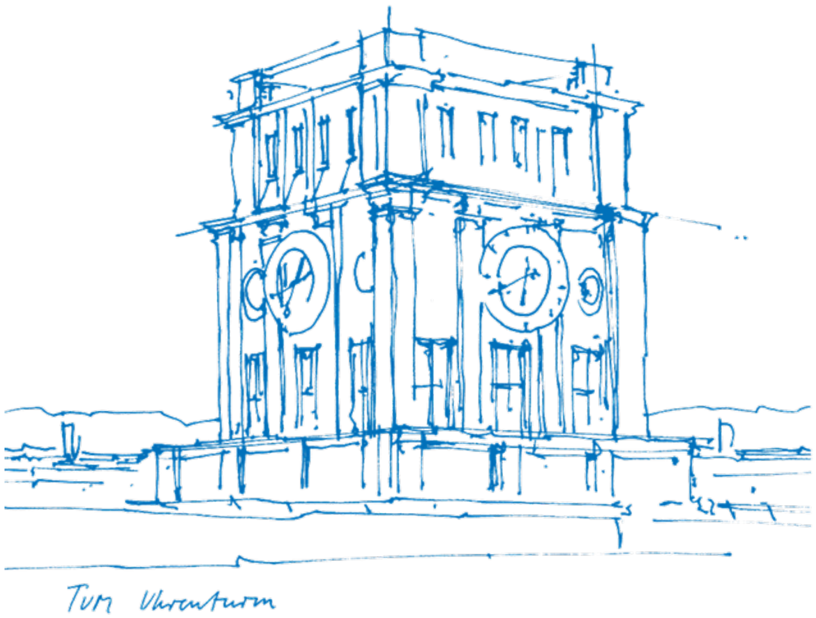
Hash Joins for Multi-core CPUs

Benjamin Wagner



Joins

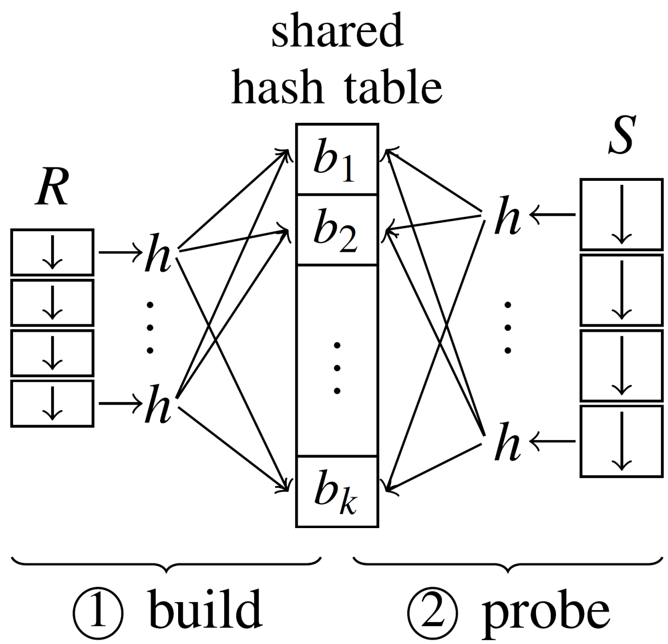
- fundamental operator in query processing
- variety of different algorithms
- many papers publishing different results
- main question: is tuning to modern hardware worth it?
- goal: perform own benchmarks on these algorithms
- only main memory hash joins are considered

Problem Statement

- two relations R=[value, ID], S=[value, ID]; usually $|R| \leq |S|$
- R is build relation, S is probe relation
- joining the relations on ”value” to produce triples [value, idR, idS]
- $R \bowtie_p S = \{x \circ y | x \in R \wedge y \in S \wedge p(x, y)\}$ with p = ”R.value=S.value”
- performance: bag instead of set semantics

No Partitioning Join (NOP)

