Cache-conscious Buffering for Database Operators with State

John Cieslewicz, William Mee, and Kenneth A. Ross

Columbia University

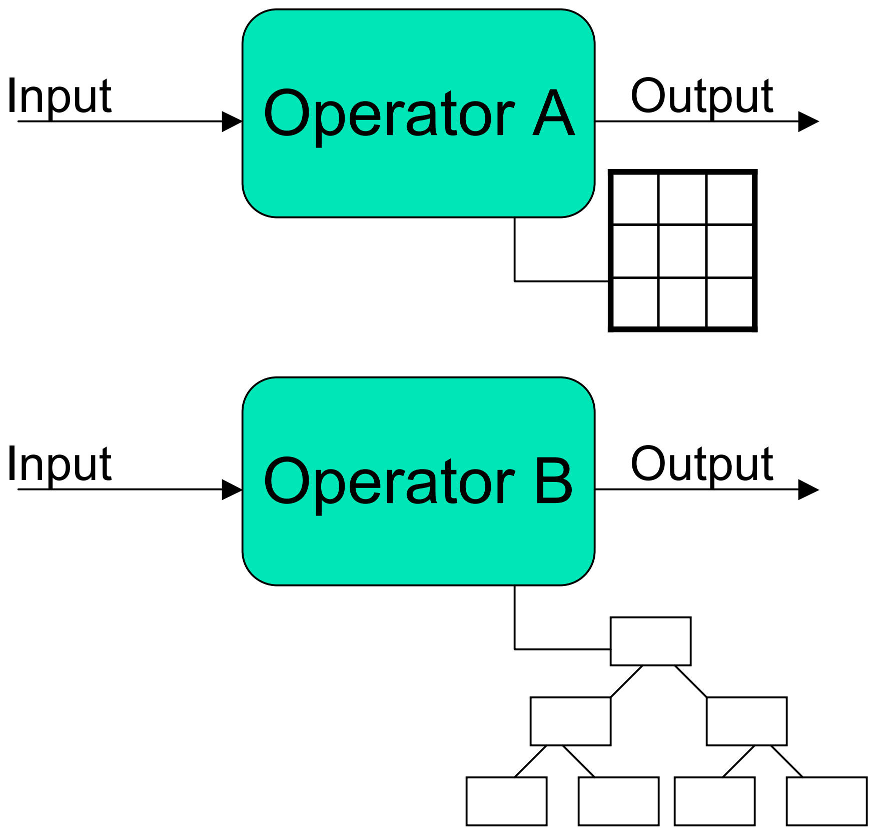
Motivation and Contributions

- Cache-conscious research for in memory OLAP operations has often focused on one operation at a time
- This work examines the impact (2x!) of temporal locality in the cache when multiple operators are used
- We will demonstrate the benefits of operator scheduling decisions based on minimizing measured L2 cache miss events

Operators with State

- Many database operators maintain state or use persistent data structures.
  - Hash tables
  - Indexes
- Processing multiple tuples takes advantage of temporal locality
  - Block processing already known to have good performance properties
- How should temporal cache information impact inter-operator scheduling and time slice (or block size)?

A Two Operator Example



- Two operators to be run by DBMS
- Both use a private data structure during processing
- Cache resident data structure = better performance

Scheduling and Cache Thrashing - Case 1

- *A* and *B* each use a data structure that is the size of the cache.
- *A* begins processing. Its data structure becomes cache resident after *n* tuples.
- If *B* is scheduled, it will suffer cache misses *and* evict *A*'s data.
- Scheduling blocks of fewer than *n* tuples means cache misses are never amortized.

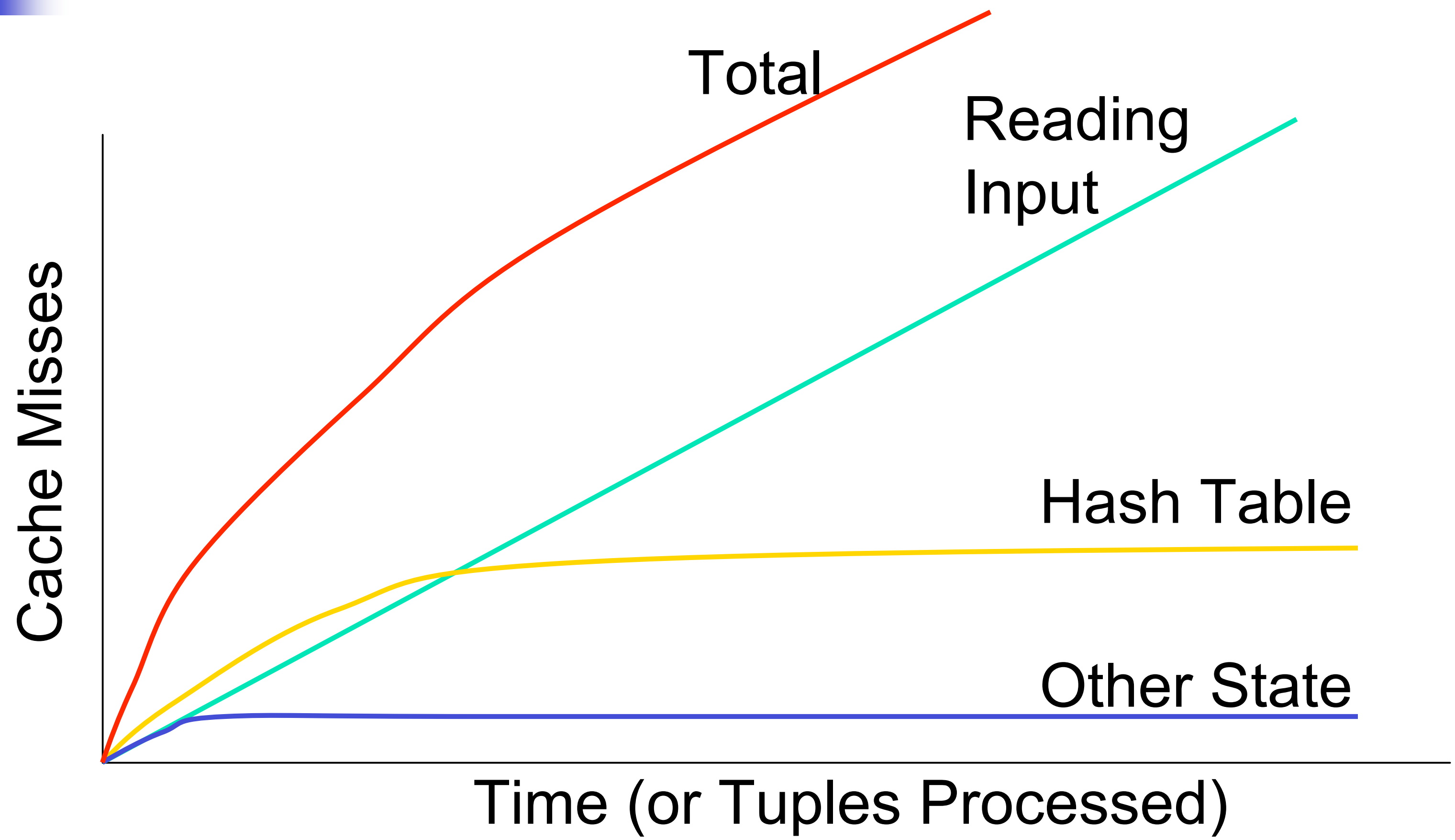
Scheduling and Cache Thrashing - Case 2

- *A* or *B* uses a data structure larger than the cache or simply scans many rows
- *A* begins processing, but data structure never becomes cache resident due to capacity constraints
- If *B* is interleaved with *A*, it suffers cache misses, but will not degrade *A's* performance

