Data Motif-based Proxy Benchmarks for Big Data and AI Workloads

Wanling Gao

ICT, Chinese Academy of Sciences

HPCA 2019, Washington D.C., USA
Proxy Benchmarks for Big Data and AI Workloads

100X Runtime Speedup
90%+ Average Accuracy
Data Adaptability
Configuration Adaptability
Cross Architecture

Proxy Benchmark Generating Methodology

DAG-like combination with different weights
(Machine Learning Model)

Data Motifs: A Lens Towards Fully Understanding Big Data and AI Workloads

Matrix Sampling Transform Logic
Graph Set Statistics Sort

Most Time-consuming Units of Computation

Data Characteristics
Type Source
Pattern Scale

• Data Motifs: A Lens Towards Fully Understanding Big Data and AI Workloads. PACT’18.
• Data Motif-based Proxy Benchmarks for Big Data and AI Workloads. IISWC 2018.
BigDataBench Publications

- Data Motifs: A Lens Towards Fully Understanding Big Data and AI Workloads. **PACT’18.**
- Understanding Big Data Analytics Workloads on Modern Processors. **TPDS’16**
- **Auto-tuning Spark Big Data Workloads on POWER8: Prediction-Based Dynamic SMT.** **PACT’16**
- BigDataBench: a Big Data Benchmark Suite from Internet Services. **HPCA’14**
- CVR: Efficient Vectorization of SpMV on X86 Processors. **CGO’18.**
- BOPS, Not FLOPS! A New Metric, Measuring Tool, and Roofline Performance Model For Datacenter Computing. **Technical report.**
- Data Motif-based Proxy Benchmarks for Big Data and AI Workloads. **IISWC 2018.**
Overview

- Simulation of Big Data and AI Workloads
  - Challenges & Motivation

- Data Motif-based Proxy Benchmarks

- Evaluation on X86 Processor

- Case Study

- How to Use
Simulation Requirements

- **How to balance simulation accuracy and time?**

- SPEC 2006: *Trillions* of instructions per benchmark

- Simulation speed:
  - *1,000X slowdown vs. Hardware → weeks or months per experiment*

*From: https://parsa.epfl.ch/simflex/doc/SimFlex-tutorial-isca2006.pdf*
Traditional Simulation Method

- **Kernel Benchmark**
  - Key codes abstracted from actual program

  - NAS Parallel Benchmarks

- **Limitations**
  - Big data and AI workloads exist no single kernel
  - Insufficient to completely reflect behaviors
Traditional Simulation Method

- Synthetic traces or benchmarks
  - Modeling the program execution
    - generating a synthetic trace through a statistical profile
    - Generating assembly code or C code

- Limitations
  - OS scheduling
  - Execution path
  - Random variable
  - Data adaptability
  - Configuration adaptability
  - New architecture
  - New software

Nondeterminism
Portability
Support
Simulation Challenges

The complexity of big data and AI workloads aggravates the challenges ......
Heavy Software Stack

- Thousands of billions of instructions
  - Long running time even on real machines
- Simulators have limited supports on complex software stacks
  - Distributed environment further aggravates this issue
    - For example: Hadoop modes
      - Standalone mode
      - Pseudo-distributed mode
      - Fully-distributed mode
- Different modes have large behavior differences
Data Impact

- Input data has a great impact on workload behaviors
  - Data type, source, pattern, distribution

- Text
- Image
- Graph
- Matrix
- Sparse
- Dense
- Structured
Motivation

- Traditional simulation method
  - A case-by-case solution
  - Focus on specific workload and architecture
  - Ignore the impact of input data

_Not suit for big data and AI workloads_
Overview

- Simulation of Big Data and AI Workloads
  - Challenges & Motivation

- Data Motif-based Proxy Benchmarks

- Evaluation on X86 Processor

- Case Study

- How to Use
How to Understand a Big Data or AI Workload

- A pipeline of **Data Motifs**
- performed on initial or intermediate data inputs