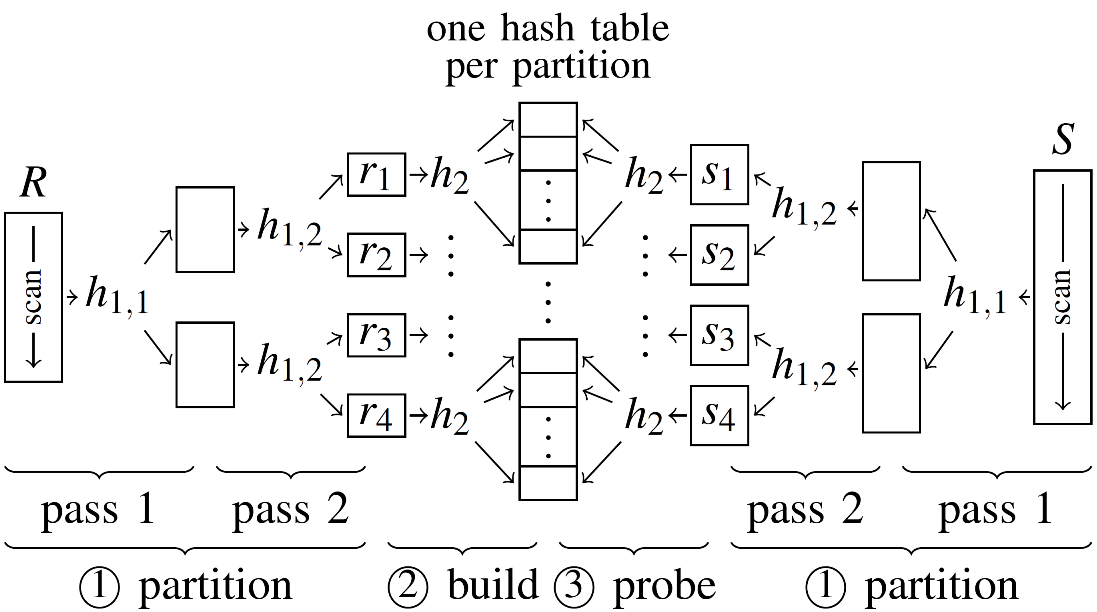
- build shared hash table with relation R
- probe the table with tuples from S



No Partitioning Join, taken from [1]

Radix Join (RPJ)

- partition R and S in one or more runs
- run regular NOP Join on separate partitions



Radix Join, taken from [1]

Radix Join (RPJ)

- partitioning happens on the least significant bits of the hash
- simple example with hash(x)=x and 2 pass partitioning
- pass 1 using 3 Bits

Value	Hash	Bucket (Pass 1)
12	1**100**	4
16	10**000**	0
121	1111**001**	1
412	110011**100**	4
2	**10**	2
523	1000001**011**	3
672	1010100**000**	0

Radix Join (RPJ)

- partitioning happens on the least significant bits of the hash
- simple example with hash(x)=x and 2 pass partitioning
- pass 2 using 3 Bits

Value	Hash	Bucket (Pass 2)
12	**001**100	4.1
16	**010**000	0.2
121	1**111**001	1.7
412	110**011**100	4.3
2	**000**010	2.0
523	1000**001**011	3.1
672	1010**100**000	0.4

Radix Join (RPJ)

- several knobs influencing performance
- can lead to improved data locality during the join
- parallelization is rather involved

