NAVIGATING BIG DATA with High-Throughput, Energy-Efficient Data Partitioning

Lisa Wu, R.J. Barker, Martha Kim, and Ken Ross
Columbia University
BIG DATA is here

Sources:
IDC Worldwide Big Data Technology and Services 2012-2015 Forecast, #233485, March 2012
The Next Web, DAZEINFO
NetApp

Columbia University
Do sunblock sales correlate with weather?
JOINs = Cross Reference
JOINs = Cross Reference
JOINs = Cross Reference

SALES

WEATHER

JOIN_{DATE}
JOINs = Cross Reference

SALES

WEATHER

JOIN\text{DATE}(\text{SALES, WEATHER})

JOIN\text{DATE}
JOINs = 47% TPC-H Execution Time

Runtime in Join

TPC-H Query

Columbia University

Sunday, July 28, 2013
Naïve JOINs of BIG DATA
Thrash the Cache
Naïve JOINs of BIG DATA Thrash the Cache

Scan SALES

WEATHER
Naïve JOINs of BIG DATA
Thrash the Cache

Scan
SALES

Random Lookup
WEATHER

Columbia University
Naïve JOINs of BIG DATA
Thrash the Cache

Small table doesn’t fit into cache → lookups thrash

Columbia University
Partitioned JOIN

SALES

WEATHER
Partitioned JOIN

SALES

WEATHER

SALES_0

SALES_1

SALES_2

SALES_3
Partitioned JOIN
Partitioned JOIN

SALES

SALES_0

SALES_1

SALES_2

SALES_3

WEATHER

WEATHER_0

WEATHER_1

WEATHER_2

WEATHER_3

DATE SALES_0 WEATHER_0

Columbia University
Partitioned JOIN

SALES

WEATHER

SALES_0

SALES_1

SALES_2

SALES_3

WEATHER_0

WEATHER_1

WEATHER_2

WEATHER_3

DATE

SALES_0

WEATHER_0

DATE

SALES_1

WEATHER_1

DATE

SALES_2

WEATHER_2

DATE

SALES_3

WEATHER_3

Columbia University
JOIN Runtime

Partition WEATHER

Partition SALES

Partitioned Join

Naive Join
JOIN Runtime

Partition WEATHER

Partition SALES

Partitioned Join

Naive Join

Join 0 1 2 3

Columbia University
JOIN Runtime

Partition ≈ 50% of State-of-the-Art Joins
Partition ≈ 50% of State-of-the-Art Joins

Partition
SALES
WEATHER

Partitioned Join
Naive Join

Join
0 1 2 3

Aggregate

time

Columbia University
JOIN Runtime

Partition ≈ 50% of State-of-the-Art Joins

Partition SALES
Partition WEATHER

Join 0 1 2 3

Sort
Aggregate

Partitioned Join
Naive Join

Columbia University
Data Partitioning is...
Data Partitioning is...

- broadly applicable in the database domain
- partitioned-operations reduce I/O cost, increase parallel processing, and reduce shipping costs
Data Partitioning is...

- broadly applicable in the database domain
- partitioned-operations reduce I/O cost, increase parallel processing, and reduce shipping costs
- widely used by commercial databases
- Oracle 11g, IBM DB2, Microsoft SQL Server
Data Partitioning is...

- broadly applicable in the database domain
- partitioned-operations reduce I/O cost, increase parallel processing, and reduce shipping costs
- widely used by commercial databases
  - Oracle 11g, IBM DB2, Microsoft SQL Server
- applicable in the non-database domain
  - divide-and-conquer, map-reduce
SW Partitioning Performance

Partitioning Throughput (GB/s)

1 thread
16 threads
Potential System Memory Throughput

0 150 300 450 600
0 7.5 15 22.5 30
25.6 GB/s
3 GB/s

Columbia University

Sunday, July 28, 2013
New Modules

HARP

Core

L1

SB\text{in}

SB\text{out}

L2

HARP

Core

L1

SB\text{in}

SB\text{out}

L2

Memory

Memory Controller
Research Overview

New Modules

Hardware accelerated range partitioner (HARP): 7.8X more performance @ 7.5X less energy
Research Overview