Scheduling and Cache Thrashing - Case 3

- *A* and *B* both use a small data structure
- *A* begins processing, data structure quickly become cache resident
- If *B* is interleaved with *A*, it also quickly becomes cache resident with *A.*
- *A* may still be evicted due to *B* scanning input
- But, small data structure size means cache miss cost to achieve cache residency is low

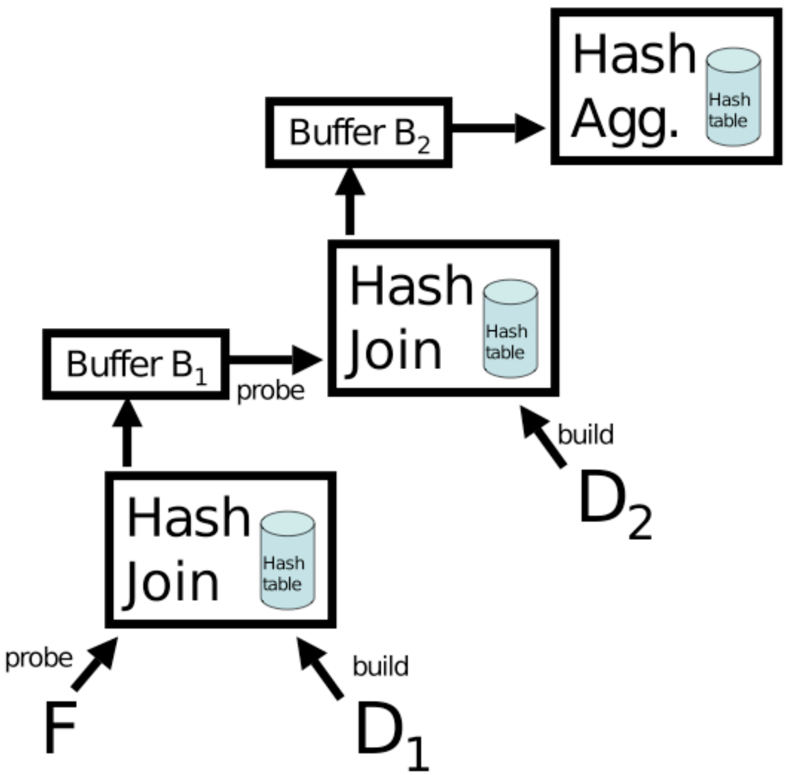
Operator Cache Miss Pattern



Some observations

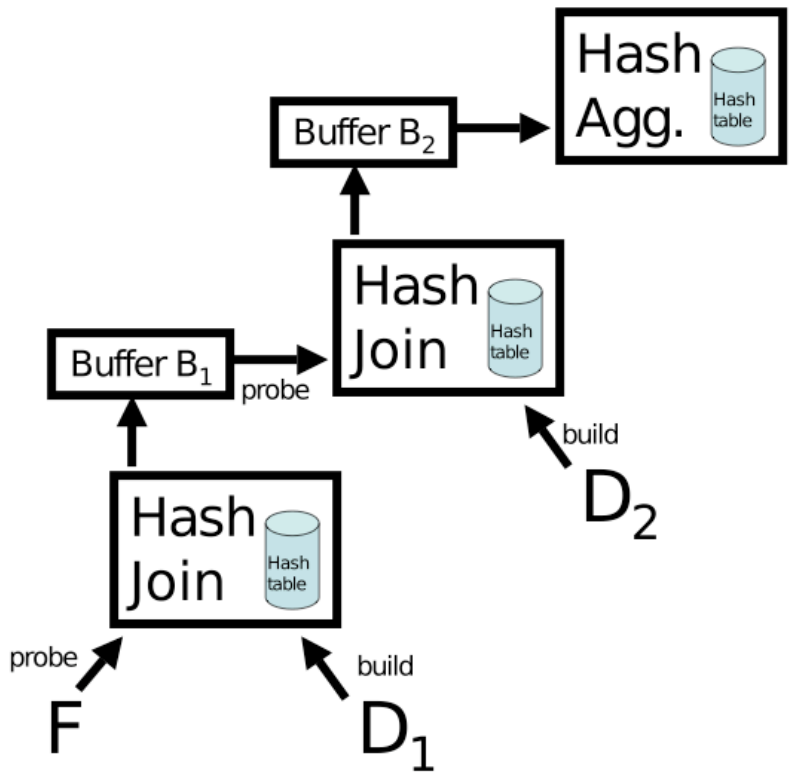
- If an operator takes longer to become cache resident, a larger block size is needed to amortize those cache misses
  - How to differentiate between an operator that takes a long time to become cache resident and that never is?
- The cache miss behavior is dependent on the operator *and* the data
  - Block size must be tailored to each operator instance using runtime performance monitoring

Amortizing Cache Misses, Enhancing Temporal Locality



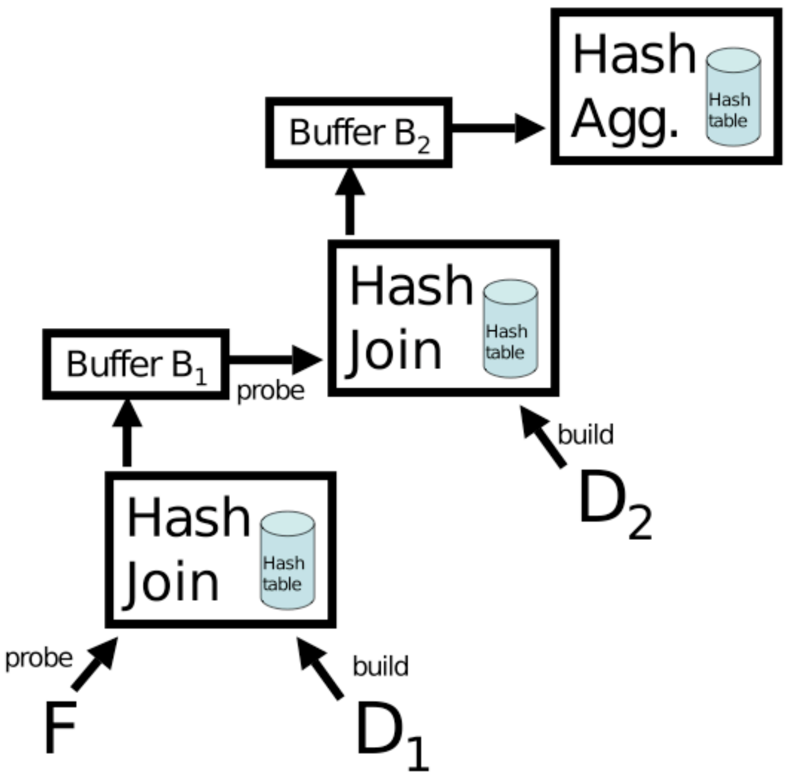
- We assume a block-oriented processing
- We insert buffers to ensure blocks of a certain size
- At runtime we determine the buffer size that amortizes cache misses and ensures fair execution

