class 13

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
always want to minimize data movement - computation & utilize all resources!
shared scans

q1 q2 q3 q4 q5 q6 ... ... qn
1) gather queries
2) schedule queries on same data to run in parallel
3) each query gets a thread/core from thread pool

data moves once
# of cores queries run in parallel
more tricks

fast scans
**compression**: trading CPU for data movement

- registers
- on chip cache
- on board cache
- memory
- disk
1. can do >1 tasks at the same time
2. can predict data accesses
3. can do the same task on >1 data
RISC CPU architecture

we can “synthesize” all instructions out of a small set of “hardware primitives”
RISC CPU architecture

e.g., res = a + b
1) load a
2) load b
3) r = a + b

fetch instruction 1), decode, execute (find memory location), read from memory, write to register
pipelining
>1 instructions at a time
pipelining
>1 instructions at a time
**Pipelining**

>1 instructions at a time

**Parallel execution**

**CPU**

fetch → decode → execute → mem → write

fetch → decode → execute → mem → write

fetch → decode → execute → mem → write

...
Pipelining

>1 instructions at a time

Parallel execution

Ideally, execution is $P$ times faster where $P$ is the depth of the pipeline.
next instruction starts before the previous one finishes

but which one is next?
**pipelining**
while executing current instruction, start next one

but which one is next?

**predict** which one is next and start executing
**pipelining**
while executing current instruction, start next one

**predict** which one is next and start executing

but which one is next?

```
... if (X)
    instr.A1
    instr.A2
...
instr.B1
instr.B2
...  
... a=k*z
    r=a+j
... dependencies
```
pipelining
while executing current instruction, start next one

**CPU**

predict which one is next and start executing

but which one is next?

speculative execution

... if (X)
    instr.A1
    instr.A2
...
instr.B1
instr.B2
...

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

dependencies
**Pipelining**

While executing current instruction, start next one.

---

Predict which one is next and start executing.

---

Speculative execution

CPU may stall or even have to flush pipeline.

---

```
... if (X)
    instr.A1
    instr.A2
... instr.B1
    instr.B2
... a = k * z
    r = a + j
... dependencies
```
avoid dependencies and branches whenever possible

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

what is the response time behavior depending on selectivity selectivity affects how many times CPU should pick instructions from “inside or outside” the if statement
for(i=0; i<A.size; i++)
    if A[i]<v
        res++
for(i=0; i<A.size; i++)
    if A[i]<v
        res++

# of tuples that qualify: 0
N

response time

you would think...

instead

# of tuples that qualify
N
N

response time

0

# of tuples that qualify
N

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

# of tuples that qualify 0
response time

you would think...

branch mispredictions

# of tuples that qualify
0
N

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
   ```
1. can do >1 tasks at the same time

2. can predict data accesses

3. can do the same task on >1 data
prefetching
predict future data accesses

to process data X
we need to fetch X

prefixing:
while processing Z
fetch X on last level cache

hardware assisted
+
software assisted

scan vs index scan
**SIMD** (single instruction multiple data)

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

![Diagram of SIMD operation](image)
result: 128 bit

res1, res2, res3, res4

run at the same time

\[ \text{op} \quad \text{op} \quad \text{op} \quad \text{op} \]

A1, A2, A3, A4

input 1: 128 bit

B1, B2, B3, B4

input2: 128 bit
result: 128 bit

res1, res2, res3, res4

run at the same time

op  op  op  op

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

input 1: 128 bit  input2: 128 bit
run at the same time

result: 128 bit

res1, res2, res3, res4

A1, A2, A3, A4
input 1: 128 bit

B1, B2, B3, B4
input 2: 128 bit
run at the same time

\[ \text{result: } 128 \text{ bit} \]

\[ \text{res1, res2, res3, res4} \]

\[ \text{op} \quad \text{op} \quad \text{op} \quad \text{op} \]

\[ \text{A1, A2, A3, A4} \quad \text{B1, B2, B3, B4} \]

\[ \text{input 1: } 128 \text{ bit} \quad \text{input 2: } 128 \text{ bit} \]
result: 128 bit

run at the same time

input 1: 128 bit

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

input 1: 128 bit

5, 4, 7, 2

input 2: 128 bit

1, 9, 6, 2

add  

6, 13, 13, 4

add

add

add
result: 128 bit

0, 4294967295, 0, 4294967295

lt
lt
lt
lt

5, 4, 7, 2

input 1: 128 bit

1, 9, 6, 2

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

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 = \texttt{SIMD\_Lt}(data[i, i+1, i+2, i+3], v[v, v, v, v])
    data = [2, 13, 15, 6] \mid v = [10, 10, 10, 10]
    t2 = \texttt{SIMD\_and}(t1[0, 1, 2, 3], data[i, i+1, i+2, i+3])
    t1 = [1, 0, 0, 1] \mid t2 = [2, 0, 0, 6]
    res[0, 1, 2, 3] = \texttt{SIMD\_add}(res[0, 1, 2, 3], t2[0, 1, 2, 3])

res = [\text{res}+2, \text{res}+0, \text{res}+0, \text{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
fast scans 2.0
DATA SYSTEMS
prof. Stratos Idreos

HARVARD
School of Engineering and Applied Sciences