Execution Pipeline

#### AlexNet Units of Computation:

1. **Convolution:** 36.91%
   ----Conv2d

2. **Sampling:** 13.45%
   ----Max Pooling
   ----Dropout

3. **Matrix Multiply:** 48.87%
   ----Fully Connected

4. **Basic Statics:** 0.76%
   ----Normalization

BigDataBench | HPCA 2019
How to Understand a Big Data or AI Workload (cont’)

- A pipeline of **Data Motifs**
  - performed on initial or intermediate data inputs

**Execution Pipeline**

1) Builds Gaussian pyramid: 13.16%
   - Matrix Multiplication
   - Transform
   - DownSample

2) Builds DoG pyramid: 4.17%
   - Matrix Subtraction
   - Matrix Inversion

3) Finds keypoints: 26.01%
   - Sort
   - Matrix Inversion

4) Compute scale, orientation & descriptors: 53.11%
   - Statistic

5) Sort: 0.53%
   - Sort

**SIFT: Units of Computation**
Data Motif Identifying Methodology

- **Data Motif**
  - Most time-consuming
  - Units of computation

- **Methodology**
  - Algorithmic analysis
  - Profiling analysis

---

Eight Data Motifs

- 40+ algorithms with a broad spectrum
  - Data mining/Machine learning
  - Natural language processing
  - Computer vision
  - Bioinformatics

<table>
<thead>
<tr>
<th>Operations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Matrix/Vector operations</td>
</tr>
<tr>
<td>Sampling</td>
<td>Selecting a subset samples according to certain statistical population</td>
</tr>
<tr>
<td>Logic</td>
<td>Bit manipulation operations</td>
</tr>
<tr>
<td>Transform</td>
<td>FFT, DCT, Wavelet transform</td>
</tr>
<tr>
<td>Set</td>
<td>Union, intersection, complement</td>
</tr>
<tr>
<td>Graph</td>
<td>Graph-theoretical computations, i.e. graph traversal</td>
</tr>
<tr>
<td>Statistic</td>
<td>Statistical computations</td>
</tr>
<tr>
<td>Sort</td>
<td>Sorting the elements in a certain order</td>
</tr>
</tbody>
</table>
Data Motif Implementation

- Various Data Input
  - Data type
    - Text, graph, Matrix
  - Data pattern and distribution

- Implementation
  - Light-weight
    - POSIX thread model
  - Preserve the execution model
    - Hadoop, TensorFlow
Proxy benchmarks

- Data Motif-based proxy benchmark generating methodology
  - The first step is understanding big data and AI workloads
Proxy benchmarks

- Data Motif-based proxy benchmark generating methodology
  - A DAG-like combination of data motifs
  - An auto-tuning tool using machine learning model
Proxy benchmarks

- Data Motif-based proxy benchmark generating methodology
  - Mimic system and micro-architectural behaviors
Proxy Benchmark Construction (1)

- Decomposing a big data or AI workload
  - Hotspot function
  - Execution time breakdown---initial weights

```
Decomposing
  Big Data and AI Workloads
  Motif components
  Initial Weights

Proxy Benchmark

Feature Selecting
  Metrics (M)
    - System metrics
    - Micro-architectural metrics
  Parameters (P)
    - Input data size
    - Weight
    - Number of tasks
    - Chunk size

Auto-Tuning
  Parameter Initialization
  Impact analysis
    - Input data size
    - Weight
    - Number of tasks
    - Chunk size
  Tuned Parameters
  Adjusting Stage
  Feedback Stage
  Accuracy Evaluation
  Deviation analysis
  Yes
```

BigDataBench | HPCA 2019
Proxy Benchmark Construction (2)

- **Feature Selecting**
  - **Metrics:** \( \vec{M} = (\text{runtime}, \text{IPC}, \text{MIPS}, L1\text{D hitR}, L2\text{ hitR}, \ldots) \)
  - **Parameters:** \( \vec{P} = (\text{dataSize}, \text{chunkSize}, \text{numTasks}, \text{weight} \\	ext{batchSize}, \text{totalSize}, \text{heightSize}, \text{widthSize}, \text{numChannels}) \)
Tunable Parameters for Data Motif