Choosing the block size



- Use hardware performance counter to measure cache misses per tuple, *c*
- *c* depends on the buffer size
- We also determine *r*, the rate at which tuples are produced
  - *r* may change, for this paper we assume it does not
- *cr* = cache misses/time

Choosing the block size



- We will assume *r* is fixed for all operators
- Given B_1 and B_2, the cache miss cost is $r(c_1+c_2+c_3)$
  - Adding capacity to B_1 may reduce c_1 and c_2 (c_1' and c_2')
  - Adding memory to B2 may reduce c_2 and c_3 (c_2'' and c_3'')
- Compare new costs:
  - $r(c_1'+c_2'+c_3)$
  - $r(c_1+c_2''+c_3'')$

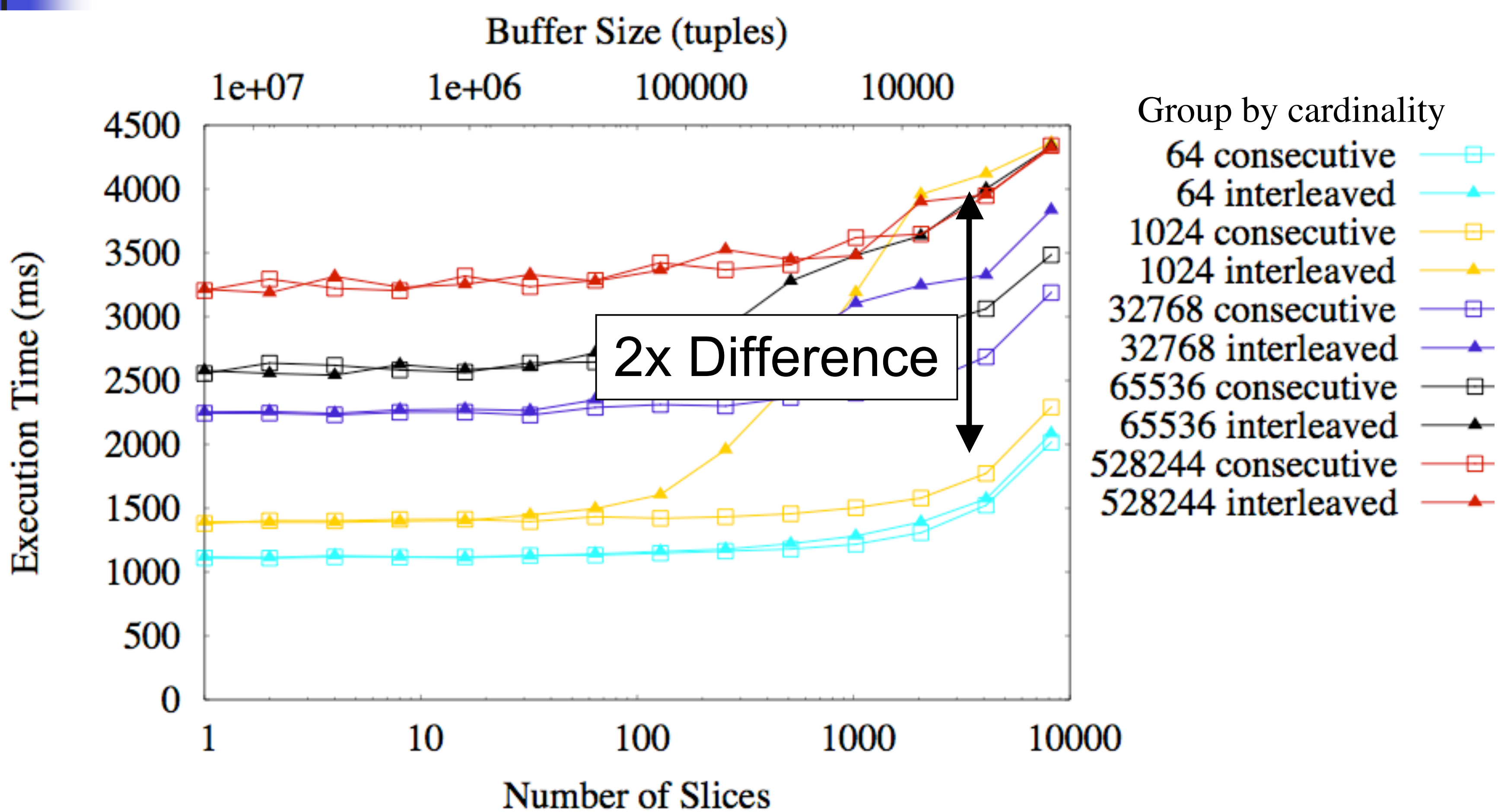
Choosing the block size

- Make buffer capacity allocations that reduce cache misses, improving performance
- Operators should only be run if:
  - Enough input tuples + output space to ensure a sufficiently long time slice
- There is always an operator that can be run.
  - See paper for a proof.
- Hard ceiling on buffer size prevents starvation of other queries

