Navigating Big Data with High-Throughput, Energy-Efficient Data Partitioning
Lisa Wu, R.J. Barker, Martha Kim, and Ken Ross
Columbia University
Outline

1. Introduction
2. Background & Motivation
3. HARP System
4. Evaluation
5. Conclusion
6. Discussion
**Data Partitioning**

<table>
<thead>
<tr>
<th>date</th>
<th>qty</th>
<th>sku</th>
</tr>
</thead>
<tbody>
<tr>
<td>5/6/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2/2/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7/27/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/1/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/10/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9/3/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5/20/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12/6/11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Input Table**

**Splitters**

<table>
<thead>
<tr>
<th>date</th>
<th>qty</th>
<th>sku</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/2/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3/1/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7/1/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/1/11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Partitioned Data**

<table>
<thead>
<tr>
<th>date</th>
<th>qty</th>
<th>sku</th>
</tr>
</thead>
<tbody>
<tr>
<td>5/6/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/1/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5/20/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7/27/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9/3/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/10/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12/6/11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Current Situation

Data growing very fast. Almost $10^{18}$ bytes per day, 2013. (even more today!)

Databases support *partition* to faster subsequent processing. Oracle, DB2, SQL Server

Partitioning enables partitions to be processed independently and more efficiently.
- parallel
- cache locality
Software Based Partitioning

- **Hash**
  - A multiplicative hash of each record’s key determines partition.

- **Direct**
  - Like hash partitioning, but eliminates hashing cost by treating the key itself as the hash value. eg: city name

- **Range**
  - Equality range partitioning. eg: date (software based range partitioning is used as comparison in this paper)

Software implementations of data partitioning have fundamental performance limitations that make it compute-bound, even after parallelization.
Low Memory Utilization

1. Modern databases are not typically disk I/O bound.
2. Some dbs having a memory-resident working set. (Facebook 28 TB data)
3. Servers running Bing, Hotmail and Cosmos show 67%-97% processor utilization but only 2%-6% memory bandwidth utilization.
4. Google’s BigTable and Content Analyzer use a couple of percent of memory bandwidth. <10%
Solution

Low memory bandwidth utilization

Compute-bound

New hardware to accelerate

1: Hardware Accelerated Range Partitioner, or HARP

2: A high-bandwidth hardware-software streaming framework that transfers data to and from HARP
Join

Figure from Lisa Wu slide
Join

Scan | SALES
Random Lookup

WEATHER

Thrash the Cache

Figure from Lisa Wu slide
Partitioning

DB: MonetDB-11.11.5
Time: Median of ten runs of each query

Half of join time used on partitioning
Partitioning

Figure from Lisa Wu slide
Partitioning

Figure from Lisa Wu slide
Outline

1. Introduction
2. Background & Motivation
3. HARP System
4. Evaluation
5. Conclusion
6. Discussion
Hardware Accelerated Range Partitioning System

- Overview of the HARP system
- Design of the HARP Accelerator
- Design of the streaming framework
Hardware Accelerated Range Partitioning System

- Overview of the HARP system
  - Design of the HARP Accelerator
  - Design of the streaming framework
HARP System Overview

New components coupled with CPU core:

- HARP Accelerator
- Stream Buffer (SB_in, SB_out)
Hardware Accelerated Range Partitioning System

- Overview of the HARP system
- Design of the HARP Accelerator
- Design of the streaming framework
HARP Accelerator

Linear Pipelining Design

From $SB_{in}$

To $SB_{out}$
HARP Accelerator ISA

HARP Instructions:
set_splitter <splitter number> <value>
partition_start
partition_stop
HARP Implementation Details

HARP Instructions:
- set_splitter
- partition_start
- partition_stop

From $SB_{in}$

To $SB_{out}$
Set Splitters

HARP Instructions:
- `set_splitter <splitter number> <value>`
- `partition_start`
- `partition_stop`
Set Splitters

HARP Instructions:
- set_splitter <splitter number> <value>
- partition_start
- partition_stop

From CPU

From $SB_{in}$

To $SB_{out}$
Pull Data from Stream Buffer

HARP Instructions:
- set_splitter
- partition_start
- partition_stop

From $SB_{in}$

To $SB_{out}$
Pull Data from Stream Buffer

HARP Instructions:
set_splitter
partition_start
partition_stop

Convert burst to stream of records (FSM)
Pull Data from Stream Buffer

HARP Instructions:
set_splitter
partition_start
partition_stop
Pull Data from Stream Buffer

HARP Instructions:
- set_splitter
- partition_start
- partition_stop
Compare on Conveyor: Pipelining