Find the optimal $\bar{P}$ whose corresponding $\bar{M}$ is close enough to the metrics of original workload.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataSize</td>
<td>The input data size for each big data motif</td>
</tr>
<tr>
<td>chunkSize</td>
<td>The data block size processed by each thread for each big data motif</td>
</tr>
<tr>
<td>numTasks</td>
<td>The process and thread numbers for each big data and AI data motif</td>
</tr>
<tr>
<td>batchSize</td>
<td>The batch size of each iteration for each AI data motif</td>
</tr>
<tr>
<td>totalSize</td>
<td>The total input data size need to be processed for each AI data motif</td>
</tr>
<tr>
<td>heightSize</td>
<td>The height dimension for one input data or filter</td>
</tr>
<tr>
<td>widthSize</td>
<td>The width dimension for one input data or filter</td>
</tr>
<tr>
<td>numChannels</td>
<td>The channel number for one input data or filter</td>
</tr>
<tr>
<td>weight</td>
<td>The contribution for each data motif</td>
</tr>
</tbody>
</table>
Proxy Benchmark Construction (3)

- Adjusting Stage
  - Decision tree based mechanism
  - Find the parameter’s impact on each metric

```
Big Data and AI Workloads
  | Decomposing
  | Motif components
  | Initial Weights
  | Proxy Benchmark

Feature Selecting
- Metrics (M)
  - System metrics
  - Micro-architectural metrics
- Parameters (P)
  - Input data size
  - Weight
  - Number of tasks
  - Chunk size

Parameter Initialization

Impact analysis
- Input data size
- Weight
- Number of tasks
- Chunk size

Auto-Tuning
- Tuned parameters
- Adjusting Stage

Feedback Stage
- Deviation analysis
- Accuracy evaluation
- No → Tuned parameters
- Yes → Qualified Proxy Benchmark
```

---

BigDataBench | HPCA 2019 | ICT
---|---|---
Feedback Stage
- Evaluate the accuracy with current parameters
- Feedback the metric with large deviation
- Adjusting-Feedback iterations until reaching the specified accuracy
Methodology Comparison

- Traditional simulation methodology
  - Kernel benchmark
  - Synthetic trace
  - Synthetic benchmark

<table>
<thead>
<tr>
<th>Methodology</th>
<th>Typical Benchmark or Tool</th>
<th>Data Set</th>
<th>Portable Cost</th>
<th>Multi-core Scalability</th>
<th>Cross Architecture</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel Benchmark</td>
<td>NPB [31]</td>
<td>Fixed</td>
<td>Recompile</td>
<td>Yes</td>
<td>Yes</td>
<td>Low</td>
</tr>
<tr>
<td>Synthetic Trace Method</td>
<td>SimPoint [32]</td>
<td>Fixed</td>
<td>Regenerate</td>
<td>No</td>
<td>No</td>
<td>High</td>
</tr>
<tr>
<td>Synthetic Benchmark</td>
<td>PerfProx [33]</td>
<td>Fixed</td>
<td>Regenerate</td>
<td>No</td>
<td>No</td>
<td>High</td>
</tr>
<tr>
<td>Data Motif-Based Proxy Benchmark</td>
<td>Data Motif Benchmark</td>
<td>On-demand</td>
<td>Recompile</td>
<td>Yes</td>
<td>Yes</td>
<td>High</td>
</tr>
</tbody>
</table>
## Five Proxy Benchmarks