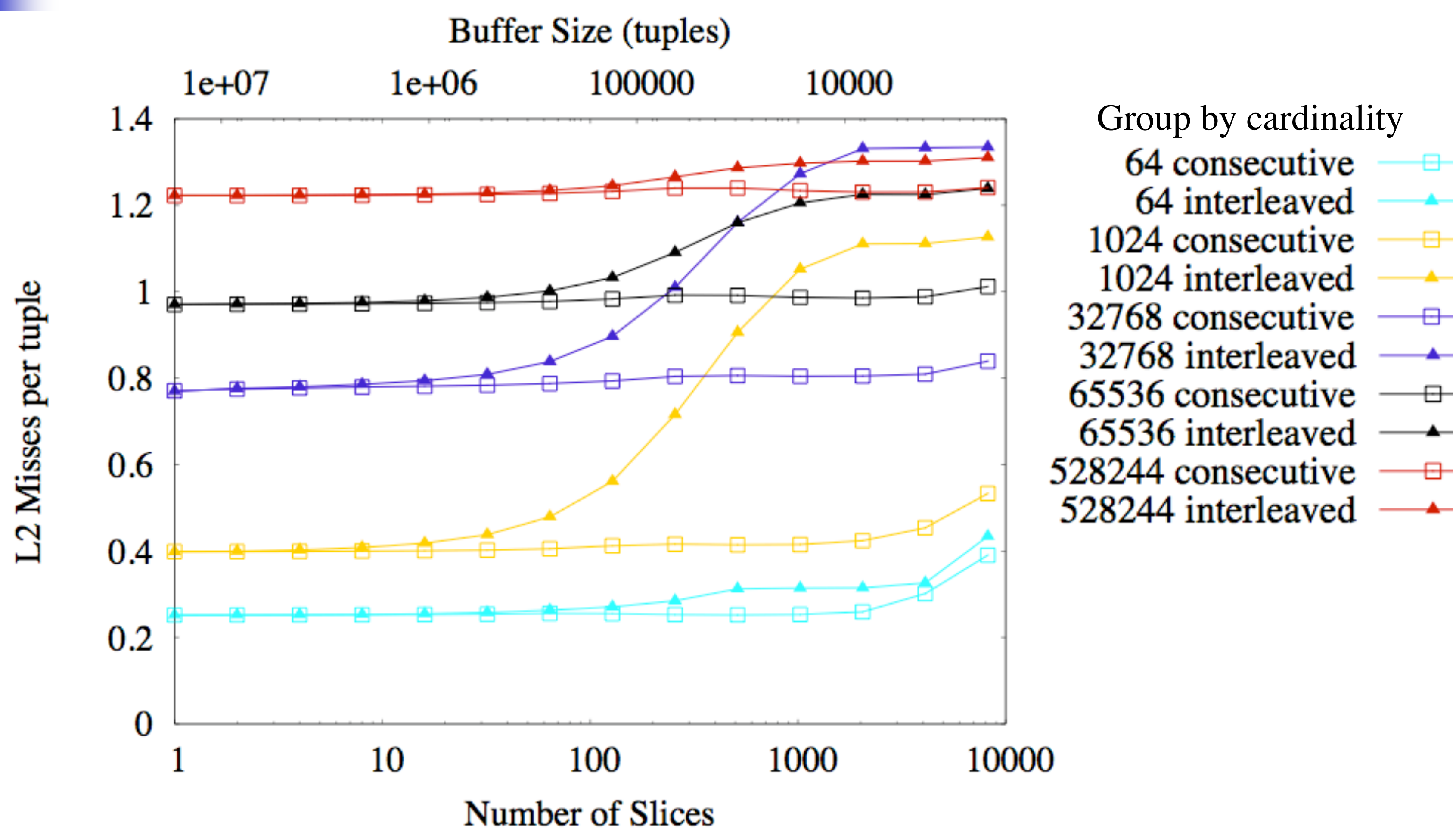
Experimental Setup

- Sun UltraSPARC T1000
  - 32 hardware threads over 8 cores
  - 3MB shared L2, 12-way associative with 64B cache lines
- We devote 31 threads to computation, one thread to coordination
- Workload:
  - Adaptive Aggregation [VLDB '07]
  - Hash Join
- Data:
  - 2^{24} tuples, uniform and self similar distributions
  - For hash aggregation, different group by cardinalities

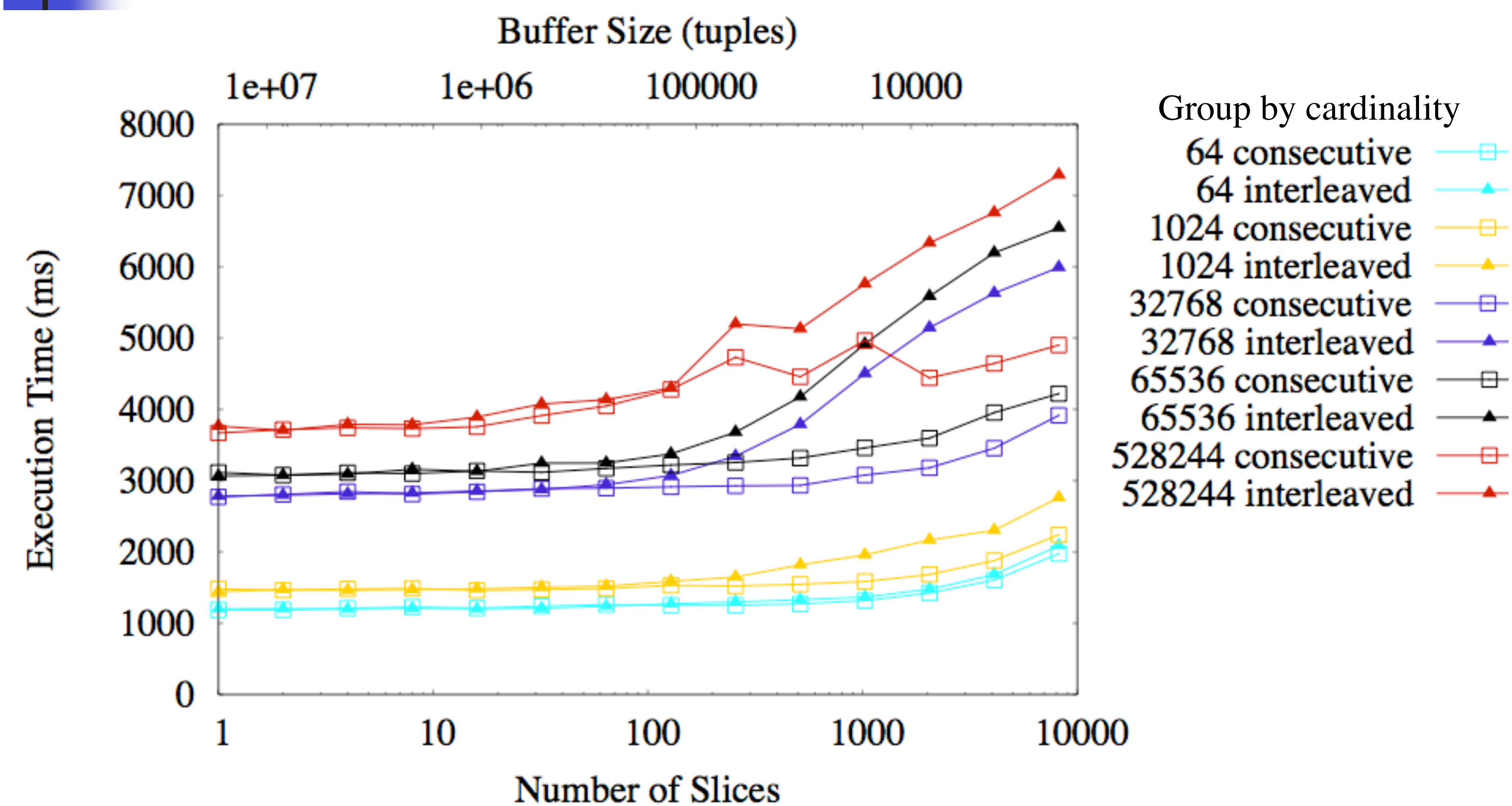
Effect of Interleaving Queries Uniform [Execution time]



