Big Data:

To Process-in-Storage or NOT to Process-in-Storage, that is the question

ICRI-CI retreat: May 24th 2016
Outline

• **Environment:** Big Data
• **Target:** reduction in energy
  • Is it Data movements or “performance”? Both?
• **Question:** where should the data be processed?
  • At the General Purpose CPU?
  • At in Memory?
  • At the Storage/NIC?

• Remember: Heterogenous computing is here ➔ WHEN and WHERE?
Big Data ➔ usage of DATA

- Read Once
- Non-Temporal Memory Access

Funnel

\[ \beta = \frac{BW_{out}}{BW_{in}} \]
Machine Learning

Input: Unstructured data

Structured data (aggregation)

Model creation

Data structuring = ETL

Model usage @ client
Does Big Data exhibit special memory access pattern?

It probably should since

- Revisiting ALL Big Data items will cause huge/slow data transfers from Data sources
- There are 2 access modes of memory operations:
  - Temporal Memory Access
  - Non-Temporal Memory access
- Many Big Data computations exhibit a Non-Temporal Memory-Accesses and/or Funnel operation
Non-Temporal Memory access
Initial analysis: Hadoop-grep Single Memory Access Pattern

~50% of Hadoop-grep unique memory references are single access
Non-Temporal Memory Accesses

Preliminary Results

- **WordCount:**
  - Access to Storage:
  - Non-temporal locality

- **Sort:**
  - Access to Storage:
  - NO Non-temporal locality
Current systems

- Memory subsystem is tuned for “Temporal Memory Access”
  - DRAM – tuned for repeated page access
  - Cache – tuned for repeated cache block access

However, many Big Data applications exhibit Non-Temporal Memory Accesses (NTMA)
Where energy is wasted?

• DRAM

• Limited BW
### Rough Energy Numbers (45nm)

<table>
<thead>
<tr>
<th>Integer</th>
<th></th>
<th>FP</th>
<th></th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td></td>
<td>FAdd</td>
<td></td>
<td>Cache (64bit)</td>
</tr>
<tr>
<td>8 bit</td>
<td>0.03pJ</td>
<td>16 bit</td>
<td>0.4pJ</td>
<td></td>
</tr>
<tr>
<td>32 bit</td>
<td>0.1pJ</td>
<td>32 bit</td>
<td>0.9pJ</td>
<td>8KB 10pJ</td>
</tr>
<tr>
<td>Mult</td>
<td></td>
<td>FMult</td>
<td></td>
<td>32KB 20pJ</td>
</tr>
<tr>
<td>8 bit</td>
<td>0.2pJ</td>
<td>16 bit</td>
<td>1pJ</td>
<td>1MB 100pJ</td>
</tr>
<tr>
<td>32 bit</td>
<td>3 pJ</td>
<td>32 bit</td>
<td>4pJ</td>
<td>DRAM 1.3-2.6nJ</td>
</tr>
</tbody>
</table>

### Instruction Energy Breakdown

- **25pJ** I-Cache Access
- **6pJ** Register File Access
- **70 pJ** Add

---

From: Bill Dally (nVidia and Stanford), Efficiency and Parallelism, the challenges of future computing
Data Center Energy Specs

Malladi, ISCA, 2012
Memory Subsystem - copies

- **Source**
  - NV Storage (TBs)
  - DRAM (GBs)
  - LL Cache (10's MBs)
  - L2$ (MBs)
  - L1$ (10's KBs)
  - Registers (KBs)

- **Destination**
  - Core

- **Copy 1 (main memory)**
  - NV Storage → DRAM (3GB/sec)

- **Copy 2 (LL Cache)**
  - DRAM → LL Cache (25GB/sec)

- **Copy 3 (L2 Cache)**
  - LL Cache → L2$ (500GB/sec)

- **Copy 4 (L1 Cache)**
  - L2$ → L1$ (Copy 5 ( Registers))

- **Copy 5 (Registers)**
  - L1$ → Core (500GB/sec)
Memory Subsystem – DRAM bypass == DDIO

Potential savings:

@ 0.5n J/B (DRAM)
10 – 20 GB/s NV BW

⇒ 5W – 10W

Initial Experiment

- Example program: read file from disk and XOR all values
- DDIO-aware code on a real system
  - Small buffer (fit into 2 ways of LLC)
  - Low latency from write to read (avoid evictions)
  - Zero-copy (O_DIRECT flag)
  - Bypass OS page cache (O_DIRECT flag)
  - Run code on chip that is connected to the SSD (OS affinity)
- Compare system with DDIO enabled and DDIO disabled
- Measure runtime, power and energy

- save of 4W out of server’s 180W → ~2%
Bandwidth

When should we use Funnel at the Data source (Storage/NIC)?
Memory Hierarchy is Optimized for
A: Bandwidth issue ➔ System are built for Temporal Locality

<table>
<thead>
<tr>
<th>Size</th>
<th>BW</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBs</td>
<td>3-20GB/sec</td>
</tr>
<tr>
<td>GBs</td>
<td>25GB/sec</td>
</tr>
<tr>
<td>10’s MBs</td>
<td>500GB/sec</td>
</tr>
<tr>
<td>MBs</td>
<td>TB/sec</td>
</tr>
</tbody>
</table>

Existing BW vs. NTMA Desired BW
B: Memory access per operation impact BW

Read Once – Non-Temporal Memory Accesses

Temporal Memory Accesses

Hint: Memory access per operation
Solution:

Flow of “Non-Temporal Data Accesses”

- Application’s high Byte/Instruction (memory accesses per Instruction)
- “Read Once” – Non Temporal locality Memory Access
- Limited BW

Funnel will be effective when:

*private communication with: Moinuddin Qureshi*
“Funnel”ing “Read-Once” data in storage


**K. Eshghi and R. Micheloni.** *SSD Architecture and PCI Express Interface*
Analytical model of the Funnel

\[ \beta = \frac{\text{BW}_{\text{OUT}}}{\text{BW}_{\text{IN}}} \]
Purposed Architecture

Baseline Configuration

- SSD Storage
- B = Bandwidth
- PCIe
- Funnel
- CPU performs NTMA and TMA work

Funnel Configurations

- SSD performs NTMA work
- Funnel
- PCIe
- CPU performs TMA work

B = Bandwidth

Funnel Configuration

- SSD Storage
- PCIe
- Funnel
- CPU performs NTMA and TMA work
Funnel Performance

- CPU performs NTMA and TMA work
- SSD Storage
- PCIe
- Funnel

SSD performs NTMA work

CPU performs: TMA work

Performance Graph

**Performance**

- CPU becomes bottleneck
- CPU becomes bottleneck

B = Bandwidth

PCIe TL

B

$B_{PCIE}$

$B_{SSD}$

$B_{TL}$

26
Funnel energy

SSD Storage → Funnel → PCIe

B = Bandwidth

CPU performs NTMA and TMA work

SSD performs NTMA work

Funnel processor overhead

CPU becomes the bottleneck

Comparing funnel with baseline energy

Funnel configuration

Energy grows as performance advantage shrinks

Funnel processor overhead

CPU becomes the bottleneck

Baseline configuration

PCIe is the bottleneck

CPU becomes the bottleneck

$B = \beta$

$E_{2,Lc}$
Energy consumption
Funnel vs. Core performance

Energy/GByte

Energy as a Function of the relation $f_{NTL}/f_{CPU,NTL}$

- $E_{total}$
- $E_{system}$
- $E_{NTL}$

Core Performance
Funnel performance
Energy consumption
Introducing System’s Bottlenecks

Energy as a Function of the relation $f_{NTL} / f_{CPU,NTL}$

Bottlenecks (PCI or Core)
- before minimum
- after minimum

Energy/GByte

Core Performance
Funnel performance
Open issues for research

- SW and OS
  - Co-Processor or
  - Heterogeneous system
- Compatibility
- Application awareness of the features
- …
Solution: ?

- Non-Temporal Memory Accesses should be processed as close as possible to the data source.
- Data that exhibit Temporal Locality should use current Memory Hierarchy.
- Use Machine Learning (context aware*) to distinguish between the two phases.

Open questions:
- SW model
- Shared Data
- HW implementation
- Computational requirement at the “Funnel”

Thank You