New Modules

Hardware accelerated range partitioner (HARP): 7.8X more performance @ 7.5X less energy

Streaming framework: Can keep up with the throughput of HARP
Remainder of the Talk

New Modules

- Brief System Overview
- HARP UArch
- Streaming Framework UArch
- HARP and Streaming Framework Evaluation
- Discussion
HARP System Architecture
HARP System Architecture

HARP communicates to/from memory through stream buffers: SBin, SBout
HARP System Architecture

HW Partitioning with SW configuration

HARP communicates to/from memory through stream buffers: SBin, S Bout
Programming Model

Original SW

Lock Acq.  Table Partition  Lock Rel.
Programming Model

Original SW

Lock Acq.  RDs Table Partition WRs  Lock Rel.
Programming Model

Original SW

Original SW in Assembly

RDs Table Partition WRs

Lock Acq.

Lock Rel.
Programming Model

Original SW

Lock Acq. → RDs Table Partition WRs → Lock Rel.

Original SW in Assembly

CMP, BR, etc.
Programming Model

Original SW

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

Original SW in Assembly

- LDs
- CMP, BR, etc.
- STs
Programming Model

Original SW

Lock Acq. | RDs Table Partition WRs | Lock Rel.

Original SW in Assembly

LDs | CMP, BR, etc. | STs

Executed on unmodified hardware
Programming Model

Original SW

Lock Acq. | RDs Table Partition WRs | Lock Rel.

Original SW in Assembly

LDs | CMP, BR, etc. | STs

Executed on unmodified hardware

Modified SW in Assembly
Programming Model

Original SW

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

Original SW in Assembly

- LDs
- CMP, BR, etc.
- STs

Executed on unmodified hardware

Modified SW in Assembly

- Hardware Accelerated
- Executed on HARP
Programming Model

Original SW

Original SW in Assembly

Modified SW in Assembly

SBLDs
SB Insts
Executed on HARP
SB Insts

RDs Table Partition WRs

Lock Acq.

Lock Rel.

LDs
CMP, BR, etc.
STs

Executed on unmodified hardware

SBSTs
Hardware Accelerated

Columbia University
Programming Model

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    SBLD    Hardware Accelerated    SBST    ASM

SB Insts    Executed on HARP    SB Insts

Columbia University
Range Partition

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Range Partition

<table>
<thead>
<tr>
<th></th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Range Partition

<table>
<thead>
<tr>
<th>key</th>
<th>15</th>
<th>8</th>
<th>27</th>
<th>20</th>
<th>52</th>
<th>29</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**splitters**

- 10
- 20
- 30
Range Partition

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Key

Partition 8

Splitters 10, 20, 30

Columbia University
Range Partition

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>8</td>
<td>27</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>16</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>52</td>
</tr>
</tbody>
</table>

Key

Partitions

Splitters
HARP Microarchitecture

From $SB_{in}$

HARP ISA

- set_splitter
- partition_start
- partition_stop

To $SB_{out}$
HARP Microarchitecture

From $SB_{in}$ to $SB_{out}$

HARP ISA
- set_splitter
- partition_start
- partition_stop

To $SB_{out}$
Step 1: HARP Configuration

HARP ISA

set_splitter
partition_start
partition_stop

From $SB_{in}$

To $SB_{out}$

Columbia University
Step 1: HARP Configuration

- **1** Serializer
- **2** Conveyor
- **3** Merge

**HARP ISA**
- set_splitter
- partition_start
- partition_stop

From $SB_{in}$ to $SB_{out}$
Step 2: Signal HARP to Start Processing

From $SB_{in}$

1. Serializer
2. Conveyor
3. Merge

HARP ISA
- set_splitter
- partition_start
- partition_stop

To $SB_{out}$
Step 2: Signal HARP to Start Processing

HARP ISA
- set_splitter
- partition_start
- partition_stop

From $SB_{in}$ to $SB_{out}$
Step 3: Serialize SBin Cachelines into Records

HARP ISA

- set_splitter
- partition_start
- partition_stop
Step 3: Serialize SBin Cachelines into Records

