class 16

hash joins

prof. Stratos Idreos

HTTP://DASLAB.SEAS.HARVARD.EDU/CLASSES/CS165/
give me all students enrolled in cs165

```
select student.name from students, enrolled, courses
where courses.name="cs165" and enrolled.courseld=course.id and
student.id=enrolled.studentId
```
key,payload

1,d1,e1,f1
1,d2,e2,f2
2,d3,e3,f3
2,d4,e4,f4
2,d5,e5,f5
3,d6,e6,f6

join

key,payload

1,a1,b1,c1
2,a2,b2,c2
3,a3,b3,c3
4,a4,b4,c4
1,a5,b5,c5

join result

1,d1,e1,f1,a1,b1,c1
1,d1,e1,f1,a5,b5,c5
1,d2,e2,f2,a1,b1,c1
1,d2,e2,f2,a5,b5,c5
2,d3,e3,f3,a2,b2,c2
2,d4,e4,f4,a2,b2,c2
2,d5,e5,f5,a2,b2,c2
3,d6,e6,f6,a3,b3,c3
new resL[]; new resR[]; k=0
for (i=0;i<L.size;i=i++)
    for (j=0;j<R.size;j++)
        if L[i] == R[j]
            resL[k] = i
            resR[k++] = j
oracle
search in O(1)
hash join

join input 1 \[\rightarrow\text{hash} \rightarrow\text{hash table} \rightarrow\text{hash} \rightarrow\text{join input 2}\]
\[ h = f(val) \]
\[ \text{bucket} = h \mod k \]

[(val1, pos1), (val7, pos7), ...]

N keys

k buckets

bucket size
assumption for this class: we know how many buckets we need (more in next class about this)
minimize data movement
maximize CPU utilization
goal: join $R(val,pos)$ and $S(val,pos) = Res(posR,posS)$

Level 1
5 blocks

Level 2
$R+S<<L2$
goal: join $R(val, pos)$ and $S(val, pos) = Res(posR, posS)$

Level 1
5 blocks

Level 2
$R + S << L2$
goal: join $R(\text{val}, \text{pos})$ and $S(\text{val}, \text{pos}) = Res(\text{pos}_R, \text{pos}_S)$

need one block to stream $R$

Level 1
5 blocks

Level 2
$R + S << L2$
goal: join $R(val,pos)$ and $S(val,pos) = Res(posR,posS)$

need one block to stream $R$

need one block to write $Res$

Level 1
5 blocks

Level 2
$R+S<<L2$

$R$

$S$

$Res$
goal: join $R(\text{val}, \text{pos})$ and $S(\text{val}, \text{pos}) = \text{Res}(\text{posR}, \text{posS})$

need one block
to stream $R$

need one block
to write $\text{Res}$

hash table, $S \leq L1-2$

Level 1
5 blocks

Level2
$R + S \ll L2$

Res
goal: join $R(val,pos)$ and $S(val,pos) = Res(posR,posS)$

need one block to stream $R$

need one block to write $Res$

hash table, $S \leq L1-2$

Level 1
5 blocks

Level 2
$R+S<<L2$

Res

$R$

$S$
goal: join $R(\text{val}, \text{pos})$ and $S(\text{val}, \text{pos}) = \text{Res}(\text{pos}_R, \text{pos}_S)$

need one block to stream $R$

need one block to write $\text{Res}$

hash table, $S \leq L1-2$

Level 1
5 blocks

Level 2
$R + S \ll L2$

Res
goal: join $R(val,pos)$ and $S(val,pos) = Res(posR,posS)$

need one block to stream $S$

create hash table on $S$

 unused

 Level 1

5 blocks

 Level2

R+S<<L2
goal: join $R(val,pos)$ and $S(val,pos) = \text{Res}(posR,posS)$

- need one block to stream $S$
- create hash table on $S$

**Diagram:**

- Level 1: 5 blocks
- Level 2: $R+S<<L2$

**Total Cost:** $R+S+\text{Res} \ (\text{blocks})$
what if $R > L_{1-2}$ & $S > L_{1-2}$?

new resL; new resS; k=0;r=0;
for (i=0,i<S.size;i++,k++)
    addToHash(ht,S[i], S[i].val mod L1-2)
    if (k==L1-2 || i==S.size-1)
        k=0
        for (j=0;j<R.size;j++)
            res=probe(ht,R[i].val);
            if (res!=null)
                resR[r]=R[j].[pos]
                resS[r++]=res.pos
        empty(ht);

