class 14

fast scans 2.0

prof. Stratos Idreos

HTTP://DASLAB.SEAS.HARVARD.EDU/CLASSES/CS165/
fast scans

hardware, data and query based optimizations
(project=m3)

apply to all algo/data structures
shared scans

q1 → q2 → q3 → q4 → q5 → q6 → ... → qn
2) schedule queries on same data to run in parallel

3) each query gets a thread/core from thread pool
Navigating big data with high-throughput, energy-efficient data partitioning
L. Wu, R. J. Barker, M. A. Kim, K. A. Ross

Meet the walkers: Accelerating index traversals for in-memory databases
O. Koçberber, B. Grot, J. Picorel, B. Falsafi, K. T. Lim, P. Ranganathan
International Symposium on Microarchitecture, 2013

Beyond the Wall: Near-Data Processing for Databases
S. Xi, O. Babarinsa, M. Athanassoulis, S. Idreos
International Workshop on Data Management on New Hardware, 2015
more tricks

fast scans
**compression**: trading CPU for data movement
1. Can do >1 tasks at the same time.
2. Can predict data accesses.
3. Can do the same task on >1 data.
why if statements can be “bad”

pipelining
while executing current instruction, prepare for the next one

but which one is next?

predict which one is next
and start executing
pipelining
>1 instructions at a time

e.g., \( \text{res} = a + b \)
1) load \( a \)
2) load \( b \)
3) \( r = a + b \)

ideally execution is \( P \) times faster
where \( P \) is the depth of the pipeline
if (X)
  instr.A1
  instr.A2
  ...
  instr.B1
  instr.B2
  ...

... a = k * z
  r = a + j
  ...

... dependencies

CPU

fetch | decode | execute | mem | write
fetch | decode | execute | mem | write
fetch | decode | execute | mem | write
fetch | decode | execute | mem | write

... dependencies

cpu might have to stall or we might have to flush pipeline and restart
avoid dependencies and branches whenever possible

\[ \text{i instructions} \]
\[ \text{for all tuples} \]
\[ \text{do \{ ... \}} \]
\[ \text{j instructions} \]
count(A<v)
for(i=0;i<A.size;i++)
  if A[i]<v
    res++

what is the response time behavior depending on selectivity
for(i=0;i<A.size;i++)
    if A[i]<v
        res++

you would think...

because

instead
more is sometimes less…

(1) with branches
for(i=0;i<N;i++)
  if (A[i]<v)
    result++

(2) no branches
for(i=0;i<N;i++)
  result+=(A[i]<v)
(1) **using &&**  
   for(i=0;i<N;i++)  
        if (f1(a1) && f2(a2)...&& fk(ak))  
             result[j++] = i  

(2) **using &**  
   for(i=0;i<N;i++)  
        if (f1(a1) & f2(a2)...& fk(ak))  
             result[j++] = i  

(3) **no branches**  
   for(i=0;i<N;i++)  
        result[j] = i  
        j+=(f1(a1) & f2(a2)...& fk(ak))

Conjunctive Selection Conditions in Main Memory  
K. Ross. TODS 2004
(1) **with branches**
   
   ```
   for(i=0;i<N;i++)
     if (A[i]<v)
       result++
   ```

(2) **no branches**

   ```
   for(i=0;i<N;i++)
     result+=(A[i]<v)
   ```

(3) **no branches**

   ```
   for(i=0;i<N;i+=2)
     result1+=(A[i]<v)
     result2+=(A[i+1]<v)
   
   result=result1+result2
   ```
prefetching
predict future data accesses

to process data X we need to fetch X

CPU
registers
on chip cache
on board cache
memory
disk

fetch

prefetching:
while processing Z fetch X on last level cache

hardware assisted
+
software assisted

scan vs index scan
1. can do >1 tasks at the same time

2. can predict data accesses

3. can do the same task on >1 data

cpu
**SIMD** (single instruction multiple data)

apply the same action on >1 data values with the same cost as for 1 value

A1, A2, A3, A4  B1, B2, B3, B4

add/min/max, etc

typical size 128 bit
run at the same time

result: 128 bit

res1, res2, res3, res4

A1, A2, A3, A4

B1, B2, B3, B4

input 1: 128 bit

input 2: 128 bit
**result**: 128 bit

6, 13, 13, 4

**input 1**: 128 bit

5, 4, 7, 2

**input 2**: 128 bit

1, 9, 6, 2
result: 128 bit

0, 4294967295, 0, 4294967295

min  min  min  min

5, 4, 7, 2  1, 9, 6, 2

input 1: 128 bit  input2: 128 bit
how: assembly || compilers || add SIMD commands
write a sum operator using SIMD instructions

```c
for(i=0;i<data.size;i++)
    res+=data[i]
```
for(i=0;i<data.size;i++)
    if data[i]<v
        res+=data[i]

you have the following instructions:

**SIMD_add**: [A1,B1,C1,D1], [A2,B2,C2,D2]->[A1+A2,B1+B2,C1+C2,D1+D2]

**SIMD_shuffle32**: [A,B,C,D]->[B,A,D,C]

**SIMD_shuffle64**: [A,B,C,D]->[C,D,A,B]

**SIMD_and**: [1,0,1,0] & [1,1,1,1] ->[1,0,1,0]

**SIMD_lt**: [10,2,4,10] < [4,7,3,6] ->[0,1,0,0]

SIMD vector=128bit
for(i=0;i<data.size;i++)
    res+=data[i]

for (i=0;i<N;i+=4)
    res[0,1,2,3]=SIMD_add(res[0,1,2,3], data[i,i+1,i+2,i+3])
    res=a,b,c,d

    t1=SIMD_shuffle32(res)
    t1=b,a,d,c

    t2=SIMD_add(res,t1)
    t2=a+b,a+b,c+d,c+d

    t3=SIMD_shuffle64(t2)
    t3=c+d,c+d,a+b,a+b

    res=SIMD_add(t2,t3)
    res=a+b+c+d,a+b+c+d,a+b+c+d,a+b+c+d

(assuming N mod 4 = 0)
for (i=0; i<data.size; i++)
    if data[i]<v
        res+ = data[i]

for (i=0; i<N; i+=4)
    t1=SIMP_D_Lt(data[i,i+1,i+2,i+3],v[v,v,v,v])
    t2=SIMP_D_and(t1[0,1,2,3],data[i,i+1,i+2,i+3])
    res[0,1,2,3]=SIMP_D_add(res[0,1,2,3], t2[0,1,2,3])

res=[res+2,res+0,res+0,res+6]

(assuming N mod 4 = 0)
column-store storage better for SIMD
data is already packed in dense arrays
fast scans

tight for-loops, fixed width dense arrays
multi-core, SIMD, share scans

apply to all algo/data structures
Selection conditions in main memory
Kenneth A. Ross
ACM Transactions on Database Systems, 2004

Implementing database operations using SIMD instructions
by Jingren Zhou, and Kenneth A. Ross
ACM SIGMOD International Conference on Management of Data, 2002

Efficient Implementation of Sorting on Multi-Core SIMD CPU Architecture
by Jatin Chhugani et al.
International Conference on Very Large Databases (VLDB), 2008
OH today-tmr: Stratos out of town - but you can find Manos
Friday no OH
Sunday 2-6:30pm with Stratos

fast scans 2.0
DATA SYSTEMS
prof. Stratos Idreos