$\Rightarrow$ overhead of partitioning vs data locality

Literature

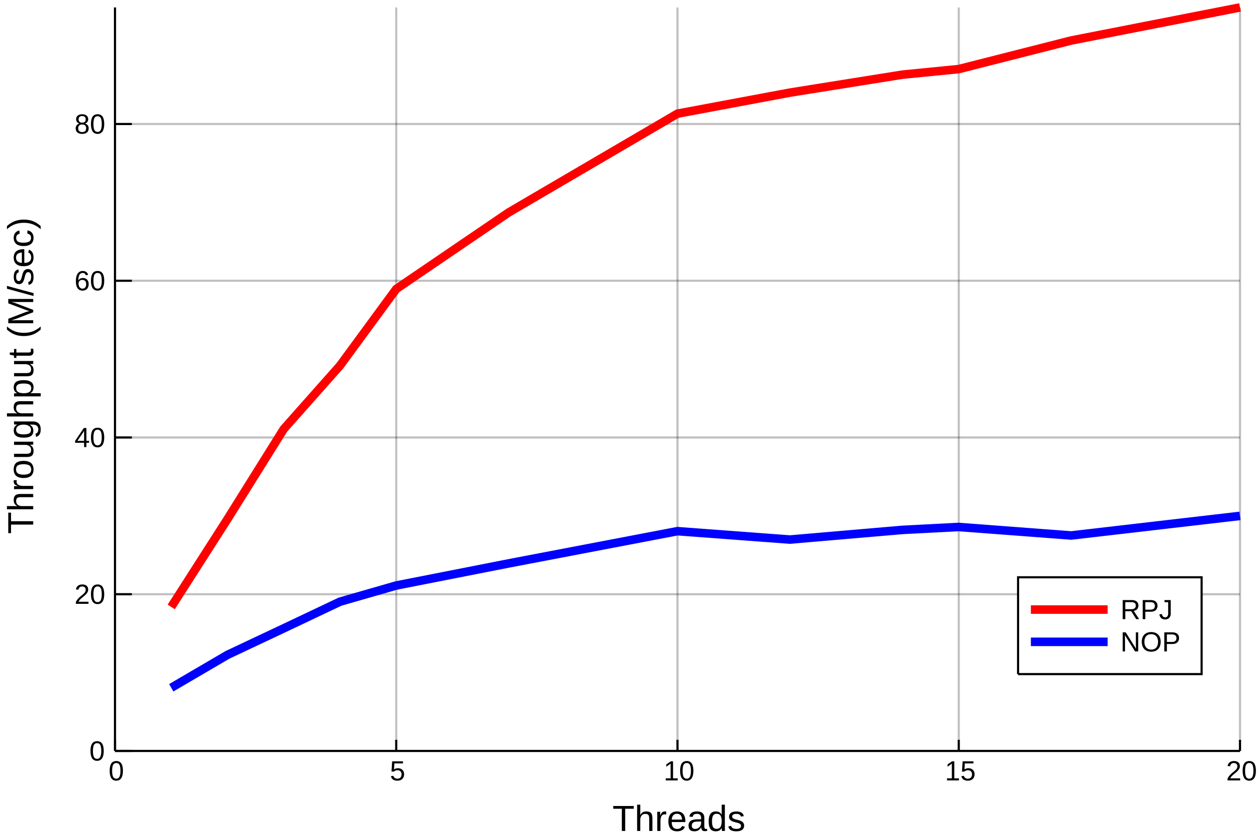
- idea to use radix partitioning is fairly old (1999) [4]
- since then: variety of papers claiming different things
- some say: algorithms should be hardware conscious [1, 2, 5]
- others: modern CPUs can hide cache miss latencies [3]

Implementation & Benchmarks

- implemented single and mutli threaded versions
- multi threaded RPJ utilizes single threaded NOP
- benchmarks taking several parameters into account
- radix join times always under optimal parameters

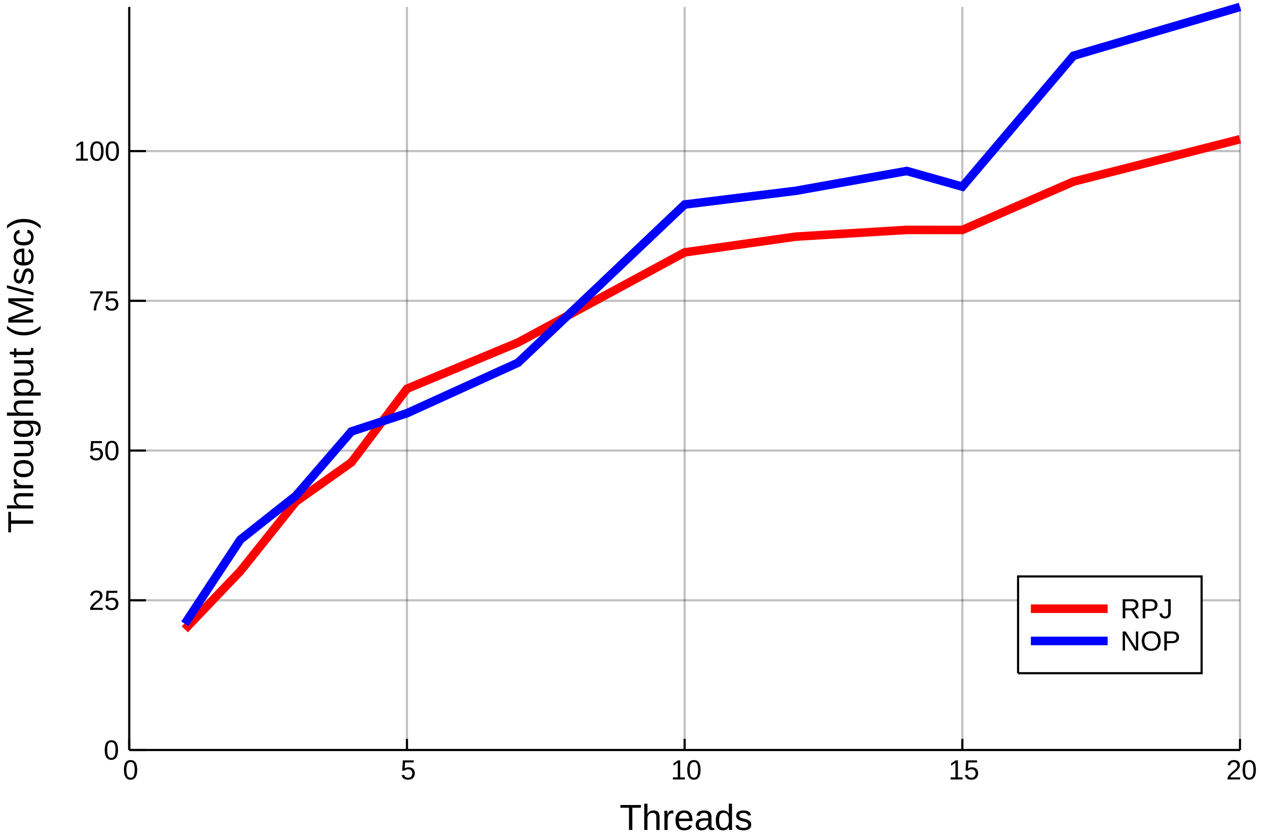
CPU:	Intel i9-7900X
# of Cores	10
# of Threads	20
Base Frequency	3.30 GHz
L1 Data Cache (per core)	32 KiB
L2 Cache (per core)	1 MiB