<table>
<thead>
<tr>
<th>Big Data &amp; AI Benchmark</th>
<th>Workload Pattern</th>
<th>Data Set</th>
<th>Involved Data Motifs</th>
<th>Data Motif Implementations of Proxy Benchmark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop TeraSort</td>
<td>I/O Intensive</td>
<td>Text</td>
<td>Sort, Sampling, Graph</td>
<td>Quick sort; Merge sort; Random sampling; Interval sampling; Graph construction; Graph traversal</td>
</tr>
<tr>
<td>Hadoop K-means</td>
<td>CPU Intensive</td>
<td>Vectors</td>
<td>Matrix, Sort, Statistics</td>
<td>Vector euclidean distance; Cosine distance; Quick sort; Merge sort; Cluster count; Average computation</td>
</tr>
<tr>
<td>Hadoop PageRank</td>
<td>CPU Intensive</td>
<td>Graph</td>
<td>Matrix, Sort, Statistics</td>
<td>Matrix construction; Matrix multiplication; Quick sort; Min/max calculation; Out degree and in degree count of nodes</td>
</tr>
<tr>
<td>TensorFlow AlexNet</td>
<td>CPU Intensive</td>
<td>Image/Matrix</td>
<td>Matrix, Sampling, Transform, Statistics</td>
<td>Fully connected; Max Pooling; Convolution; Batch normalization</td>
</tr>
<tr>
<td>TensorFlow Inception-V3</td>
<td>CPU Intensive</td>
<td>Image/Matrix</td>
<td>Matrix, Sampling, Logic, Transform, Statistics</td>
<td>Fully connected; Softmax; Max pooling; Average pooling; Dropout; ReLu; Convolution; Batch normalization</td>
</tr>
</tbody>
</table>
Overview

- Simulation of Big Data and AI Workloads
  - Challenges & Motivation

- Data Motif-based Proxy Benchmarks

- Evaluation on X86 Processor

- Case Study

- How to Use
Experiment Setup

- Five-node cluster

- Big Data Workloads (Hadoop)
  - TeraSort: 100GB records
  - Kmeans: 100GB sparse vector data (90% sparsity)
  - PageRank: $2^{26}$-vertex graph data

- AI Workloads (TensorFlow)
  - AlexNet: CIFAR-10 dataset with 10,000 steps
  - Inception-V3: ImageNet dataset with 1,000 steps
## Evaluation on X86 Xeon E5645

### Configurations (Westmere)

<table>
<thead>
<tr>
<th>Hardware Configurations</th>
<th>Intel CPU Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Type</td>
<td>Intel® Xeon E5645</td>
</tr>
<tr>
<td>Intel CPU Core</td>
<td>6 <a href="mailto:cores@2.40G">cores@2.40G</a></td>
</tr>
<tr>
<td>L1 DCache</td>
<td>L1 DCache</td>
</tr>
<tr>
<td>6 × 32 KB</td>
<td>6 × 32 KB</td>
</tr>
<tr>
<td>L1 ICache</td>
<td>L2 Cache</td>
</tr>
<tr>
<td>6 × 32 KB</td>
<td>6 × 256 KB</td>
</tr>
<tr>
<td>Memory</td>
<td>Ethernet</td>
</tr>
<tr>
<td>32GB,DDR3</td>
<td>Hyper-Threading</td>
</tr>
<tr>
<td>1Gb</td>
<td>Disabled</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software Configurations</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Linux</td>
</tr>
<tr>
<td>CentOS 6.4</td>
<td>3.11.10</td>
</tr>
<tr>
<td>Kernel</td>
<td>JDK</td>
</tr>
<tr>
<td>JDK Version</td>
<td>1.7.0</td>
</tr>
<tr>
<td>Hadoop Version</td>
<td>Hadoop</td>
</tr>
<tr>
<td></td>
<td>2.7.1</td>
</tr>
</tbody>
</table>
# Metrics