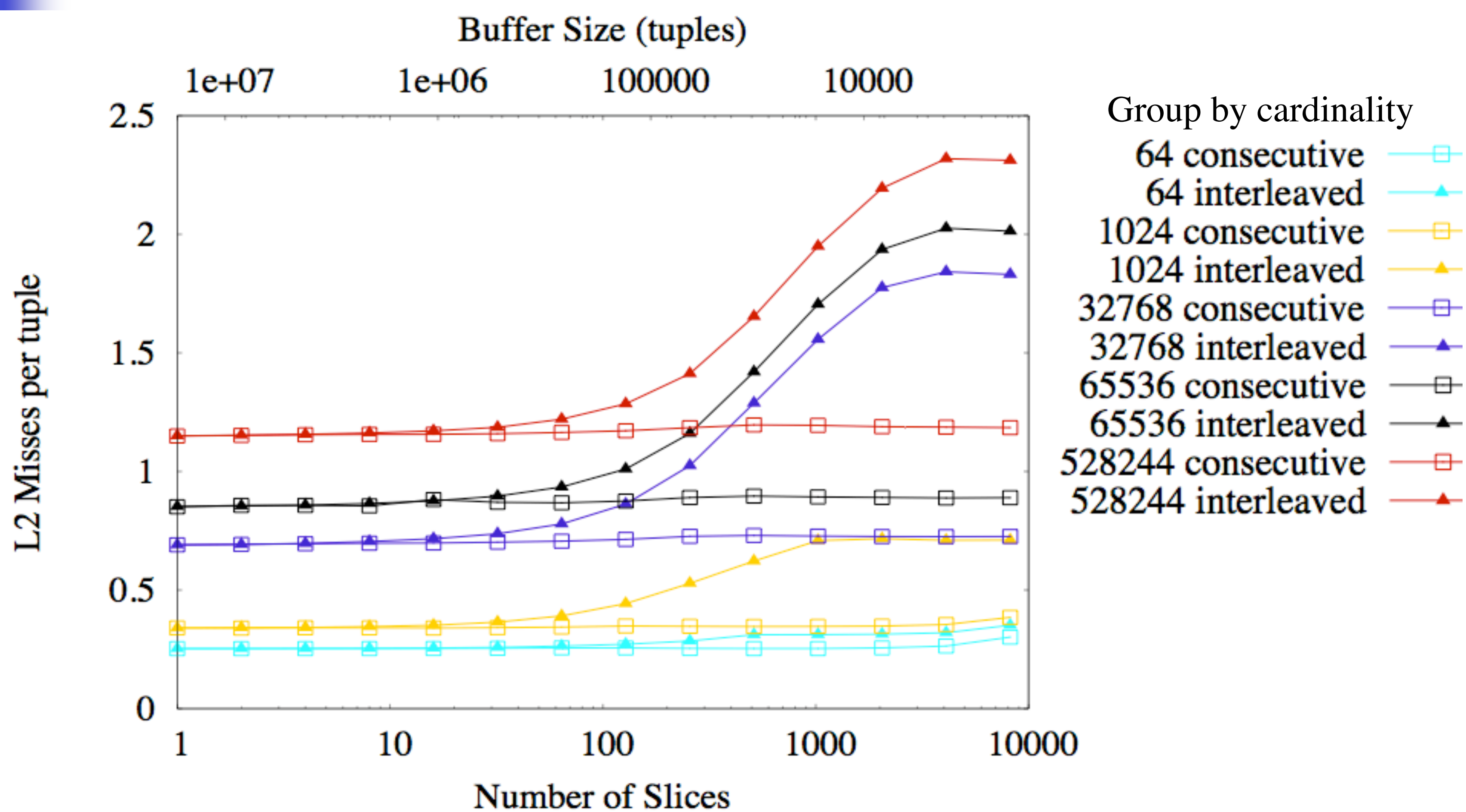
Effect of Interleaving Queries Uniform [L2 Cache Misses]



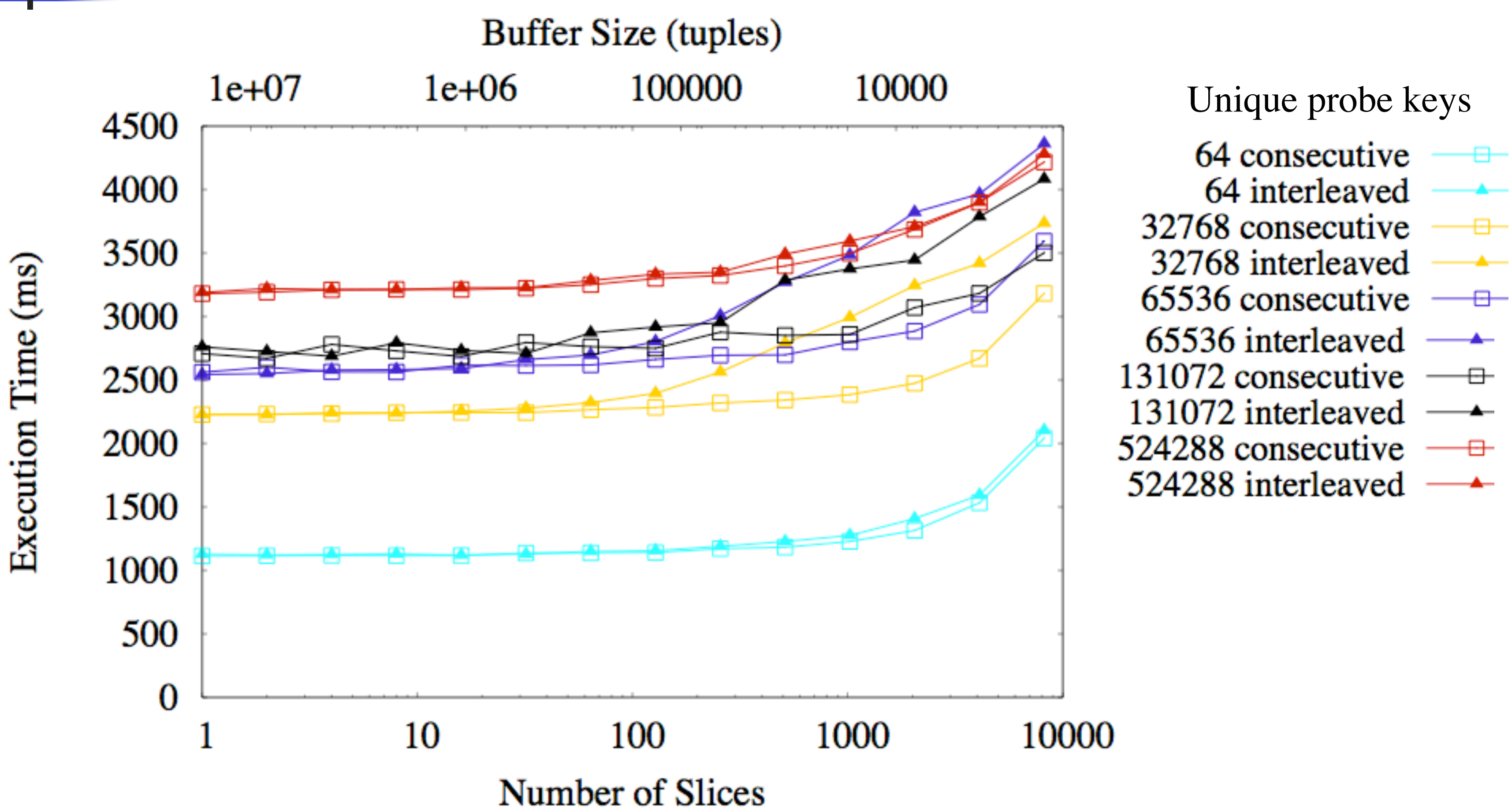
Effect of Interleaving Queries Self similar [Execution Time]



