DATABASE CRACKING:
Fancy Scan, not Poor Man’s Sort!

Hardware Folks
Holger Pirk
Stefan Manegold

Cracking Folks
Eleni Petraki
Strato Idreos
Martin Kersten

Don
EVALUATING RANGE PREDICATES
COMPLEXITY ON PAPER

• Scanning: $O(n)$
• Sorting: $O(n \times \log(n))$
• Cracking: $O(n)$
  • Essentially a single Quicksort-Step
COSTS IN REALITY

• Implement microbenchmarks
  • 1 Billion uniform random integer values
  • Pivot in the middle of the range
  • Workstation machine (16 GB RAM, 4 Sandy Bridge Cores)
SO: WHAT’S GOING ON?
CACHE MISSES?

- L1I Misses
- L2 Misses
- L1D Misses
- L3 Misses

Scanning: 0.0
Cracking: 0.0
Sorting: 0.0

1.5B
1.4B
1.2B
1.0B
0.8B
0.6B
0.4B
0.2B
0.0

NOPE!
CPU COSTS

Micro-ops Issued?

No

Allocation Stall?

No

Frontend Bound

Yes

Backend Bound

Micro-op Ever Retire?

No

Bad Speculation

Yes

Retiring

Cache Miss Stalls

Other Stalls
CPU COSTS

- **Data Stalls**
- **Bad Speculation**
- **Retiring**
- **Pipeline Frontend**
- **Pipeline Backend**

14% !!!
CPU COSTS

- Scanning
- Cracking
- Sorting

- Data Stalls
- Bad Speculation
- Retiring
- Pipeline Frontend
- Pipeline Backend

Lots of Potential
WHAT CAN WE DO ABOUT IT?
INCREASING CPU EFFICIENCY
PREDICATION

for(i=0; i<size; i++)

if(input[i] < pivot) {
    output[outI] = input[i];
    outI++
}

for(i=0; i<size; i++)

    output[outI] = input[i];

    outI += (input[i] < pivot);
PREDICATION

- Turns control dependencies into data dependencies
- Eliminates Branch Mispredictions
- Causes unconditional (potentially unnecessary) I/O
  - (limited to caches)
- Works only for out-of-place algorithms
PREDICATED CRACKING
PREDICATED CRACKING

pivot 5

active

backup

7 2 4 8 2 9 3 8 1 5 0 7 5 3
PREDICATED CRACKING

pivot 5

active 3

backup 7

3 2 4 8 2 9 3 8 1 5 0 7 5 7
PREDICATED CRACKING

polar
5

backup
7

State Before Iteration

3 2 4 8 2 9 3 8 1 5 0 7 5 7
PREDICATED CRACKING

Evaluate Predicat & Write
PREDICATED CRACKING

pivot 5  
cmp 1  
active 3  
backup 7  

+=  
1-  
-=  

3 2 4 8 2 9 3 8 1 5 0 7 5 3  

Advance Cursor
PREDICATED CRACKING

pivot 5
cmp 1
active 2
backup 7

3 2 4 8 2 9 3 8 1 5 0 7 5 3

Read Next Element
PREDICATED CRACKING

pivot 5
cmp 1
backup 2
active 7

3 2 4 8 2 9 3 8 1 5 0 7 5 3
PREDICATED CRACKING

- Predication for in-place algorithms
- No branching $\Rightarrow$ No branch mispredictions
- Somewhat intricate
- Lots of copying stuff around (integer granularity $\Rightarrow$ inefficient)
- Bulk-copying would be more efficient
VECTORIZED CRACKING
VECTORIZED CRACKING

- Turns in-place cracking into out-of-place cracking
- Copies Vector-sized chunks and cracks them into the array
  - Makes vanilla-predication possible
- Uses SIMD-copying for vector copying
- Challenge: ensure that values aren't "accidentally" overwritten
VECTORIZED CRACKING
RESULTS
RESULTS: WORKSTATION

- Wallclock time in s

- RESULTS:
  - Scan
  - Vectorized
  - Predicated (Register)
  - Predicated (Cache)
  - Original
RESULTS: SERVER

Wallclock time in s

- Scan
- Vectorized Predicated (Register)
- Predicated (Cache)
- Original

Not there yet!
PARALLELIZATION
PARALLELIZATION

• Obvious Solution: Partitioning
CRACK & MERGE

Partition
CRACK & MERGE

Merge
REFINED CRACK & MERGE
REFINED CRACK & MERGE

Smaller Merge
RESULTS: WORKSTATION

Seconds

- Scan
- RVPCrack
- RPCrack
- PVCrack
- PCrack
- Vectorized
RESULTS: SERVER

<table>
<thead>
<tr>
<th>Method</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan</td>
<td>0.00</td>
</tr>
<tr>
<td>RVPCrack</td>
<td>0.75</td>
</tr>
<tr>
<td>RPCrack</td>
<td>1.50</td>
</tr>
<tr>
<td>PVCrack</td>
<td>2.25</td>
</tr>
<tr>
<td>PCrack</td>
<td>3.00</td>
</tr>
<tr>
<td>Vectorized</td>
<td>3.00</td>
</tr>
</tbody>
</table>
IMPACT OF SELECTIVITY: WORKSTATION

Wallclock time in s

Qualifying Tuples/Pivot

- Vectorized
- Partition & Merge
- Vectorized Partition & Merge
- Refined Partition & Merge
- Vectorized Refined Partition & Merge
- Scanning
IMPACT OF SELECTIVITY: SERVER

Wallclock time in s

Vectorized
Partition & Merge
Vectorized Partition & Merge
Refined Partition & Merge
Vectorized Refined Partition & Merge
Scanning

Qualifying Tuples/Pivot
CONCLUSIONS