<table>
<thead>
<tr>
<th>Category</th>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Micro-architectural</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processor Performance</td>
<td>IPC</td>
<td>Instructions per cycle</td>
</tr>
<tr>
<td></td>
<td>MIPS</td>
<td>Million instructions per second</td>
</tr>
<tr>
<td>Instruction Mix</td>
<td>Instruction ratios</td>
<td>Ratios of load, store, branch, floating-point, and integer instructions</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>Branch Miss</td>
<td>Branch miss prediction ratio</td>
</tr>
<tr>
<td>Cache Behavior</td>
<td>L1I Hit Ratio</td>
<td>L1 instruction cache hit ratio</td>
</tr>
<tr>
<td></td>
<td>L1D Hit Ratio</td>
<td>L1 data cache hit ratio</td>
</tr>
<tr>
<td></td>
<td>L2 Hit Ratio</td>
<td>L2 cache hit ratio</td>
</tr>
<tr>
<td></td>
<td>L3 Hit Ratio</td>
<td>L3 cache hit ratio</td>
</tr>
<tr>
<td>System Metrics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>Read Bandwidth</td>
<td>Memory load bandwidth</td>
</tr>
<tr>
<td></td>
<td>Write Bandwidth</td>
<td>Memory store bandwidth</td>
</tr>
<tr>
<td></td>
<td>Total Bandwidth</td>
<td>memory load and store bandwidth</td>
</tr>
<tr>
<td>Disk I/O Behavior</td>
<td>Disk I/O Bandwidth</td>
<td>Disk read and write bandwidth</td>
</tr>
</tbody>
</table>
## Runtime Speedup on Xeon E5645

- **136X** for TeraSort, **743X** for K-means, **160X** for PageRank
- **155X** for AlexNet, **376X** for Inception-V3

<table>
<thead>
<tr>
<th>Workloads</th>
<th>Execution Time (Second)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Real version</td>
</tr>
<tr>
<td>TeraSort</td>
<td>1500</td>
</tr>
<tr>
<td>K-means</td>
<td>5971</td>
</tr>
<tr>
<td>PageRank</td>
<td>1444</td>
</tr>
<tr>
<td>AlexNet</td>
<td>1556</td>
</tr>
<tr>
<td>Inception-V3</td>
<td>6782</td>
</tr>
</tbody>
</table>
Performance Accuracy

- The accuracy of all selected metrics

\[
\text{Accuracy}(Val_R, Val_P) = 1 - \left| \frac{Val_P - Val_R}{Val_R} \right|
\]

- Val\(_R\) – average value of real benchmark on all slaves
- Val\(_P\) – average value of corresponding proxy benchmark
Proxy Benchmark Accuracy

- System and Micro-architectural Data Accuracy
  - The average accuracy of all metrics are greater than 90%
Instruction Mix Breakdown

Proxy benchmark preserve the instruction mix characteristics

- Floating-Point
- Load
- Store
- Branch
- Integer
Disk I/O Behaviors

- Disk I/O bandwidth

\[ BW_{DiskI/O} = \frac{(Sector_{Read} + Sector_{Write}) \times Size_{Sector}}{RunTime} \]

- Disk Bandwidth (MB/s) Graph

  - TeraSort
  - Kmeans
  - PageRank
  - AlexNet
  - Inception V3

  - Real
  - Proxy
Overview

- Simulation of Big Data and AI Workloads
  - Challenges & Motivation
- Data Motif-based Proxy Benchmarks
- Evaluation on X86 Processor
- Case Study
- How to Use
Case 1—Data input

- Can proxy benchmark reflect the impact of input data?
  - Data type and access pattern have a great impact
  - Data sparsity impact on memory bandwidth for Hadoop Kmeans
    - Dense vector: all elements are none-zero
    - Sparse vector: 90% elements are zero
Case 1—Data Input (Cont’)

- Using the same proxy benchmark for K-means
- Drive it by two different sparsity data sets

The accuracy is not affected by the input data
Case 2---Configuration Adaptability

- Is proxy benchmark adaptable to cluster configuration?
  - Dynamic resource requirements in data center
    • Memory size alteration
    • Hardware configuration alteration
    • Cluster scale alteration

- A new cluster configuration
  - Three-node cluster with the same E5645 processor
  - Memory configuration changes to 64 GB
Runtime Speedup on New Cluster

- Run the same proxy benchmarks and original workloads on the new cluster
  - 170X for TeraSort, 509X for K-means, 120X for PageRank
  - 121X for AlexNet, 307X for Inception-V3

