HIGH THROUGHPUT HEAVY HITTER AGGREGATION FOR SIMD PROCESSORS

Orestis Polychroniou
Kenneth A. Ross

Columbia University
AN OVERVIEW

• In a Glimpse
• Our Motivation
• Problem Definition
• Algorithmic Design
• Implementation & SIMD
• Experimental Results
• Closing Remarks
Manager: “I want a **plot** of our sales per product.”
Employee: “**All** products?”
Manager: “Yes **all** products.”
Employee: “But **most** of our income comes from X, Y, Z products.”
Manager: “Well tell me about the **top** products then.”
Employee: “**Ok wait...**”

.......... 
Manager: “**Not ready yet ?**”
Employee: “**Well the system does most work** for **all** products.”
Manager: “**Is this necessary ?**”
Employee: “**Well maybe... Could it do better ?**”
IN A GLIMPSE

- What do you do?
  - Best effort aggregation for heavy hitters

- What is so special about it?
  - We do it only for heavy hitters and it is fast

- Why do you do it?
  - People see and use top results
  - Hardware is faster on smaller working sets
HUMAN MOTIVATION

- Analytics & Business Intelligence
  - **Big** data are available everywhere
  - Results used for **human** decisions

- Common Properties
  - Very **large** input handled by machine
  - Small output handled by **humans**

- Observation #1

  *No matter how big the data, a small part of the output will be considered by the human factor (analysts, ...).*
SOFTWARE MOTIVATION

- Common Analytics
  - Select – Project – Join: **large** intermediate results
  - **Aggregate** – Sort (Rank): use top results

- Aggregation Step
  - May **produce** few results / groups
  - If not, **top** results will be **seen** anyway

- Observation #2

  The DBMS will aggregate before returning any results. It will **work** for **1,000,000,000** groups, even if you **use** **100** groups.
HARDWARE MOTIVATION

- Caches are fast
  - Faster than RAM by 1-2 orders of magnitude
  - Can still fit thousands of groups
  - Private caches allow shared nothing parallelism

- Caches levels have variable speeds
  - L1 is 2-4 cycles, L3 is 25-40 cycles
  - Tradeoff between speed & capacity

- Observation #3

  The smaller the working (result) set, the faster the scan/probe phase.
  But there are many tradeoffs.
Skewed data are common

- Zipf distribution is important
- Skewed distributions for synthetic data
- Strategic real data exhibit skew
- Importance of items by rank (frequency)

Sampling can estimate result

- Top-K items will be in a sample
- A verification step is required
- Avoid going over the data multiple times
**Problem Definition**

- Heavy hitter groups
  - Aggregate top $K$ groups by tuple cardinality
  - Defined by higher input frequency
  - Hopefully important groups for analysis

- Example query

```sql
select product_id, count(*)
from sales
group by product_id
order by count(*) desc
limit 1000;
```
Our Solution

- Identify possible heavy hitters
  - Sample input randomly
  - Extract heavy hitter candidates
  - Configure & build a hash table

- Scan over input data & probe
  - Update candidates found in the table
  - Increment non-candidate counts

- Verify heavy hitter groups
  - Max non-candidate is threshold
  - Like an 1D count sketch
VERIFICATION VISUALIZED

- Candidate Aggregates
- Non-Candidate Counts
- Verified Aggregates

counts  sum(X)  max(Y)
Tradeoff Aspects

- Candidate aggregates
  - Store the whole incomplete aggregate
  - If smaller then higher in cache & faster
  - If larger more candidates & more accurate

- Non-candidate counts
  - Store only a count
  - Less counts make it faster
  - More counts more accurate

- Goal
  - Choose fastest configuration
  - Accurate enough to verify top K
WHERE AND HOW TO USE

- Conventional Aggregation
  - Small group-by cardinality

- Optimization Step
  - Loop over configurations
  - Estimate configurations using sample
  - Choose best configuration
  - Early failure detection

- Verified < K
  - On failure roll back to conventional
  - Fast enough to retry other configuration
FAILURE CASES

- Correct top K are not among the candidates
  - Sample size was small and inaccurate
  - Cannot distinguish top groups by sampling

- Cannot verify K candidates
  - Not enough non-candidate counts
  - High verification threshold

- Wrong table configuration
  - Not enough candidate aggregates
  - Not enough non-candidate counts
# Hash Table

- **Multiplicative hashing**
  - **Fast** computation
  - **Random** multiplier

- **Perfect hashing**
  - No branching and branch **mispredictions**
  - Fast reply for "is key \( x \) in the table?"
  - Birthday paradox explains **small** load factor

- **Bucketized hashing**
  - Load factor of perfect hashing **increases**
  - Fast branch free probe through **SIMD**
Cuckoo hashing
- Two choice probe without branching
- 2X perfect hashing with larger load factor
- Combine with bucketized hashing