hash first $L_{1-2}$ values from $S$
scan all $R$ and probe
repeat until we hash everything in $S$

Total Cost=$R*S/(L_{1-2})+S+Res$ (blocks)
grace hash join

join input 1                join input 2

hash partitioning

one pass to partition

then one pass to
join each pair of partitions independently in memory
for every L.key = R.key pair
return [L.pos,R.pos]
data/results
one column-at-a-time

1) design a grace join algorithm and give its cost
2) which table do we start from?
3) what if partitions end up being >L1-2?
4) when can we do grace join in 2 passes max?
5) utilize multi-cores
6) can you keep all cores at 100% all the time?
1. read input into stream buffer, hash and write to respective partition buffer
2. when input buffer is consumed, bring the next one
3. when a partition buffer is full, write to L2

we can partition into L1-1 pieces in one pass
grace hash join

join input 1

join input 2

hash partitioning

then join each pair of partitions independently in memory
as long as at least one of the pieces <= L1-2
grace hash join

join input 1

join input 2

hash partitioning

both left and right side >L1-2
grace hash join

hash partitioning

apply recursively if a partition does not fit in memory
when can we do grace join in exactly 2 passes

1) at least one side should fit in L1-2

2) simplify problem by considering one side only: if all partitions we create fit in L1-2 we are ok

3) the maximum number of partitions we can create in one pass is L1-1

4) so if R/L1-1 <= L1-2 -> R <= (L1-1)(L1-2) we will not need to repartition any pieces
minimize data movement
maximize CPU utilization
how to partition in parallel?

join

```
partition each input
for each pair of partitions
create hash table
probe hash table
```

how to create HT in parallel?

how to probe HT in parallel?
1. compute in parallel
2. control access to output buffers (latches)

(or split data/memory pieces=partitions/cores)
Stream input → hash values & buckets in parallel → cores

Stream input → p1 hash table

Unused

p1 inner

p1 outer
compute hash values & buckets in parallel

cores

hash

control

stream input

p1 hash table

result

p1 inner

p1 outer

p1 result
Translation Look aside Buffer (TLB)

we need to translate virtual memory addresses to physical memory
partition $R$ such as each hash table fits in $L3$ ($L3-2$) as much as $K=L3-1$ partitions in one pass

every time a buffer/cache line is full spill to memory we may be writing to $K$ memory areas

if $K>TLB$ then we incur TLB misses
radix join

recursively partition with maximum \texttt{outputs} \leq \texttt{TLB}

\textbf{may do more passes} but sequential access for reads and random access for writes \textless \texttt{TLB}

\textbf{no TLB misses}
joins
(project=m4)

what to do in m4?
nested loops and hash joins
cache conscious and multi-core (within reason)
Join Processing in Databases with Large Main Memories
L. Shapiro
ACM Transactions on Database Systems. 11(3), 1986

Cache Conscious Algorithms for Relational Query Processing
A. Shatdal, C. Kant and J. Naughton
Very Large Databases Conference, 1984

Database Architecture Optimized for the new Bottleneck: Memory Access
P. Boncz, S. Manegold and M. Kersten
Very Large Databases Conference, 1999

Sort vs. Hash Revisited: Fast Join Implementation on
Modern Multi-Core CPUs
Changkyu Kim, et al.
International Conference on Very Large Databases, 2009
hash joins

DATA SYSTEMS

prof. Stratos Idreos