Performance in the Multicore Era

Gustavo Alonso

Systems Group -- ETH Zurich, Switzerland
Performance in the multicore era
BACKGROUND

- SWISSBOX

SwissBox: An Architecture for Data Processing Appliances by Gustavo Alonso, Donald Kossmann, Timothy Roscoe: CIDR 2011: 32-37
The SwissBox project

- Build an open source data appliance
  - Hardware
  - Software
- What is a DB appliance?
  - Database in a box
  - Funny database
  - Funny box
Redesign of system software from the ground up
  – Operating system (Barrelfish)
  – Storage engines (Crescando, Ibex, ...)
  – Database engines (SharedDB, ...)
  – Run time systems (SKB for JVM, ...)

Fresh look at hardware and customized hardware
  – Database appliances (SwissBox)
  – Hardware acceleration (FPGA, many core)

Our biggest problem today: computer architecture
Today’s show

- Multicore
  - Anatomy of a join
  - Implementation(s) on multicore
  - Discussion
  - Implications
- COD (operating system – database codesign)
  - Opening up OS interfaces
  - Capturing/understanding performance
- Ultimate goal is to move database operators into hardware ...
The joy of joins

"Main-Memory Hash Joins on Multi-Core CPUs: Tuning to the Underlying Hardware" by Cagri Balkesen, Jens Teubner, Gustavo Alonso, and Tamer Ozsu, ICDE 2013
Joins are a complex and demanding operation
Lots of work on implementing all kinds of joins
In 2009
   – Kim, Sedlar et al. paper (PVLDB 2009)
      Radix join on multicore
      Sort Merge join on multicore
Claim fastest implementation to date
Key message: when SIMD wide enough, sort merge will be faster than radix join
In 2011
- Blanas, Li et al. (SIGMOD 2011)
  No partitioning join
  (vs. radix join version of Kim paper)
  Claim: Hardware is good enough, no need for careful tailoring to the underlying hardware
In 2012

- Albutiu, Kemper et al. (PVLDB 2012)
  Sort merge joins
  (vs join version of Blanas)

Claim: **Sort merge already better and without using SIMD**
The basic hash join
Canonical Hash Join

1. Build phase

\[ R \]

\[ \text{hash(key)} \]

\[ \text{hash table} \]

\[ \text{bucket 0} \]

\[ \text{bucket 1} \]

\[ \text{bucket n-1} \]

2. Probe phase

\[ S \]

\[ \text{hash(key)} \]

\[ \text{match} \]

✓ Complexity: \( O(|R| + |S|) \), i.e., \( O(N) \)

✓ Easy to parallelize

Performance in the multicore era
Need for Speed

Hardware-Conscious
Hash Joins
Partitioned Hash Join (Shatdal et al. 1994)

- **Idea**: Partition input into disjoint chunks of cache size

\[
\begin{align*}
R & \quad \text{h}_1(\text{key}) & 1 \quad h_2(k) & 1 \\
\text{h}_1(\text{key}) & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
p & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
p & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
S & \quad \text{h}_1(\text{key})
\end{align*}
\]

- **Steps**:
  1. Partition
  2. Build
  3. Probe

- **No more cache misses during the join**

**Problem**: \( p \) can be too large!

\( p > \#\text{TLB-entries} \rightarrow \text{TLB misses} \)
\( p > \#\text{Cache-entries} \rightarrow \text{Cache thrashing} \)

"Cache conscious algorithms for relational query processing", Shatdal et al, VLDB ‘94

Performance in the multicore era
Multi-Pass Radix Partitioning

- **Problem:** Hardware limits fan-out, i.e. $T = \#TLB$-entries (typically 64-512)
- **Solution:** Do the partitioning in multiple passes!

**Input relation**

1st pass $h_1$ (key)

1st $\log_2 T$ bits of hash(key)

1st pass $h_1$ (key)

2nd Pass $h_2$ (key)

2nd $\log_2 T$ bits of hash(key)

2nd Pass $h_2$ (key)

1

... $i^{th}$ pass ...

i = $\log_T p$

Partition - 1

Partition - $T^i$

**TLB & Cache** efficiency compensates multiple read/write passes


Performance in the multicore era
Parallel Radix Join

Parallelizing the Partitioning: Pass - 1

Each thread scatters out its tuples based on the prefix sum

Sort vs. Hash Revisited: Fast Join Implementation on Modern Multi-Core CPUs, VLDB ‘09

Performance in the multicore era
Parallel Radix Join

Parallelizing the Partitioning: Pass - (2 .. i)

Each thread individually partitions sub-relations from pass-1

Performance in the multicore era
Trust the force

Hardware-Oblivious Hash Joins
Parallel Hash Join („no partitioning join“ of Blanas et al.)

1. Build Phase
   1. Acquire latch
   2. Store in bucket

2. Probe Phase
   Compare & match

Performance in the multicore era
Improving on Hardware-Oblivious Hash Joins
The main claim behind no partitioning is
- Hardware is good enough
  To hide cache misses
  To hide internal details of the architecture
The code is multithreaded, uses latches, and pays no attention to NUMA

These are all very important claims = query optimizer is easier and can ignore the hardware
Cache Efficiency

Hash table in Blanas et al.

