CS 764: Topics in Database Management Systems
Lecture 3: Radix Join

Xiangyao Yu
9/15/2021
Today’s Paper: Radix Join

Database Architecture Optimized for the new Bottleneck: Memory Access

Peter Boncz*  
Data Distilleries B.V.  
Amsterdam - The Netherlands  
P.Boncz@difi.nl

Stefan Manegold  
Martin Kersten  
CWI  
Amsterdam - The Netherlands  
{S.Manegold, M.Kersten}@cwi.nl

1 Introduction

Custom hardware -- from workstations to PCs -- has been experiencing tremendous improvements in the past decades. Unfortunately, this growth has not been equally distributed over all aspects of hardware performance and capacity. Figure 1 shows that the speed of commercial microprocessors has been increasing roughly 10% every year, while the speed of commodity DRAM has improved by little more than 5% over the past decade [Moo98]. Part of the reason for this is that there is a direct tradeoff between capacity and speed in DRAM chips, and the highest priority has been for increasing capacity. The result is that from the perspective of the processor, memory has been getting slower at a dramatic rate. This affects all computer systems, making it increasingly difficult to achieve high processor efficiencies.

Three aspects of memory performance are of interest: bandwidth, latency, and address translation. The only way to reduce effective memory latency for applications is to have fast, fast memory that is not subject to blocking. In this paper, we focus on the design and implementation of a fast, fast memory, that is, a fast cache with low latency.

*This work was carried out when the author was at the University of Amsterdam, supported by EURYIP grant 022-35-91. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a formal request to the Endowment.


Figure 1: Hardware trends in DRAM and CPU speed
Agenda

Hardware background
In-memory partitioned hash join
Radix join
Experimental results
Column-store and encoding
Radix join vs. non-partition hash join
Memory Wall

The growth of memory speed is slower than the growth of CPU speed:
- Latency
- Bandwidth

Figure 1: Hardware trends in DRAM and CPU speed
Memory/Cache Hierarchy

Higher bandwidth
Lower access latency
Smaller capacity

Figure 2: Hierarchical Memory System
Memory/Cache Hierarchy

Optimizing join in DRAM/Disk system
   – GRACE hash join

Optimizing join in SRAM/DRAM system?

Higher bandwidth
Lower access latency
Smaller capacity
Optimizing Join in Main-Memory DBMS

Intuitive solution: Partition tables into shards that fit in SRAM cache

Figure 2: Hierarchical Memory System
Optimizing Join in Main-Memory DBMS

Intuitive solution: Partition tables into shards that fit in SRAM cache

Challenges:
- TLB becomes a performance bottleneck if too many partitions exist
- Determine the memory layout of data partitions (e.g., fragmentation)

Figure 2: Hierarchical Memory System
Translation Lookaside Buffer (TLB)

A cache of OS page table to accelerate virtual address to physical address translation

- TLB hit has no cost
- TLB miss requires an expensive page table walk

TLB has a small number of entries

If the number of partitions is greater than the number of TLB entries, the system experience **TLB trashing**, i.e., constant TLB misses.
Thrashing

**TLB thrashing**: Number of accessed pages (i.e., number of partitions) is greater than the number of TLB entries in hardware.

**Cache thrashing**: Number of accessed cachelines (i.e., number of partitions) is greater than the cache capacity.

**Page thrashing** (in last lecture): Number of accessed pages (i.e., number of partitions) is greater than the memory capacity.
Intuitive solution: Partition tables into shards that fit in SRAM cache

Challenges:

- TLB becomes a performance bottleneck if too many partitions exist
- Determine the memory layout of data partitions (e.g., fragmentation)

Do not have too many partitions per round of partitioning. Limiting factor includes cache size and TLB size.
How to track **location and size** for different partitions?

- Frequent memory allocation (e.g., malloc) is expensive
- Loss of memory capacity due to fragmentation
Cluster on the **lower B bits of the integer hash-value** of the partition key

– For pass $p$, use $B_p$ bits for partitioning
Radix Partitioning

Cluster on the lower $B$ bits of the integer hash-value of the partition key
- For pass $p$, use $B_p$ bits for partitioning

The output array of Radix partitioning has identical structure as the input array
- No complex memory allocation
- No fragmentation
Radix Partitioning

Cluster on the **lower B bits of the integer hash-value** of the partition key
- For pass $p$, use $B_p$ bits for partitioning

The output array of Radix partitioning has identical structure as the input array
- No complex memory allocation
- No fragmentation

Q: How to know where to write in the output array? (e.g., 47 in the example)
- Need to scan the array twice; first time collect size per partition
Radix Partitioning

Fully clustering $B$ bits may require multiple passes

Number of partitions per pass is bounded by TLB and cache size
Join
Similar to GRACE hash join, join the corresponding partitions from the two relations
Can use either hash join or nested-loop join
Evaluation: Radix Clustering

The machine’s TLB has 64 entries
Evaluation: Join Performance

Nested-loop join prefers small partitions

Hash-join achieves similar performance for a range of partition sizes
Evaluation: Overall

Sort-merge < Simple hash < phash L2 < phash TLB and the rest
A Different View Point

12 years later
Multicore processors

Two design considerations
– minimizing the number of processor cache misses => **Radix Join**
– minimizing processor synchronization costs => **No partition hash join**
Evaluation on Multicore

Uniform dataset

Highly skewed dataset

Important to minimize synchronization overhead in multicore processors
Other Topics

Column-store for analytical databases

Other Topics

Column-store for analytical databases

Dictionary encoding
- Many other encoding/compression schemes exist. E.g., bit-packing, delta encoding, RLE, etc.
Radix Join – Comments and Q/A

• Radix join vs. Radix sort
• Virtual OID is confusing
• Radix join widely used in current database design?
• Disadvantage of radix join?
• Why is vertical decompose superior?
Group Discussion

We want to join three tables, \( S \bowtie R \bowtie T \). Assume \( S \) is large but \( R \) and \( T \) are relatively small. Assume the two joins are on different join keys. Would you use non-partitioned hash join or radix join for this query? Please justify your choice.
Before Next Lecture

Submit review for