Benchmarks - RPJ vs NOP



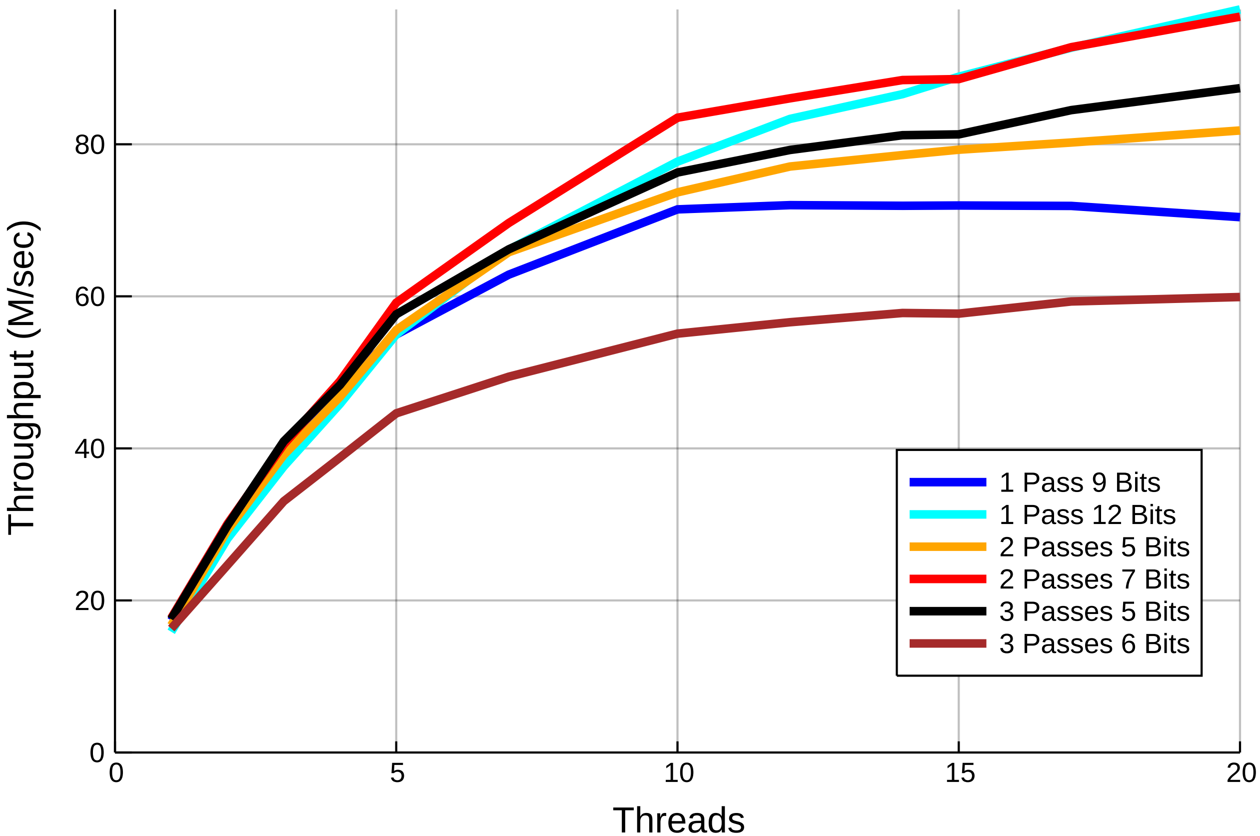
S uniformly distributed, $|R| = |S|$ ≈16.8M