HARP Instructions:
- set_splitter
- partition_start
- partition_stop
Compare on Conveyor: Pipelining

HARP Instructions:
- set_splitter
- partition_start
- partition_stop

From $SB_{in}$

To $SB_{out}$
Compare on Conveyor: Pipelining

HARP Instructions:
set_splitter
partition_start
partition_stop
Merge Data to Output Stream Buffer

HARP Instructions:
set_splitter
partition_start
partition_stop

From $SB_{in}$

$To SB_{out}$
Merge Data to Output Stream Buffer

HARP Instructions:
set_splitter
partition_start
partition_stop
Merge Data to Output Stream Buffer

HARP Instructions:
- set_splitter
- partition_start
- partition_stop

Pull burst of records from the full partition buffer (FSM)

(One record per cycle)
Signal to Stop & Drain In-flight Data

HARP Instructions:
set_splitter
partition_start
partition_stop
Signal to Stop & Drain In-flight Data

HARP Instructions:
set_splitter
partition_start
partition_stop
Technical Insights of the HARP Accelerator

- Pipeline is always full
- One record per cycle
- Comparison with pipeline eliminates control instructions
- Reduces instruction fetch overhead
Hardware Accelerated Range Partitioning System

- Overview of the HARP system
- Design of the HARP Accelerator
- Design of the streaming framework
  (Datapaths between HARP accelerator and memory)
Stream Buffer

- Moves data between memory and the HARP accelerator
- At or above the partitioning rate of HARP
- Minimal modifications to existing processor architecture
Stream Buffer ISA

Stream Buffer Instructions:

sbload sbid, [mem addr]
sbstore [mem addr], sbid
sbsave sbid
sbrestore sbid
Load Data from Memory to Stream Buffer

Steps:

sbload sbid, [mem addr]
Load Data from Memory to Stream Buffer

Steps:

1. Issue \texttt{sbload} from core

\texttt{sbload sbid, [mem addr]}
Load Data from Memory to Stream Buffer

Steps:

1. Issue `sbload` from core
2. Send `sbload` from request buffer to memory
Load Data from Memory to Stream Buffer

Steps:

1. Issue `sbload` from core
2. Send `sbload` from request buffer to memory
3. Data return from memory to SB_in

```
sbload sbid, [mem addr]
```
Load Data from Memory to Stream Buffer

Steps:

1. Issue \texttt{sbload} from core
2. Send \texttt{sbload} from request buffer to memory
3. Data return from memory to \texttt{SB}_in

Use \textit{existing} microarchitectural vector load request path (request buffer)
Store Data to Memory

```
sbstore [mem addr], sbid
```

Steps:
Store Data to Memory

\texttt{sbstore \{mem addr\}, sbid}

Steps:
1. Issue \texttt{sbstore} from core
Store Data to Memory

`sbstore [mem addr], sbid`

Steps:
1. Issue `sbstore` from core
2. Copy data from SB_out to store buffer
Store Data to Memory

\texttt{sbstore \{mem addr\}, sbid}

Steps:
1. Issue \texttt{sbstore} from core
2. Copy data from SB\_out to store buffer
3. Write back data to memory via existing datapath

Use \textit{existing} store data path and structures (write combining buffers)
Seamless Execution after Interrupts

Instructions:

- `partition_stop` (HARP accelerator)
- `sbsave`: Save the contents of the stream buffer
- `sbrestore`: Ensure the streaming states are identical before and after the interrupt
Seamless Execution after Interrupts

Instructions:

partition_stop (HARP accelerator)

sbsave: Save the contents of the stream buffer

sbrestore: Ensure the streaming states are identical before and after the interrupt

Improves Robustness
Compatibility with Current Software

Original SW
- Lock Acq.
- RDs Table Partition WRs
- Lock Rel.

Original SW in Assembly
- ASM
- LDs
- CMP, BR, etc.
- STs
- ASM
- Executed on unmodified hardware

Modified SW in Assembly
- ASM
- SBLDs
- Hardware Accelerated
- SBSTs
- ASM
- SB Insts
- Executed on HARP
- SB Insts
1. Introduction
2. Background & Motivation
3. HARP System
4. Evaluation
5. Conclusion
6. Discussion
Methodology

- Baseline HARP Parameters:
  - 127 splitters for 255 way partitioning
  - 16 bytes records with 4 bytes keys
  - 64 bytes DRAM bursts: 4 records/burst
- HARP Throughput
  - convert cycle counts and cycle time into bandwidth (GB/sec)
  - 1 million random records
- Streaming Instruction Throughput
  - rate of copy memory from one location to another. 1 GB table.
    - built-in C library
    - hand-optimized X86 scalar assembly
    - hand-optimized X86 vector assembly
HARP Throughput

