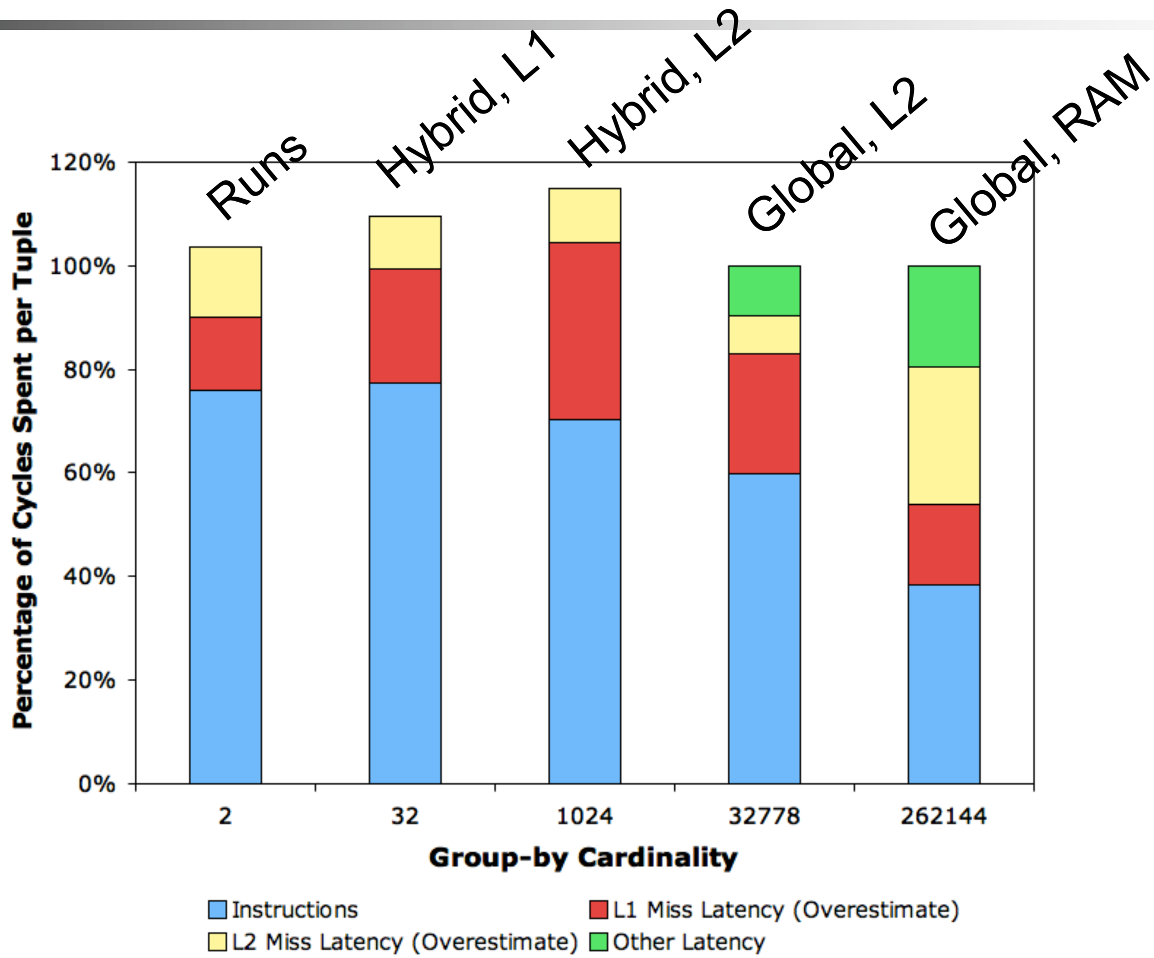
Experiments

- $2^{24} \approx 16$ Million Input Tuples
- 7 Input Distributions, 3 Queries
- Q1: `SELECT G, count(*), sum(V), sum(V*V) FROM R GROUP BY G`
- Q2: `SELECT G, max(V), min(V), max(V) FROM R GROUP BY G`
- Q3: `SELECT DISTINCT G FROM R`