HARP ISA

set_splitter
partition_start
partition_stop

Columbia University
Step 4: Comparator Conveyor

HARP ISA
set_splitter
partition_start
partition_stop

From $SB_{in}$

To $SB_{out}$
Step 4: Comparator Conveyor

From $SB_{in}$ to $SB_{out}$

1. Serializer
2. Conveyor
3. Merge

HARP ISA
- set_splitter
- partition_start
- partition_stop

Columbia University

Sunday, July 28, 2013
Step 5: Merge Output
Records to SB\text{out}

1 Serializer

2 Conveyor

3 Merge

From \textit{SB}_\text{in}

21

To \textit{SB}_\text{out}

Columbia University

set\_splitter
partition\_start
partition\_stop

HARP ISA
Step 5: Merge Output Records to SBout

HARP ISA
- set_splitter
- partition_start
- partition_stop

From $SB_{in}$

1. Serializer
2. Conveyor
3. Merge

To $SB_{out}$

Columbia University
Step 6: Drain In-Flight Records and Signal HARP to Stop Processing

HARP ISA
set_splitter
partition_start
partition_stop

From $SB_{in}$

1 Serializer
2 Conveyor
3 Merge

To $SB_{out}$
Step 6: Drain In-Flight Records and Signal HARP to Stop Processing

HARP ISA

set_splitter
partition_start
partition_stop

To $SB_{out}$
Streaming Framework
Architecture

Inspired by Jouppi’s work
Streaming Framework Architecture

Software-controlled data streaming in/out

Inspired by Jouppi’s work
Step 1: Issue sbload from Core

SB ISA
sbload
sbstore
sbsave
sbrestore

Columbia University
Step 1: Issue `sbload` from Core

SB ISA:
- `sbload`
- `sbstore`
- `sbsave`
- `sbrestore`
Step 2: Send sbload from Req Buffer to Memory

SB ISA
saload
sbstore
sbsave
sbrestore
Step 2: Send sbload from Req Buffer to Memory

SB ISA
- sbload
- sbstore
- sbsave
- sbrestore
Step 2: Send `sbbload` from Req Buffer to Memory

**SB ISA**
- `sbbload`
- `sbbstore`
- `sbbsave`
- `sbbrestore`
Step 3: Data Return from Memory to SBin

SB ISA
- sbload
- sbstore
- sbsave
- sbrestore

C: Cache  S: SB
Step 3: Data Return from Memory to SBin

SB ISA
- sbload
- sbstore
- sbsave
- sbrestore

HARP

Core

Store Buffer

SBin

SBout

L1 C: Cache

SBout

L2

Req Buffer

S: SB

Memory

Columbia University
Step 3: Data Return from Memory to SBin

SB ISA
- sbload
- sbstore
- sbsave
- sbrestore

C: Cache  S: SB
Memory

Core
Store Buffer
SBin
SBout
HARP

Columbia University
Step 4: HARP Pulls Data from SBin and Pushes Data to SBout

SB ISA
- sbload
- sbstore
- sbsave
- sbrestore

C: Cache  S: SB

Memory
Step 4: HARP Pulls Data from SBin and Pushes Data to SBout

SB ISA
- sbload
- sbstore
- sbsave
- sbrestore

Columbia University
Sunday, July 28, 2013
Step 5: Issue \texttt{sbstore} from Core

- \texttt{sbload}
- \texttt{sbstore}
- \texttt{sbsave}
- \texttt{sbrestore}
Step 5: Issue sbstore from Core

SB ISA
sbload
sbstore
sbsave
sbrestore
Step 6: Data Copied from head of SBout to Store Buffer