Benchmarks - RPJ vs NOP



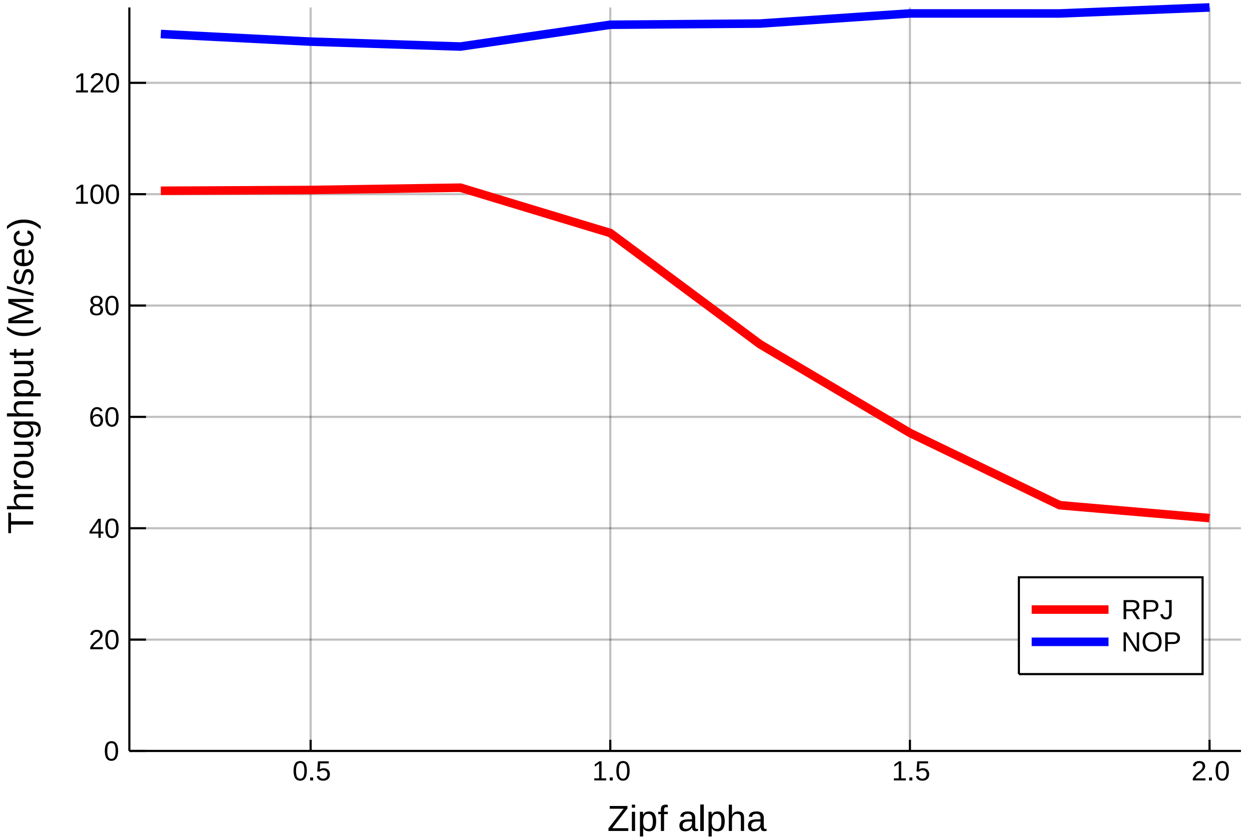
S uniformly distributed, $|R| \approx 65k; |S| \approx 33.6M$