- Latch array
- Hash table pointer array
- Buckets (as linked list)

- Head*
- Free
- Next
- Tuple 1
- Tuple 2

1-Byte 8-Bytes 48-Bytes

Expect 3 cache miss / tuple

Hash table in our code

- Hash table as array of buckets

- 48-Bytes

Expect 1 cache miss / tuple

Performance in the multicore era
### SMT vs code efficiency

- SMT helps to hide cache miss latencies
- ... but for code with less redundancy, only negligible benefit

![Graph showing performance comparison between SMT and non-SMT code](image)

- no partitioning (code of Blanas et al.)
- no partitioning (our code)

Number of threads vs performance (cycles per thread): 112 cy/tpl for 1 thread, 29 cy/tpl for 8 threads.
Comparison of Hardware-Oblivious Implementations

Workload: 16M ≈ 256M, 16-byte tuples; Machine: Intel Xeon L5520, 2.26GHz, 4-cores, 8-threads

Performance in the multicore era
Improving on Hardware Conscious Hash joins
Radix join is fairly robust against a parameter misconfiguration.

Trade-off between partitioning and join costs.

Workload: 977MiB $\bowtie$ 977MiB, 8-byte tuples; Intel Nehalem: 2-passes, AMD Bulldozer: 1-pass.
SIMD in Hash Join?

- Classical bucket chained hash table [Manegold et al., TKDE 2002]
- Relation re-ordering and histogram-based [Kim et al., VLDB 2009]

- Contiguous tuples in \( R' \) can be compared with SIMD
- Software prefetching for potential matches
Does SIMD Matter for Hash Join?

- Classic bucket chaining has an edge over other optimizations
- SIMD improves hash joins

Workload: 977MiB × 977MiB, 8-byte tuples; Intel Nehalem, 8 threads, 2-passes

Performance in the multicore era
Order of Build and Probe in Radix Join

- Order of build & probe is important for cache efficiency!

![Diagram showing the order of build and probe in a radix join operation.]

**Wrong order:** build all then probe  
⚠️ Hash tables will be evicted from cache!

**Correct order:** build and immediately probe → no more cache misses! ✔

Performance in the multicore era
Comparison of Hardware-Conscious Implementations

- **Difference code efficiency and optimal configurations**
- **Order of building and probing hash tables**

Workload: 16M \(\bowtie\) 256M, 16-byte tuples; Machine: Intel Xeon L5520, 2.26GHz, 4-cores, 8-threads

Performance in the multicore era
And the winner is ...
Effect of Workloads – Case 1

- **Workload A**: 16M ⋈ 256M, 16-byte tuples, i.e., 256MiB ⋈ 4096MiB

- Hardware –oblivious vs. –conscious with our optimized code

- ≈ -30% may seem in favor of hardware-oblivious, but ...

- Effective on-chip threading
- Efficient sync. primitives (ldstub)
- Larger page size

Performance in the multicore era
Effect of Workloads – Case 2

- **Workload B**: Equal-sized tables, 977MiB \(\propto\) 977MiB, 8-byte tuples

- Picture radically changes: Hardware-conscious is better by **3.5X** on Intel and **2.5X** on others

- With larger build table, overhead of not being hardware-conscious is clearly visible
Scalability

- **Workload B**: Equal-sized tables, 977MiB $\bowtie$ 977MiB, 8-byte tuples

```
radix  n-part
```

- Intel Sandy Bridge 2.7GHz, 8 cores/16 threads
- Fastest reported join performance to date!
Hardware-Consious or Not?

✗ Hardware-oblivious algorithms work well only under a narrow parameter window and on particular hardware
  • Large pages
  • Software prefetching
  • Small build table

✔ Hardware-conscious algorithms
  – are significantly **faster** under a wider range of setup
  – can be **tuned** to the hardware
  – and are **robust** to wider set of parameters
What does it mean?

- Underlying hardware affects performance in many ways
  - Difficult for the optimizer
  - Difficult for the database administrator
  - Frustrating for users

- Algorithm performance determined by many factors
  - Who knows about the hardware?
  - Who should make the decision?

- Some (few) people can hack anything, the question is whether the effort can be generalized and replicated.
Operating System
Database
Co-design

COD: Database / Operating System Co-Design by Jana Giceva, Tudor-Ioan Salomie, Adrian Schupbach, Gustavo Alonso, Timothy Roscoe. CIDR 2013
## Example: deployment on multicores

<table>
<thead>
<tr>
<th></th>
<th>Min Cores</th>
<th>Partition Size [GB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Nehalem</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>AMD Barcelona</td>
<td>5</td>
<td>1.6</td>
</tr>
<tr>
<td>AMD Shanghai</td>
<td>3</td>
<td>2.6</td>
</tr>
<tr>
<td>AMD MagnyCours</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

**Experiment setup**
- 8GB datastore size
- SLA latency requirement 8s
- 4 different machines

Performance in the multicore era
What is the knowledge we have?

Who knows what?

COD: Database/Operating System co-design

Performance in the multicore era
Cod’s Interface

**Policy Engine**

Push application-specific facts:
- #Requests (in a batch)
- Datastore size (#Tuples, and TupleSize)
- SLA response time requirement

Needed for:
- cost / utility functions:
  \[ RT[ms] = c \times \frac{\#tuples}{\#cores} \times (a \times \#queries + b) \]

**DB storage engine**

**Application-specific**
- DB-specific Facts & properties
- Cost functions

Performance in the multicore era
This is the end ...  
(or maybe just the beginning)
Are we ready for many core?

"normal" CPU

Performance in the multicore era
Intelligent storage engine

Performance in the multicore era

- Laptop
- Electricity meter
- Power supply
- MySQL (Ibex)
- FPGA board
- SSD
- SATA II
- SIRC over Ethernet
Conclusions

- Hardware for databases ...
  - Each machine is a different world (getting worse)
  - Many core changes the picture completely
  - Heterogeneous cores and interconnects
  - Performance purely dictated by hardware (?)

- We need
  - Benchmarks
  - Deeper analysis
  - More people working in the area
  - Deeper interaction with hardware designers