Baseline throughput: 3.13GB/sec

- 1 thread
- 16 threads
- 1 thread + HARP

Partitioning Throughput (GB/s) vs. Number of Partitions

7.8x improvement
Streaming Throughput

Throughput

4.6 GB/sec

slightly larger than HARP throughput

> 3.13 GB/sec
## Area & Energy

Baseline HARP:
- **Area**: 6.9% in total
- **Power**: 4.3% in total

### HARP Unit

<table>
<thead>
<tr>
<th>Num. Parts.</th>
<th>Area $mm^2$</th>
<th>% Xeon</th>
<th>Power W</th>
<th>% Xeon</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0.16</td>
<td>0.4%</td>
<td>0.01</td>
<td>0.3%</td>
</tr>
<tr>
<td>31</td>
<td>0.31</td>
<td>0.7%</td>
<td>0.02</td>
<td>0.4%</td>
</tr>
<tr>
<td>63</td>
<td>0.63</td>
<td>1.5%</td>
<td>0.04</td>
<td>0.7%</td>
</tr>
<tr>
<td>127</td>
<td>1.34</td>
<td>3.1%</td>
<td>0.06</td>
<td>1.3%</td>
</tr>
<tr>
<td><strong>255</strong></td>
<td><strong>2.83</strong></td>
<td><strong>6.6%</strong></td>
<td><strong>0.11</strong></td>
<td><strong>2.3%</strong></td>
</tr>
<tr>
<td>511</td>
<td>$5.82^4$</td>
<td>13.6%</td>
<td>$0.21^4$</td>
<td>4.2%</td>
</tr>
</tbody>
</table>

### Stream Buffers

<table>
<thead>
<tr>
<th>Num. Parts.</th>
<th>Area $mm^2$</th>
<th>% Xeon</th>
<th>Power W</th>
<th>% Xeon</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0.07</td>
<td>0.2%</td>
<td>0.063</td>
<td>1.3%</td>
</tr>
<tr>
<td>31</td>
<td>0.07</td>
<td>0.2%</td>
<td>0.079</td>
<td>1.6%</td>
</tr>
<tr>
<td>63</td>
<td>1.3</td>
<td>0.2%</td>
<td>0.078</td>
<td>1.6%</td>
</tr>
<tr>
<td>127</td>
<td>0.11</td>
<td>0.3%</td>
<td>0.085</td>
<td>1.7%</td>
</tr>
<tr>
<td><strong>255</strong></td>
<td><strong>0.13</strong></td>
<td><strong>0.3%</strong></td>
<td><strong>0.100</strong></td>
<td><strong>2.0%</strong></td>
</tr>
<tr>
<td>511</td>
<td>0.18</td>
<td>0.4%</td>
<td>0.233</td>
<td>4.7%</td>
</tr>
</tbody>
</table>
Energy

Efficient when compared with software implement. (both serial & parallel)
Skew Tolerance

At most 19% degradation in throughput when records are not well distributed.

Parameter: one partition
Some Other Observations

1. **Number of partitions**
   - throughput highly sensitive to the number of partitions when < 63
   - area & power grow linearly with the number of partitions

1. **Record width**
   - throughput scales linearly with record width

1. **Key width**
   - has slight impact on throughput.
Conclusion

- Advantages of partitioning
  - broadly applicable in database domain, join, sort, aggregation
  - applicable in the non-database domain, map-reduce
- Design of HARP
  - throughput and power efficiency advantages over software
  - flexible yet modular data-centric acceleration framework
- Strategy
  - more partitions: low throughput, efficient in cache locality,
  - fewer partitions: high throughput, relatively inefficient in cache locality
  - need to find the balance between efficiency and throughput
Outline

1. Introduction
2. Background & Motivation
3. HARP System
4. Evaluation
5. Conclusion
6. Discussion
Discussion Points

1. Is multi-thread + HARP helpful to further improve the throughput of data partitioning?
2. Is this hardware design too inflexible?
   - Can HARP efficiently deal with records of larger sizes?
   - Is HARP extensible for applications that require partitioning over records of different, or unknown formats?
1. The authors demonstrate partitioning is important in large SQL databases. Is this application domain broad enough to justify the design of a custom accelerator?
2. In general, for a specific application, is it more profitable to integrate the accelerator with a general purpose CPU, or to design independent hardware?
Oracle’s SPARC M7 Application Accelerator

The M7's most significant innovations revolve around what is known as "software in silicon," a design approach that places software functions directly into the processor.