Benchmarks - RPJ resilience



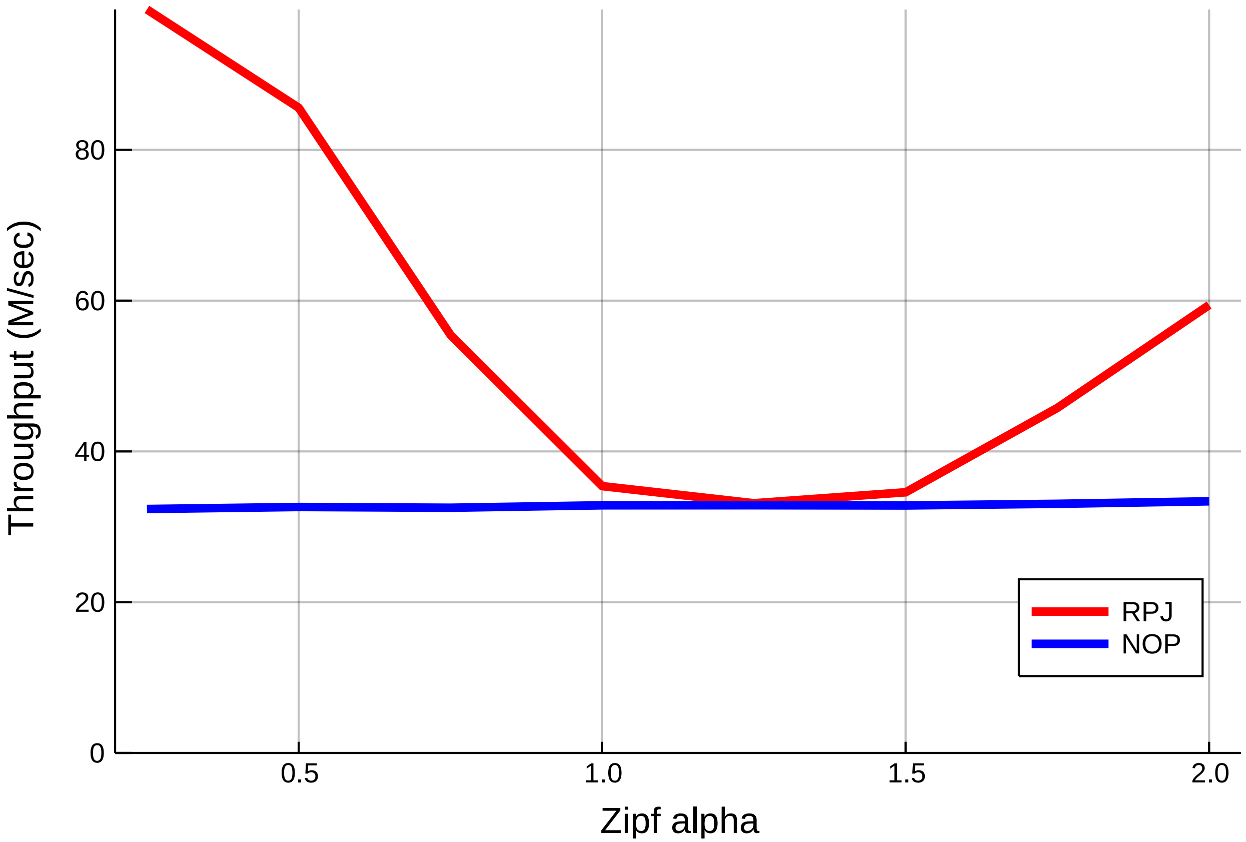
S uniformly distributed, $|R| = |S|$ ≈16.8M

Benchmarks - High Skew



S zipf distributed, $|R| \approx 65k; |S| \approx 33.6M$

Benchmarks - High Skew



S zipf distributed, $|R| = |S| \approx$16.8M

References

- Balkesen et al. “Main-Memory Hash Joins on Multi-Core CPUs: Tuning to the Underlying Hardware”. In: *ICDE* (2013).
- Kim et al. “Sort vs. Hash Revisited: Fast Join Implementation on Modern Multi-Core CPUs”. In: *VLDB* (2009).
- Patel Blanas Li. “Design and Evaluation of Main Memory Hash Join Algorithms for Multi-core CPUs”. In: *SIGMOD* (2011).
- Kersten Boncz Manegold. “Database Architecture Optimized for the new Bottleneck: Memory Access”. In: *VLDB* (1999).
- Dittrich Schuh Chen. “An Experimental Comparison of Thirteen Relational Equi-Joins in Main Memory”. In: *SIGMOD* (2016).

Questions?