Effect of Interleaving Queries Self similar [L2 Cache Misses]



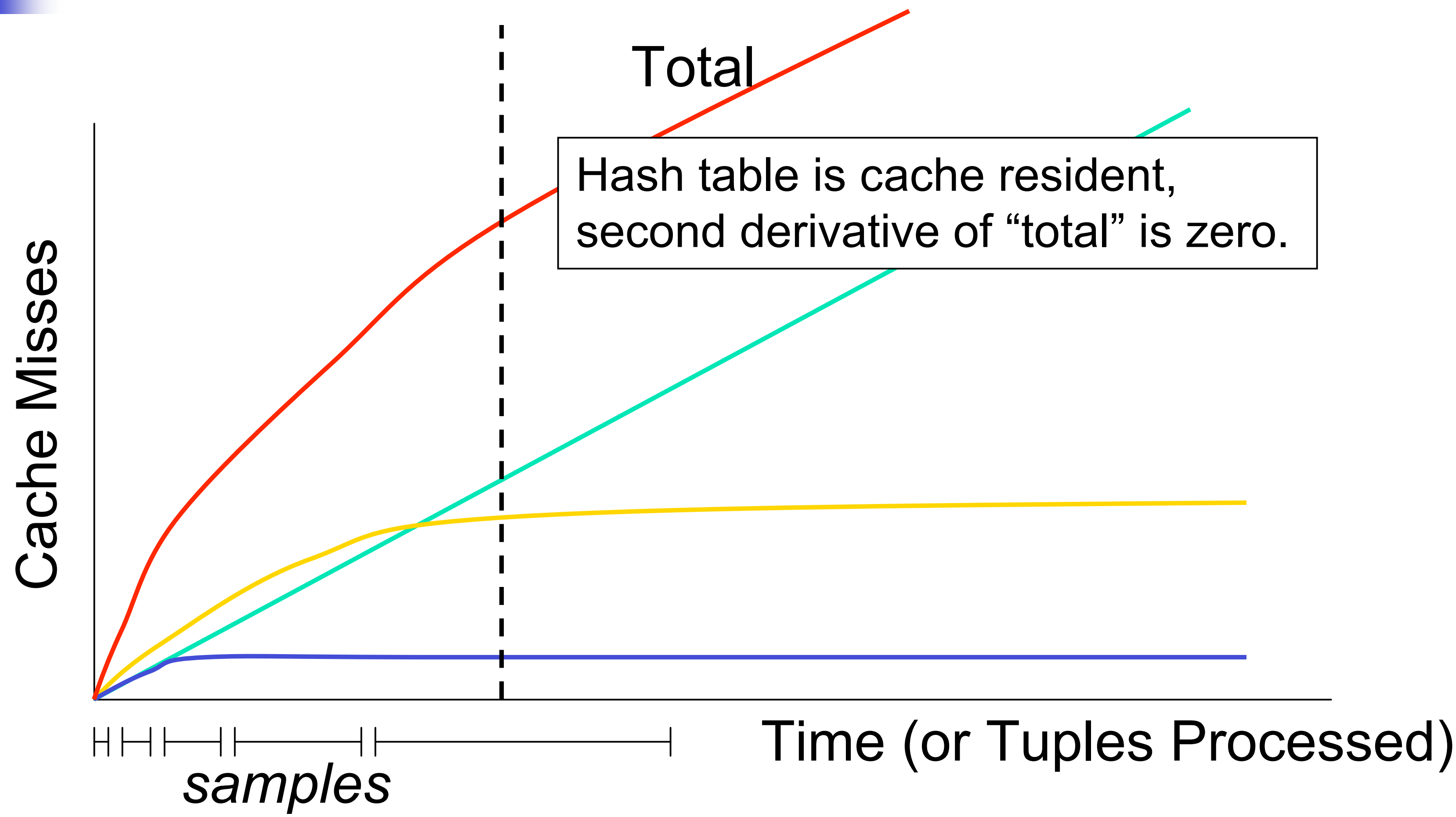
Effect of Interleaving Queries [Hash Join]



Runtime Performance Monitoring

- Hardware counters for events such as cache and TLB misses
  - Negligible performance impact to access these counters (we use them sparingly)
- Determining cache behavior of an operator instance:
  - We sample the number of accumulated L2 misses before and after an operator is run
  - Without interleaving another operator, run the operator again with a larger batch size
  - Look for point at which rate of cache miss change stops changing (second derivative)

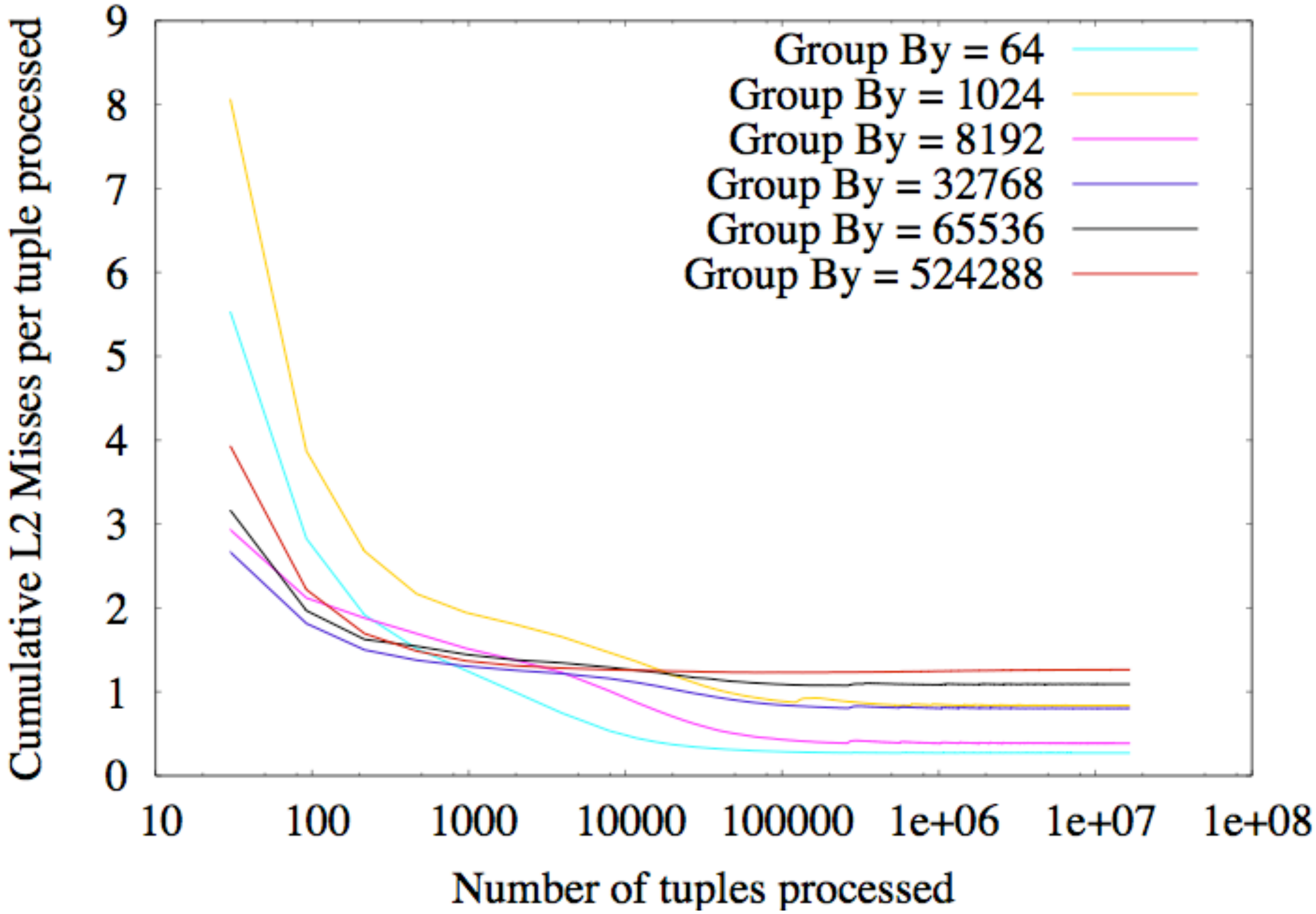
Sampling to determine the cache miss pattern