Hash configurations
- Cuckoo or perfect?
- Bucket size?
- Cache level?
- # of non-candidate counts?
- More choices more tradeoffs
HASH TABLE UPDATES

- Branch free update
  - Updates **nullified** if keys do not match
  - Non candidate counts updated **offline**

- Why SIMD?
  - Scalar code uses slower control flags
  - Transform to data **dependencies**

- SIMD where?
  - To **batch** compare keys
  - To **update** & **nullify** faster
- Perfect hashing
- 2-wide bucket

```sql
select count(*), sum(value)
from table
group by key
order by count(*) desc
limit ...
```
- Perfect hashing
- 4-wide bucket

```
select count(*), sum(X),
   max(X), min(X), sum(X*X)
from table group by key
order by count(*) desc ...
```
CONVENTIONAL AGGREGATION

- Single pass
  - **Large** hash table for aggregates with random hits on RAM
  - **PLAT** method used for cross-core cache invalidations due to heavy hitters [Ye, DaMoN11]

- Multiple passes
  - Bound by RAM throughput
  - Hash tables on cache
PERFORMANCE TRADEOFF

- 2 CPUs @ 2.4 GHz
- Intel Nehalem Appeared 2008
- 4 physical cores / CPU
- 2 hardware threads / core

![Graph showing performance tradeoff with throughput (billion records per sec) vs. capacity (number of keys). The graph includes markers for different configurations: X SMT on, + SMT off, with labels 1-wide Regular, 2-wide Regular, 4-wide Regular, 1-wide Cuckoo, 2-wide Cuckoo, 4-wide Cuckoo.]
QUALITY TRADEOFF

- 32 KB L1 cache private / core
- 256 KB L2 cache private / core
- Version SIMD SSE 4.2
Combined Results

Graphs showing throughput (billion records per sec) vs. capacity (# keys) for different configurations of L1/L2.

- Count vs. L1/L2 capacity for different SMT on/off and key widths (Regular, Cuckoo).
- Count, sum(value) vs. L1/L2 capacity for different SMT on/off and key widths.
- Count, min(value), max(value), sum(value), sum(value*value) vs. L1/L2 capacity for different SMT on/off and key widths.

Bottom graphs display correct keys vs. theta for L1-resident and L2-resident hybrid tables (L1 split 1:1, L2 split 1:1, L2 split 1:3).
REALISTIC EXPERIMENT

- Wikipedia
  - Hourly Wikipedia visits for January 2012
  - Group by URL & get average visit hour

- Skew
  - 3,463,321,585 visits
  - 102,216,378 distinct URLs
  - Top-3 URLs are 1.6% of total
  - Top-100 URLs are 6.65% of total
  - Top-10,000 URLs are 25.3% of total
# WIKIPEDIA DATASET

<table>
<thead>
<tr>
<th>Candidates</th>
<th>Non-Cand.</th>
<th>Scheme</th>
<th>1-wide</th>
<th>2-wide</th>
<th>4-wide</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Time</td>
<td>HH.</td>
<td>Freq.</td>
</tr>
<tr>
<td>L1 ×1/2</td>
<td>L1 ×1/2</td>
<td>Regular Cuckoo</td>
<td>2.32</td>
<td>9</td>
<td>3.62</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.41</td>
<td>12</td>
<td>3.62</td>
</tr>
<tr>
<td>L1 ×1/4</td>
<td>L1 ×3/4</td>
<td>Regular Cuckoo</td>
<td>2.15</td>
<td>14</td>
<td>3.39</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.47</td>
<td>15</td>
<td>2.78</td>
</tr>
<tr>
<td>L2 ×1/2</td>
<td>L2 ×1/2</td>
<td>Regular Cuckoo</td>
<td>3.59</td>
<td>92</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.49</td>
<td>217</td>
<td>0.63</td>
</tr>
<tr>
<td>L2 ×1/4</td>
<td>L2 ×3/4</td>
<td>Regular Cuckoo</td>
<td>2.77</td>
<td>103</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.92</td>
<td>215</td>
<td>0.62</td>
</tr>
<tr>
<td>L1</td>
<td>L2 ×3/4</td>
<td>Regular Cuckoo</td>
<td>2.59</td>
<td>84</td>
<td>0.89</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.74</td>
<td>162</td>
<td>0.73</td>
</tr>
</tbody>
</table>

- **# verified top groups**
- **min (Kth item) frequency (x 10^-4)**
- **execution time (seconds)**

```sql
select count(*) as visits,
       avg(hour) as mean_visit_hour
from wikipedia
group by URL
order by count(*) desc;
```
Final Remarks

- Usefulness
  - Applied on specific queries
  - Requires skew in data
  - Best effort approach
  - Useful for data exploration

- Quality
  - 5-20x faster than conventional aggregation
  - Get top 250 results out of > 25 GB in time < 5 sec
  - Smallest forms 0.006 % of total
Any questions