Dark Silicon Accelerators for Database Indexing

Onur Kocberber, Kevin Lim, Babak Falsafi, Partha Ranganathan, Stavros Harizopoulos

© 2012 EPFL PARSA
Dark Silicon and Big Data Challenges

- Data explosion
  - Data growing faster than technology

- End of “Free energy”
  - Higher density → higher energy

- Challenge: CPUs ill-matched to server workloads
  - Most of time waiting for data rather than computing

Need to specialize for data-centric workloads
How Do Data-Centric Workloads Access Data?

• Databases create and use an index
  – Data structures for fast data lookup
  – Most often balanced tree or hash table
  – Frequently accessed

• Indexing is pointer-intensive
  – Underutilize general-purpose CPUs
  – IPCs as low as **0.25** on OoO core
Contribution: Database Indexing Widget

• Index lookups on general-purpose CPUs:
  – Pointer-intensive \( \rightarrow \) low IPC
  – Time-intensive \( \rightarrow \) poor energy-efficiency

• Database Indexing Widget
  – Dedicated hardware for database index lookups
  – Full-service offload: core sleeps when widget runs
  – Up to 65% less energy per query
Outline

Introduction
Indexing in Databases
Indexing Widget
Results
Modern Databases and Indexing

Two types of contemporary in-memory databases:

- **Column-store analytical processing**
  - MonetDB with DSS

- **Scale-out transaction processing**
  - VoltDB with OLTP

- Two fundamental indexing operations
  - Hash table probe
  - Tree traversal
How Much Time is Spent Indexing?
Measurement on Xeon 5670 CPU with HW Counters

Indexing can account for up to 73% of execution
Example: Hash Join

SQL:
SELECT A_name FROM A,B WHERE A_age = B_age

1 Build

Table A (2M rows)

<table>
<thead>
<tr>
<th>age</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>26</td>
</tr>
<tr>
<td>3</td>
<td>71</td>
</tr>
<tr>
<td>4</td>
<td>19</td>
</tr>
</tbody>
</table>

Table B (60M rows)

<table>
<thead>
<tr>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
</tbody>
</table>

Hash Table (A)

2 Probe

Result

<table>
<thead>
<tr>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

Hash table probes dominate execution

© 2012 EPFL PARSA
Indexing with Hash Table Probes

Each hash probe operation:
→ 100-200 dynamic instructions: hash, then chase pointers
→ 50% memory ref.
Indexing with Tree Traversals

SQL:
```
SELECT A_Product, A_Customer FROM A WHERE A_age = 25
```

Index on A_age

Key

25

10

8

12

15

25

Tuple Ptr

Customer | Age | Date | Product

Result
Indexing with Tree Traversals

SQL:
SELECT A_Product, A_Customer FROM A WHERE A_age = 25

Index on A_age

Each index traversal:
→ 10K-15K dynamic instructions: lots of pointer chasing
→ 50-60% memory ref.
Outline

Introduction
Indexing in Databases
Indexing Widget
Results
Indexing Widget Overview

• Dedicated offload engine for index lookups
  – Activated on-demand by the core
  – Full-service index lookup
  – Core sleeps when widget runs

• Widget features
  – Efficient: Specialized control and functional units
  – Low-latency: Caches frequently-accessed index data
  – Tightly-integrated: Uses core’s L1-D and TLB
Widget Details

From Core

Configuration Registers
- Index Addr.
- Key
- Search Type
- Result Table Addr.
- Data type

Controller (FSM)
- Hash
- Tree

Computational Logic

Buffer (SRAM)

1. Configure
2. Run
3. Return

© 2012 EPFL PARSA
If (hasWidget) {
  widget.index=&A;
  widget.key=&B;
  widget.type=EQUAL;
  widget.result=&R;
  widget.data= int;
  ...
  ...
  widget.run();
} else {
  Hashprobe();
}
Widget Details

Configuration Registers
- Index Addr.
- Key
- Search Type
- Result Table Addr.
- Data type

Controller (FSM)
- Hash
- Tree

Computational Logic

Buffer (SRAM)

From Core

To/From L1

© 2012 EPFL PARSA
Widget Details

From Core

Configuration Registers
- Index Addr.
- Key
- Search Type
- Result Table Addr.
- Data type

Controller (FSM)
- Hash
- Tree

Computational Logic

Buffer (SRAM)

To/From L1

Store

To/From L1

© 2012 EPFL PARSA
Methodology

• First-order analytical model
  – Execution traces: Pin
  – Execution profiling: Vtune, Oprofile

• Benchmark Applications
  ◦ OLTP: TPC-C on VoltDB
  ◦ DSS: TPC-H on MonetDB

• Model Parameters
  ◦ L1 / L2 / Off-chip latency: 2 / 12 / 200 cycles
  ◦ Widget buffer: 2-way set associative cache

• Energy Estimations
  ◦ Mcpat

© 2012 EPFL PARSA
Energy Efficiency with Indexing Widget

Up to 65% reduction in energy
Performance with Indexing Widget

Widget does not hurt performance
Conclusions

• Data explosion, dark silicon trends call for specialization
  – Rethinking of architectures to achieve efficiency

• Databases spend significant time in indexing
  – Mostly pointer chasing: general purpose CPUs are poorly suited

• Augment CPU with indexing widget
  – Dedicated offload engine: core sleeps when widget runs
  – Improves efficiency: 65% less energy, 3x faster query execution

More challenges:
Data types, data sharing, generalization...
Thanks!