SB ISA
- sbload
- sbstore
- sbsave
- sbrestore

Diagram:
- HARP
- SB in
- SB out
- L1: Core
- L2
- LLC
- Memory
- Req Buffer

Columbia University
Step 6: Data Copied from head of SBOut to Store Buffer

SB ISA
- sbload
- sbstore
- sbsave
- sbrestore
Step 7: Data Written Back to Memory via Existing Store Datapath

SB ISA
- sbload
- sbstore
- sbsave
- sbrestore
Step 7: Data Written Back to Memory via Existing Store Datapath

SB ISA
- sbload
- sbstore
- sbsave
- sbrestore

Columbia University
Sunday, July 28, 2013
Interrupts and Context Switches

SB ISA
- sbload
- sbstore
- sbsave
- sbrestore

Columbia University
Interrupts and Context Switches

SB ISA
- sbload
- sbstore
- sbsave
- sbrestore

Architectural

Memory

HARP

Core

L1

Store Buffer

L2

LLC

Req Buffer

Columbia University
Interrupts and Context Switches

SB ISA
- sbload
- sbstore
- sbsave
- sbrestore

Architectural

Memory

L1
- Core
  - Store Buffer
- L2
- LLC
  - Req Buffer

HARP
- SBout
- SBin
Interrupts and Context Switches

SB ISA
- sbload
- sbstore
- sbsave
- sbrestore

Core
- L1
  - Store Buffer
L2
LLC
  - Req Buffer
Memory

Architectural

Columbia University
Accelerator Integration Choice

• Tightly coupled and software controlled:
  • area/power savings
  • coherence
  • utilize hardware prefetchers
  • software-managed data layout
  • address-free domain for accelerators
Remainder of the Talk

- Brief System Overview
- HARP UArch
- Streaming Framework UArch
- HARP and Streaming Framework Evaluation
- Discussion and DSE
Evaluation Methodology
Evaluation Methodology

- HARP
  - Bluespec System Verilog implementation
  - Cycle-accurate simulation in BlueSim
  - Synthesis, P&R with Synopsys (32nm std cells)
Evaluation Methodology

• HARP
  • Bluespec System Verilog implementation
  • Cycle-accurate simulation in BlueSim
  • Synthesis, P&R with Synopsys (32nm std cells)

• Streaming framework
  • 3 versions of 1GB table memcpy: c-lib, ASM (scalar), ASM(vector)
  • Conservative area/power estimates with CACTI
Area and Power Overheads

---

<table>
<thead>
<tr>
<th>Number of Partitions</th>
<th>Area [% Xeon core]</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0%</td>
</tr>
<tr>
<td>31</td>
<td>0%</td>
</tr>
<tr>
<td>63</td>
<td>HARP 4%</td>
</tr>
<tr>
<td>127</td>
<td>Stream Buffers 8%</td>
</tr>
<tr>
<td>255</td>
<td>HARP 11%</td>
</tr>
<tr>
<td>511</td>
<td>HARP 15%</td>
</tr>
</tbody>
</table>

Columbia University

Sunday, July 28, 2013
Area and Power Overheads

![Bar chart showing area and power overheads for different numbers of partitions.](chart.png)

- **Area (% Xeon core):**
  - HARP:
    - 0% for 15 partitions
    - 2% for 31 partitions
    - 4% for 63 partitions
    - 6% for 127 partitions
    - 8% for 255 partitions
    - 10% for 511 partitions
  - Stream Buffers:
    - 0% for 15 partitions
    - 2% for 31 partitions
    - 4% for 63 partitions
    - 6% for 127 partitions
    - 8% for 255 partitions
    - 10% for 511 partitions

- **Power (% Xeon core):**
  - 0% for 15 partitions
  - 2% for 31 partitions
  - 4% for 63 partitions
  - 6% for 127 partitions
  - 8% for 255 partitions
  - 10% for 511 partitions

Columbia University

Sunday, July 28, 2013
**SW Partitioning Performance**

![Graph showing partitioning throughput (GB/s) vs. number of partitions for 1 and 16 threads.](image)