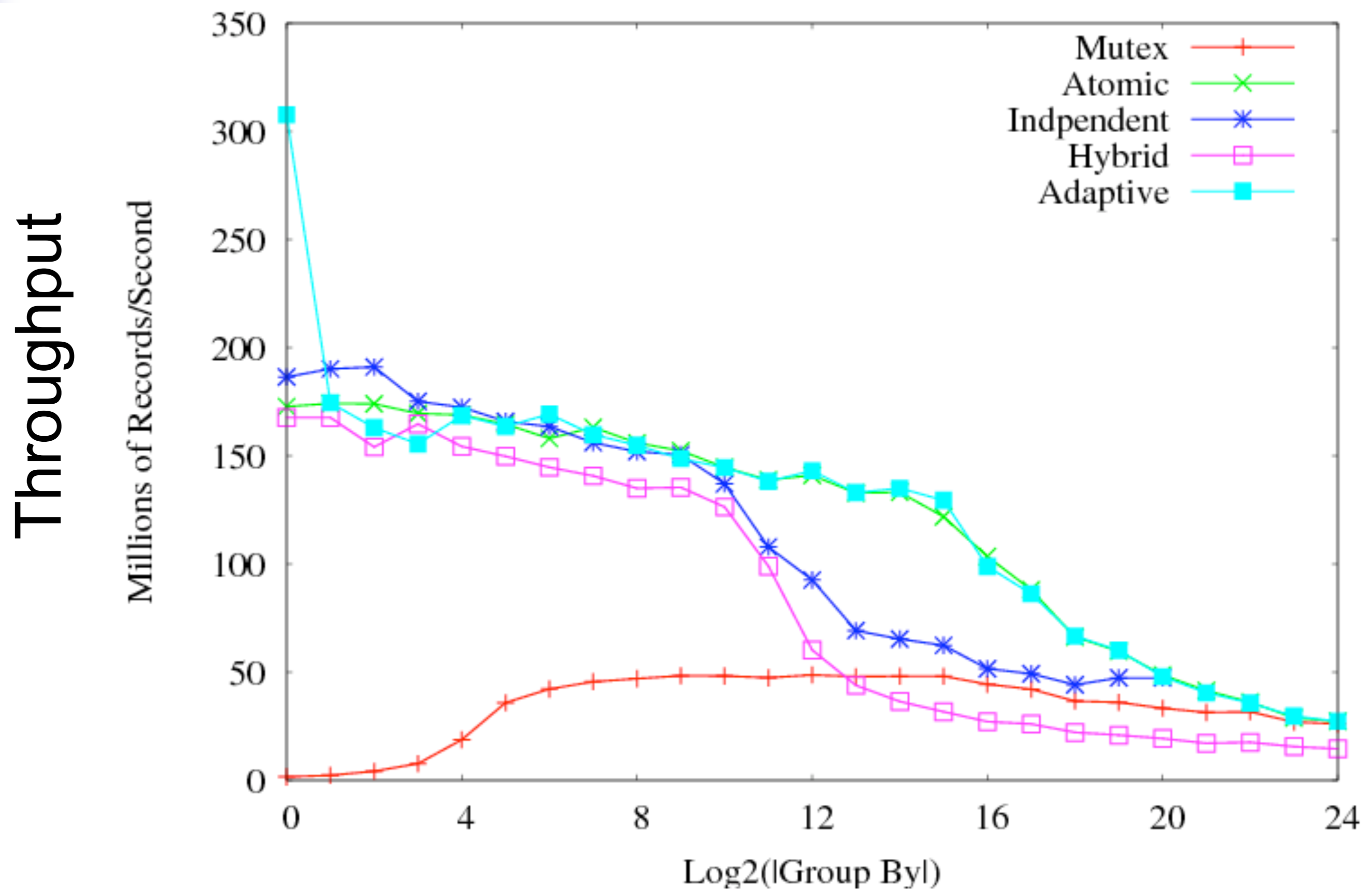
Q1 (sum & count): Uniform Input



Time Breakdown (Uniform Input)