<table>
<thead>
<tr>
<th>Workloads</th>
<th>Execution Time (Second)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Real version</td>
<td>Proxy version</td>
<td></td>
</tr>
<tr>
<td>TeraSort</td>
<td>2721</td>
<td>16.04</td>
<td></td>
</tr>
<tr>
<td>K-means</td>
<td>7143</td>
<td>14.03</td>
<td></td>
</tr>
<tr>
<td>PageRank</td>
<td>1693</td>
<td>14.07</td>
<td></td>
</tr>
<tr>
<td>AlexNet</td>
<td>1333</td>
<td>11.03</td>
<td></td>
</tr>
<tr>
<td>Inception-V3</td>
<td>5839</td>
<td>19.04</td>
<td></td>
</tr>
</tbody>
</table>
Accuracy on New Cluster

- On the new cluster, the average accuracy of all metrics are also *greater than 90%*
Case 3---Cross Architecture

- Can proxy benchmark reflect the relative performance among different architectures?

- A three-node cluster with Haswell processors
  - Xeon E5-2620 v3 (Haswell)
  - Memory 64 GB
Consistent Performance Trend

- Intel processors from different generations
  - Westmere v.s. Haswell
- Runtime speedup behavior of original workloads and proxy benchmarks

\[
\text{Speedup}(\text{Time}_{\text{Westmere}}, \text{Time}_{\text{Haswell}}) = \frac{\text{Time}_{\text{Westmere}}}{\text{Time}_{\text{Haswell}}}
\]
Speedup across Westmere and Haswell

- Proxy benchmarks reflect consistent speedup trends with original big data and AI workloads
  - Hadoop TeraSort: 2722 seconds on Westmere and 1723 seconds on Haswell
  - Proxy TeraSort: 16.1 seconds on Westmere and 10 seconds on Haswell
Case 4---ARMv8 Processor

- Two-node cluster

- Workloads
  - Hadoop TeraSort: 50GB records
  - Hadoop K-means: 50GB vector data
  - Hadoop PageRank: $2^{24}$-vertex graph data
ARM Configuration

- 32 Physical cores
  - Separate L1 data cache and L1 instruction cache
  - Shared L2 cache every four cores, Shared L3 cache for all cores

<table>
<thead>
<tr>
<th>Hardware Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
</tr>
<tr>
<td>Number of Processors</td>
</tr>
<tr>
<td>Number of Cores</td>
</tr>
<tr>
<td>Frequency</td>
</tr>
<tr>
<td>L1 Cache(I/D)</td>
</tr>
<tr>
<td>L2 Cache</td>
</tr>
<tr>
<td>L3 Cache</td>
</tr>
<tr>
<td>Architecture</td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>Ethernet</td>
</tr>
<tr>
<td>Hyper-Threading</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
</tr>
<tr>
<td>Linux Kernel</td>
</tr>
<tr>
<td>GCC Version</td>
</tr>
<tr>
<td>JDK Version</td>
</tr>
<tr>
<td>Hadoop Version</td>
</tr>
</tbody>
</table>
### Runtime Speedup on ARMv8

- **336X** for TeraSort, **386X** for Kmeans, **690X** for PageRank

<table>
<thead>
<tr>
<th>Workloads</th>
<th>Execution Time (Second)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hadoop version</td>
</tr>
<tr>
<td>TeraSort</td>
<td>1378</td>
</tr>
<tr>
<td>Kmeans</td>
<td>3347</td>
</tr>
<tr>
<td>PageRank</td>
<td>4291</td>
</tr>
</tbody>
</table>
Accuracy on ARMv8

- System and Micro-architectural Data Accuracy
  - 93%, 95%, 92% for TeraSort, K-means and PageRank on average
Multi-core Scalability on ARMv8

- Varied CPU cores
  - CPU-Hotplug mechanism: 4, 8, 16, 32 cores
  - Hadoop benchmarks
    - Adjust the Hadoop configurations to get peak performance
  - Proxy benchmarks
    - Run directly without any modification
Multi-core Scalability

- Runtime and MIPS
  - Similar multi-core scalability trends
## Speedup across X86_64 and ARM