- **Partitioning Throughput (GB/s)**
  - Y-axis range: 0 to 8
  - Y-axis labels: 0, 2, 4, 6, 8

- **Number of Partitions**
  - X-axis range: 0 to 600
  - X-axis labels: 0, 150, 300, 450, 600

- **Legend**:
  - 1 thread
  - 16 threads

- **Note**: Graph illustrates the performance of SW partitioning for 1 and 16 threads as the number of partitions increases. The throughput decreases significantly with an increase in the number of partitions.
Performance Evaluation

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

1 thread
16 threads
1 thread + HARP

Columbia University

Sunday, July 28, 2013
Performance Evaluation

**Graph:**
- **Y-axis:** Partitioning Throughput (GB/s)
- **X-axis:** Number of Partitions
- **Legend:**
  - Blue diamonds: 1 thread
  - Blue circles: 16 threads
  - Red squares: 1 thread + HARP

- **Analysis:**
  - The graph shows a comparison of throughput for 1 thread, 16 threads, and 1 thread with HARP.
  - The throughput decreases as the number of partitions increases.
  - The throughput for 1 thread + HARP is significantly higher than 1 thread and 16 threads, indicating a 7.8x performance improvement.

**Note:**
- Columbia University
- Sunday, July 28, 2013
Performance Evaluation

Partitioning Throughput (GB/s)

Number of Partitions

1 thread
16 threads
1 thread + HARP

8.8x
7.8x

Columbia University
Streaming Framework Provides Sufficient BW to Feed HARP?

![Graph showing the relationship between number of partitions and partitioning throughput. The graph plots the throughput in GB/s against the number of partitions, with a downward trend indicating decreased throughput as the number of partitions increases. The data points for a single thread plus HARP are marked.]
Streaming Framework Provides Sufficient BW to Feed HARP?

![Graph]

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

- **memcpy**
- **vector ASM**
- **scalar ASM**
- 1 thread + HARP

Our measurements

Columbia University
Streaming Framework Provides Sufficient BW to Feed HARP?

Partitioning Throughput (GB/s)

Number of Partitions

Our measurements
From the literature

memcpy
vector ASM
memcpy
scalar ASM

1 thread + HARP

Columbia University

Sunday, July 28, 2013
HARP Energy vs. SW

Partitioning Energy (J/GB)

- 1 thread
- 16 threads
- 1 thread + HARP

Number of Partitions

Columbia University
HARP Energy vs. SW

Partitioning Energy (J/GB)

- 1 thread
- 16 threads
- 1 thread + HARP

Number of Partitions

0 150 300 450 600

Columbia University

6.3x

Sunday, July 28, 2013
HARP Energy vs. SW

Partitioning Energy (J/GB) vs. Number of Partitions

- 1 thread
- 16 threads
- 1 thread + HARP

6.3x improvement over SW
7.3x improvement over 1 thread
Remainder of the Talk

- Brief System Overview
- HARP UArch
- Streaming Framework UArch
- HARP and Streaming Framework Evaluation
- Discussion
Design Space Exploration (in the paper)

- 255-way partitioning
- 4B keys
- 16B records
Design Space Exploration (in the paper)

- Way partitioning

4B keys
16B records
Design Space Exploration (in the paper)

- 255-way partitioning
- 4B keys
- 16B records

Columbia University
Design Space Exploration (in the paper)
Coping with Fixed Resources: Partitioning Factor > Partitioner Size
Coping with Fixed Resources: Partitioning Factor > Partitioner Size
Coping with Fixed Resources: Partitioning Factor > Partitioner Size
Coping with Fixed Resources: Record Width > Record Size
Coping with Fixed Resources: Record Width > Record Size
Coping with Fixed Resources: Record Width > Record Size
Conclusion

• Data partitioning, which does not appear compute-heavy, can still benefit from acceleration

• Microarchitecture to pair streaming accelerator(s) to work closely with CPU

• Demonstrate how accelerators can rebalance system and improve memory bandwidth utilization, a scarce resource in big data analytics