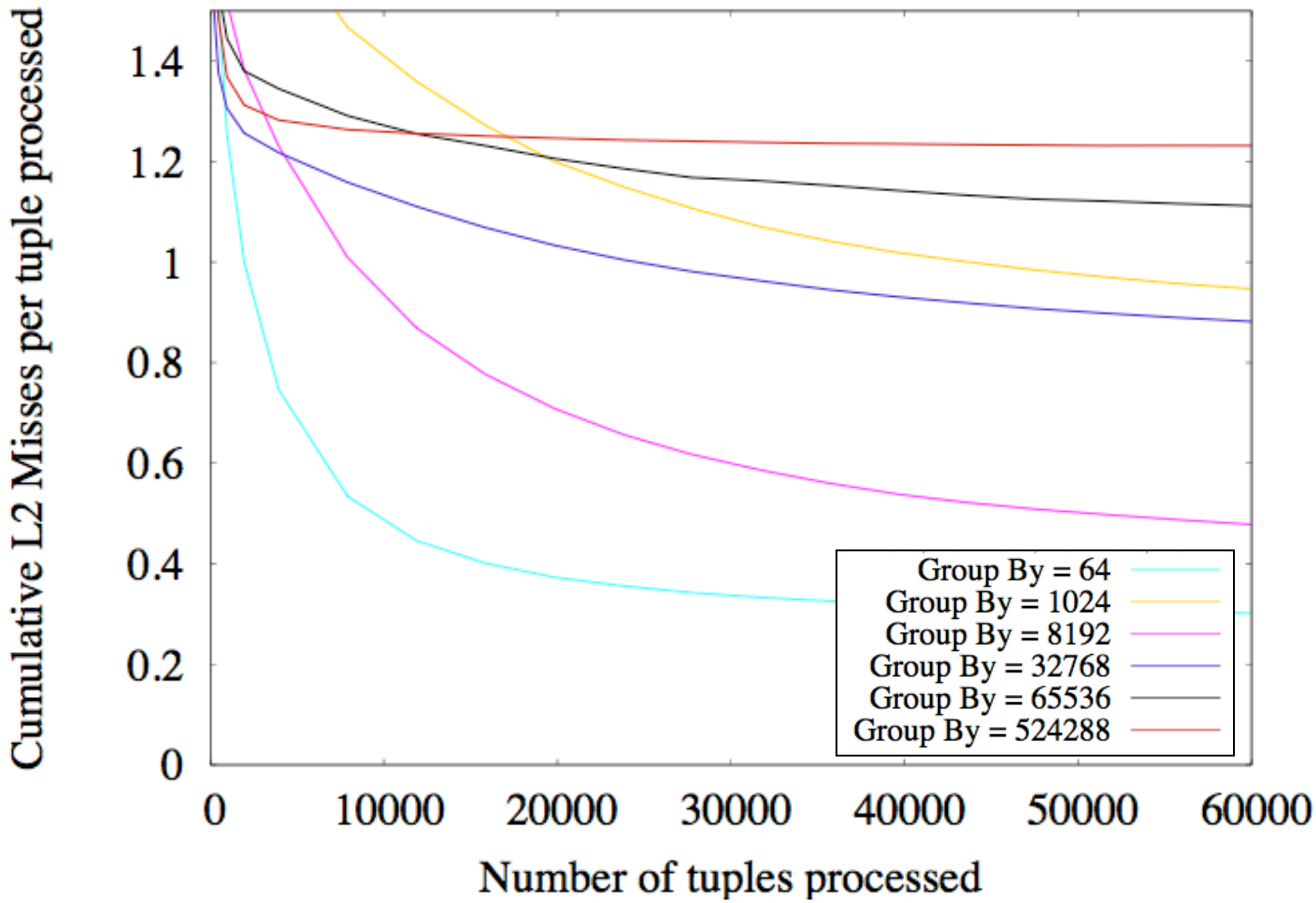
But, wait! I'm not ready…

- Some operators have phases of operation when cache misses may not be representative of the steady state.
  - The adaptive aggregation operation has a sampling phase of its own that can be considerably different
- Sampling interface exposes a "Don't sample me now" flag.