<table>
<thead>
<tr>
<th>Hardware Configurations</th>
<th>ARMv8</th>
<th>Xeon E5-2690 V3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>ARMv8</td>
<td>Xeon E5-2690 V3</td>
</tr>
<tr>
<td>Number of Processors</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Number of Cores</td>
<td>32</td>
<td>12</td>
</tr>
<tr>
<td>Frequency</td>
<td>2.1GHz</td>
<td>2.6GHz</td>
</tr>
<tr>
<td>L1 Cache(I/D)</td>
<td>48KB/32KB</td>
<td>32KB/32KB</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>8 x 1024KB</td>
<td>12 x 256KB</td>
</tr>
<tr>
<td>L3 Cache</td>
<td>32MB</td>
<td>30MB</td>
</tr>
<tr>
<td>Architecture</td>
<td>ARM</td>
<td>X86_64</td>
</tr>
<tr>
<td>Memory</td>
<td>64GB, DDR4</td>
<td>64GB, DDR4</td>
</tr>
<tr>
<td>Ethernet</td>
<td>1Gb</td>
<td>1Gb</td>
</tr>
<tr>
<td>Hyper-Threading</td>
<td>None</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software Configurations</th>
<th>EulerOS V2.0</th>
<th>Red-hat Enterprise Linux Server release 7.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux Kernel</td>
<td>4.1.23-vhulk3.6.3.aarch64</td>
<td>3.10.0-123.e17.x86-64</td>
</tr>
<tr>
<td>GCC Version</td>
<td>4.9.3</td>
<td>4.8.2</td>
</tr>
<tr>
<td>JDK Version</td>
<td>jdk1.8.0_101</td>
<td>jdk1.7.0_79</td>
</tr>
<tr>
<td>Hadoop Version</td>
<td>2.5.2</td>
<td>2.5.2</td>
</tr>
</tbody>
</table>
Speedup Computation

**Equation**

\[
\text{Speedup}(\text{Time}_{X86\_64}, \text{Time}_{ARM}) = \frac{\text{Time}_{ARM}}{\text{Time}_{X86\_64}}
\]

**Configuration**

- Hadoop benchmarks
  - Optimized Hadoop configuration according to hardware environments
- Proxy benchmarks
  - The same version on X86_64 and ARMv8
Runtime Speedup across X86_64 and ARM

- Consistent speedup trends
  - Hadoop TeraSort (1.61X runtime speedup)
    - 1378 seconds on ARMv8
    - 856 seconds on Haswell
  - Proxy TeraSort (1.60X runtime speedup)
    - 4.1 seconds on ARMv8
    - 2.56 seconds on Haswell
Overview

- Simulation of Big Data and AI Workloads
  - Challenges & Motivation
- Data Motif-based Proxy Benchmarks
- Evaluation on X86 Processor
- Case Study
- How to Use
How to Use

- We provide Proxy_Benchmark.tar.gz package
  - tar -xf Proxy_Benchmark.tar.gz
  - cd Proxy_Benchmark
    - Containing five proxy benchmarks
      - TeraSort, PageRank, Kmeans, AlexNet, Inception-V3

- Compile
  - make all

- Running scripts
  - ./run-proxy-xxx.sh (e.g., terasort)
Deploy on GEM5 FS Mode

1) Use cross compilation tool
   - E.g., cross compilation for ARM: arm-linux-gcc-4.4.3.tar

2) mount Gem5 image
   - E.g., mount -o,loop,offset=32256 /disk_path/disks/aarch32-ubuntu-natty-headless.img /mount_path

3) copy the Proxy_Benchmark into the Gem5 image
   - cp –r Proxy_Benchmark /mount_path

4) unmount the image
   - umount /mount_path
Deploy on GEM5 FS Mode (cont’)

Start GEM5

- E.g., ./build/ARM/gem5.opt ./configs/example/fs.py --disk-image=/disk_path/disks/aarch32-ubuntu-natty-headless.img
- $GEM5_HOME/m5term localhost 3456

Running the scripts in the GEM5

- ./run-proxy-xxx.sh (e.g., terasort)
Conclusion

- Proxy benchmarks have been applied to chip design
  - A data motif-based proxy benchmark generating methodology

- The website:
  - http://prof.ict.ac.cn/download.html
Thank You!