Handling Memory Accesses in Big Data Environment

Chipex 2016
A New Architecture Avenues in Big Data Environment

- The Era of Heterogeneous
  - HW/SW fits application
  - Dynamic tuning
  - Accelerators
  - \( \Rightarrow \) performance, energy efficiency
- Big Data = big
  - In general non repeated access to all the “Big Data”
  - What are the implications?
Heterogeneous computing: Application Specific Accelerators

Continue performance trend by tuned architecture to bypass current technological hurdles
A New Architecture Avenues in Big Data Environment

- Heterogeneous computing – "tuning" HW to respond to specific needs
- Example: Big Data memory access pattern
- Potential savings
  - Reduction of Data Movements and bypass DRAM
  - Bandwidth issue
    ➔ Potential solution
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)

Data structuring = ETL

Model creation

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

![Graphs showing I/O utilization for WordCount and Sort]
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>8KB</td>
</tr>
<tr>
<td>32 bit</td>
<td>0.1pJ</td>
<td>32 bit</td>
<td>0.9pJ</td>
<td>32KB</td>
</tr>
<tr>
<td>Mult</td>
<td></td>
<td>FMult</td>
<td></td>
<td>1MB</td>
</tr>
<tr>
<td>8 bit</td>
<td>0.2pJ</td>
<td>16 bit</td>
<td>1pJ</td>
<td></td>
</tr>
<tr>
<td>32 bit</td>
<td>3 pJ</td>
<td>32 bit</td>
<td>4pJ</td>
<td></td>
</tr>
</tbody>
</table>

#### Instruction Energy Breakdown

- Register File Access: 6pJ
- Add: 70pJ
- Control: 70pJ

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

Malladi, ISCA, 2012
Memory Subsystem - copies

Size

TBs

GBs

10’s MBs

MBs

L1$

L2$

LL Cache

DRAM

NV Storage

BW

3GB/sec

25GB/sec

500GB/sec

Copy 1 (main memory)

Copy 2 (LL Cache)

Copy 3 (L2 Cache)

Copy 4 (L1 Cache)

Copy 5 (Registers) - Destination

Source

Core

10’s KBs

KBs
Memory Subsystem – DRAM bypass == DDIO

Potential savings:

@ 0.5n J/B (DRAM)
10 – 20 GB/s NV BW

⇒ 5W – 10W

Reference: “Optimizing Read-Once Data Flow in Big-Data Applications”
Bandwidth

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

- **Size**
  - TBs
  - GBs
  - 10's MBs
  - MBs
  - 10’s KBs
  - KBs

- **BW**
  - NV Storage: 3-20GB/sec
  - DRAM: 25GB/sec
  - LLC Cache
  - L2$: 500GB/sec
  - L1$: TB/sec

- **Existing BW**
- **NTMA Desired BW**

Highest Bandwidth
B: Memory access per operation impact BW

Read Once – Non-Temporal Memory Accesses

- SSD = CPU Bandwidth

Temporal Memory Accesses

- CPU Bandwidth
- SSD Bandwidth

Hint: Memory access per operation
Solution:

Flow of “Non-Temporal Data Accesses”

Use Funnel when Bandwidth bottleneck occurs:
- “high” memory accesses per Instruction
- Limited BW
- Non temporal locality memory access

*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

CPU performs NTMA and TMA work

Funnel Configurations

SSD performs NTMA work

CPU performs TMA work
Funnel Performance

CPU becomes bottleneck

SSD Storage
B=Bandwidth

PCIe

Funnel

B

CPU performs NTMA and TMA work

Performance Graph

Performance

CPU becomes bottleneck

SSD performs NTMA work

CPU performs: TMA work

Performance

0.2
0.4
0.6
0.8
1.0

beta

baseline

funnel
Funnel energy

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

- SSD performs NTMA work
- CPU performs TMA work

CPU becomes the bottleneck

Funnel processor overhead

Energy grows as performance advantage shrinks

Funnel configuration

Baseline configuration

PCIe is the bottleneck

CPU becomes the bottleneck

βB

B=Bandwidth

Energy

beta
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”

Summary

Memory access is a critical path in computing

Funnel should be used for:
- Resolve BW systems’ bottleneck for specific applications
- Solve the System’s BW issues for “Read Once” cases
- Reduction of Data movement
- Free up system’s memory resources (re-Spark)
- Simple-energy-efficient engines at the front end

Issues
- ...


THANK YOU