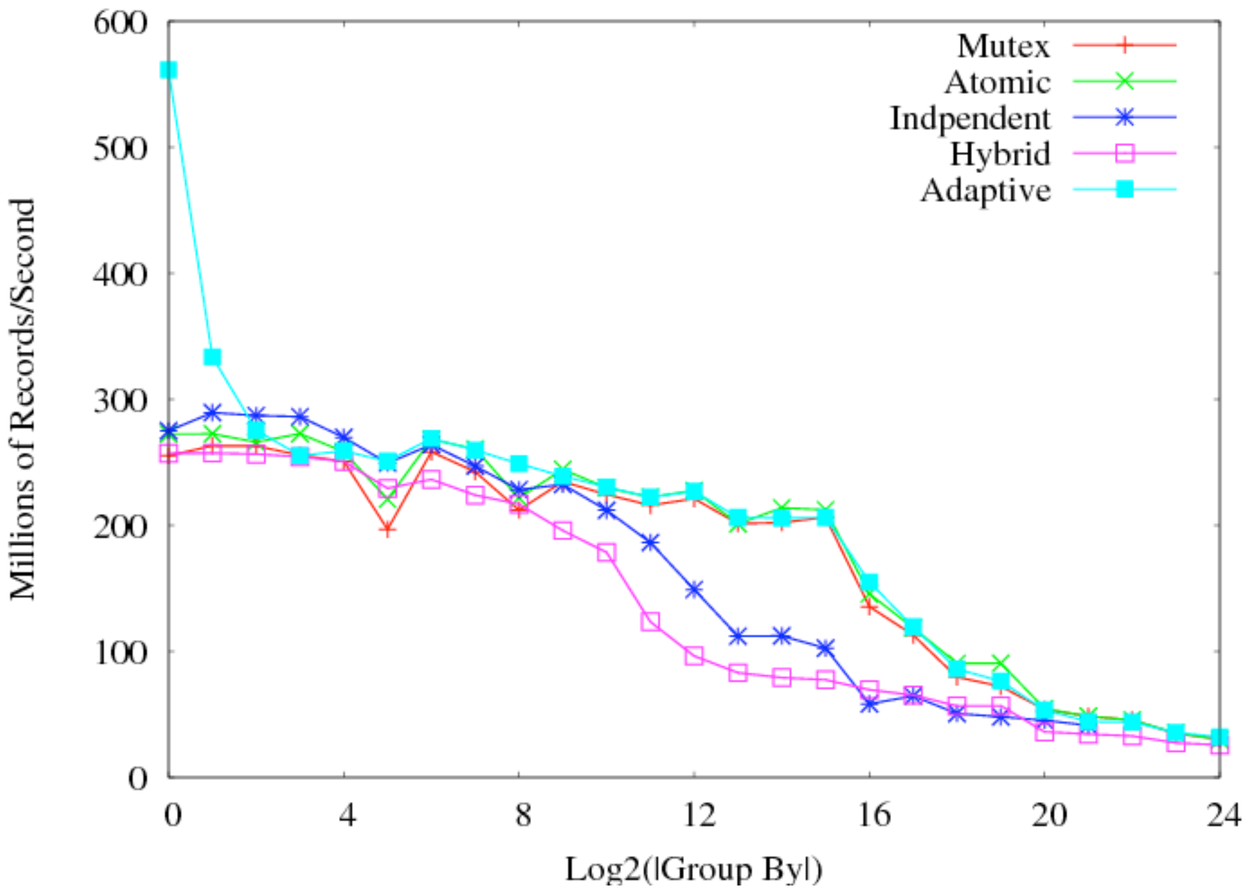


Q2 (min & max): Uniform Input

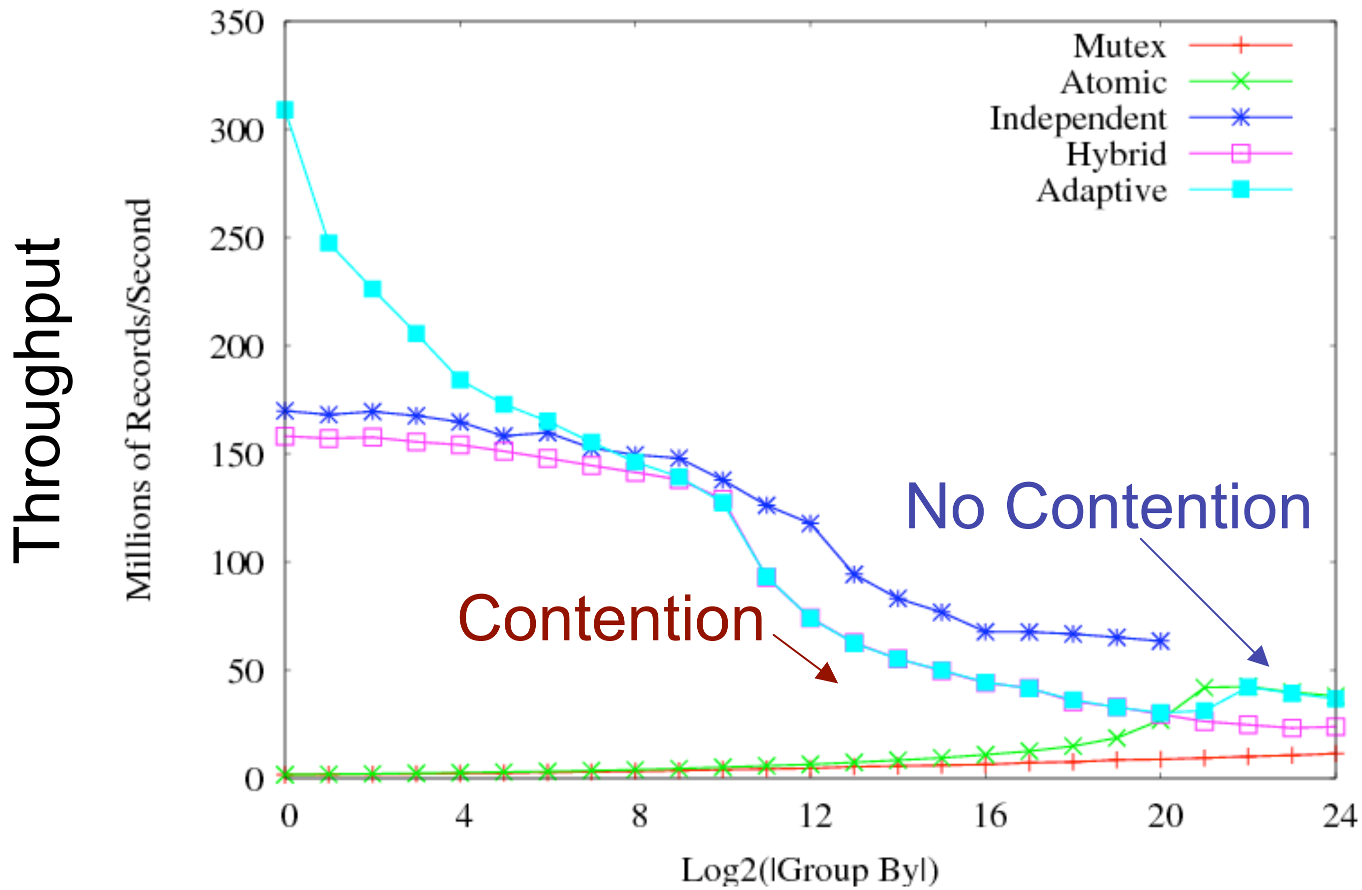


Q3 (DISTINCT): Uniform Input

Throughput



Q1 (sum & count): Self-similar



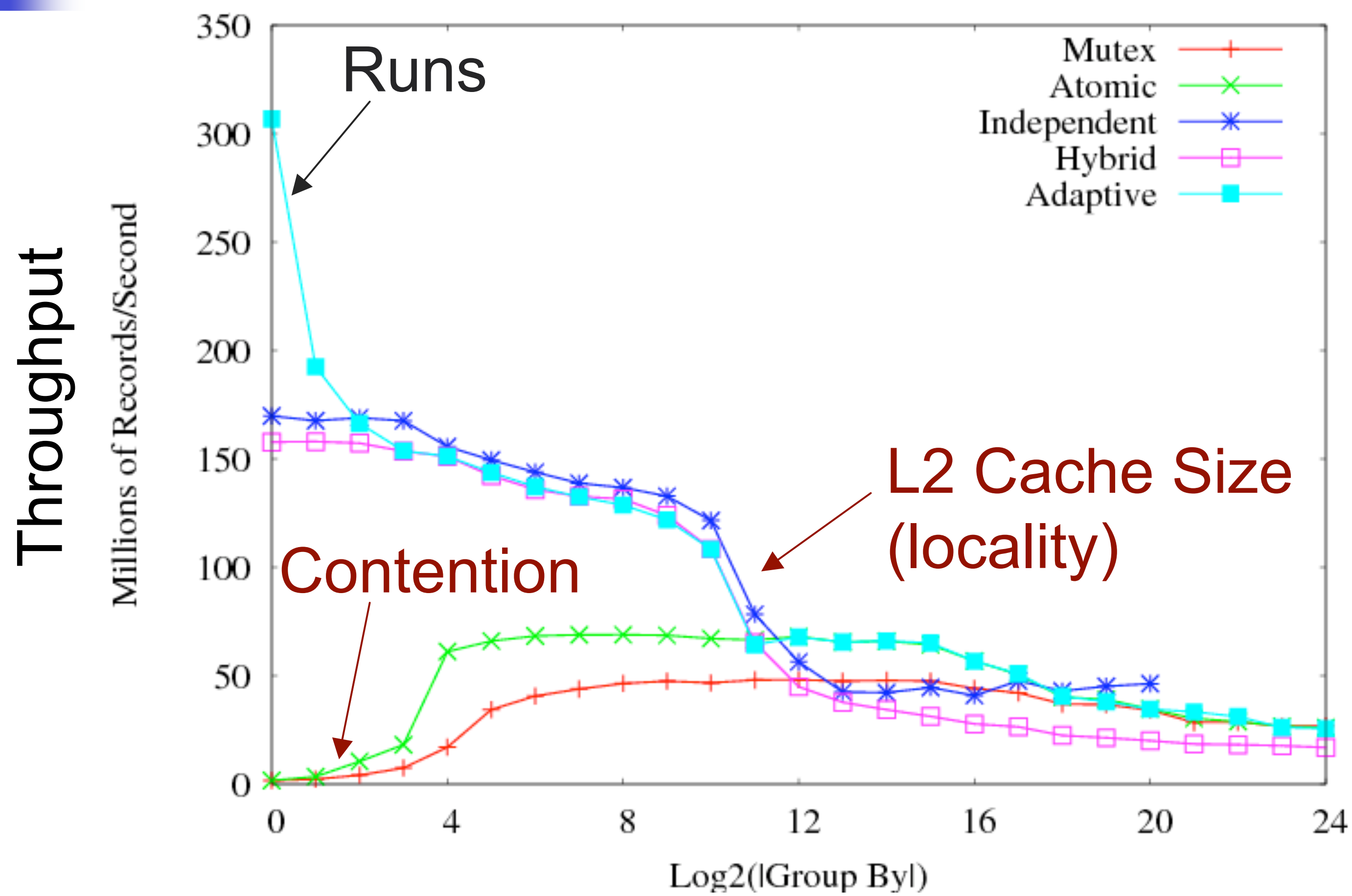
See the paper for…

- The full contention model
- Results with other input distributions
- The impact of resampling the input stream
  - Able to adapt to changes in the distribution
- Scales with the number of computed aggregates
  - Global table with mutex / lock eventually out performs atomic instructions

Conclusion

- Investigated aggregation performance on a real chip multiprocessor
- Identified locality and contention as key performance issues
- Introduced an adaptive aggregation operator that uses lightweight sampling to choose the best aggregation strategy

Das Ende



Throughput
Millions of Records/Second
350
300
250
200
150
100
50
0
0
4
8
12
16
20
24
Log2(|Group By|)
Mutex
Atomic
Independent
Hybrid
Adaptive
Runs
L2 Cache Size
(locality)
Contention