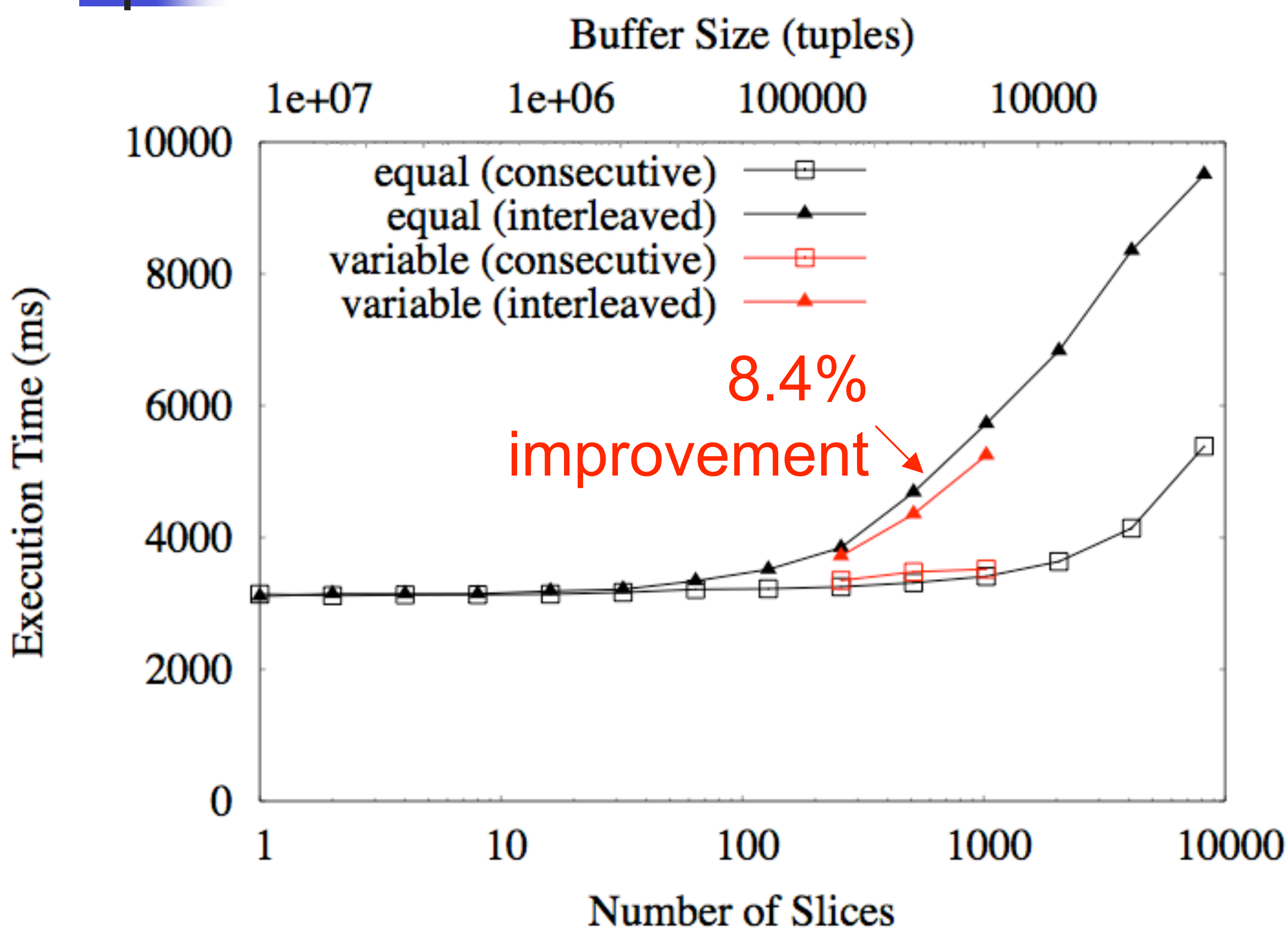
Cumulative Cache Misses per Tuple (Aggregation)



Cumulative Cache Misses Per Tuple (Aggregation) -- Detail

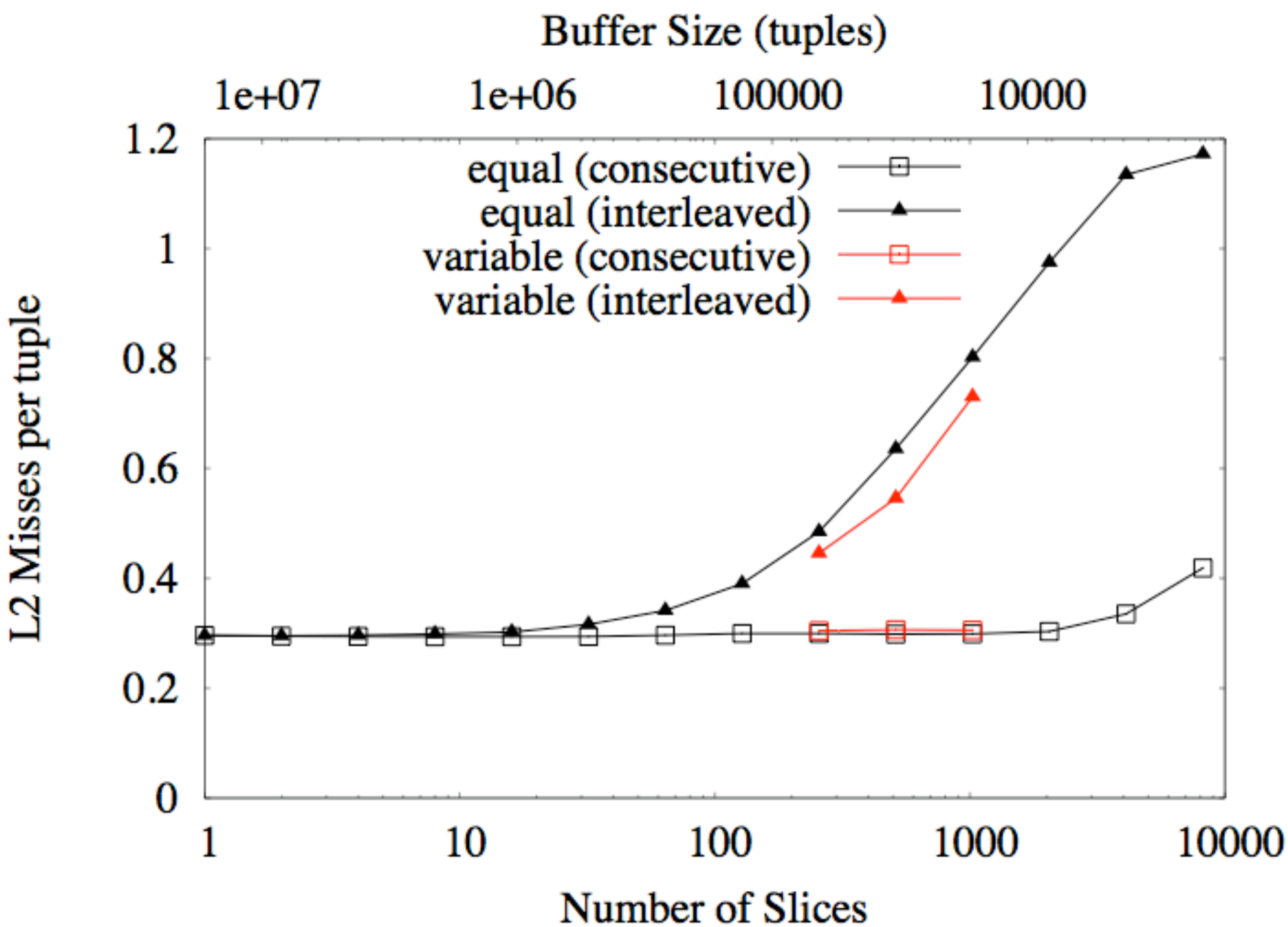


Variable Buffer Size [Execution Time]



- Allocate memory non-uniformly among ten aggregate operators.
- 20 operators, half with group by cardinality = 1024, half = 64

Variable Buffer Size [L2 Cache Misses]



Future Work

- Consider other modes of parallelism
  - multiple operators at the same time means more types of cache interference
- Improve the model. In these experiments we considered *r* the rate tuples are produced to be fixed.
- Group multiple operators with small state needs together in one execution chain

Conclusion

- Scheduling for temporal locality of database operator state is important
  - 2x improvement in some cases
- Large buffers (MB) can be useful
- Allocating buffers most useful for operators with state sized close to L2 capacity
  - Smaller state doesn't cost much to load
  - Larger state will never be cache resident

L2 Misses [Hash join]