Micro Benchmarks

Wanling Gao

ICT, Chinese Academy of Sciences

HPCA 2019, Washington D.C., USA
A scalable big data and AI benchmark suite

- Treat big data, AI and Internet service workloads as a pipeline of units of computation handling (input or intermediate) data

- Target: find the main abstractions of time-consuming units of computation (data motifs)
  - The combination of data motifs = complex workloads
    - Similar to Relational Algebra
  - Data motifs-based scalable benchmarking methodology

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.**
Micro Benchmark Target

- Capture one class of unit of computation in big data and AI

- Easily be ported to a new computer system or architecture at an earlier stage
Outline

- Summary of Micro Benchmark
- Micro Benchmark Characterization
- Conclusion
Summary

- 27 micro benchmarks
  - Covering 6 workload types
    - Offline analytics, Graph analytics
    - Streaming, NoSQL, Data warehouse
    - AI
  - Covering 8 data motifs
    - Transform, Graph, Set, Sort, Matrix, Logic, Sampling, Basic statistics
  - Covering 5 application domains
    - Internet Service (Social network, Search engine, E-commerce)
    - Recognition Science
    - Medical Science
# Micro Benchmarks

<table>
<thead>
<tr>
<th>Micro Benchmark</th>
<th>Involved Motif</th>
<th>Application Domain</th>
<th>Workload Type</th>
<th>Data Set</th>
<th>Software Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort</td>
<td>Sort</td>
<td>SE, SN, EC, MP, BI</td>
<td>Offline analytics</td>
<td>Wikipedia entries</td>
<td>Hadoop, Spark, Flink, MPI</td>
</tr>
<tr>
<td>Grep</td>
<td>Set</td>
<td></td>
<td>Offline analytics</td>
<td>Wikipedia entries</td>
<td>Hadoop, Spark, Flink, MPI</td>
</tr>
<tr>
<td>WordCount</td>
<td>Basic statistics</td>
<td></td>
<td>Streaming</td>
<td>Random Generate</td>
<td>Spark streaming</td>
</tr>
<tr>
<td>MD5</td>
<td>Logic</td>
<td></td>
<td>Offline analytics</td>
<td>Wikipedia entries</td>
<td>Hadoop, Spark, MPI</td>
</tr>
<tr>
<td>Connected Component</td>
<td>Graph</td>
<td>SN</td>
<td>Graph analytics</td>
<td>Facebook social network</td>
<td>Hadoop, Spark, Flink, GraphLab, MPI</td>
</tr>
<tr>
<td>RandSample</td>
<td>Sampling</td>
<td>SE, MP, BI</td>
<td>Offline analytics</td>
<td>Wikipedia entries</td>
<td>Hadoop, Spark, MPI</td>
</tr>
<tr>
<td>FFT</td>
<td>Transform</td>
<td>MP</td>
<td>Offline analytics</td>
<td>Two-dimensional matrix</td>
<td>Hadoop, Spark, MPI</td>
</tr>
<tr>
<td>Matrix Multiply</td>
<td>Matrix</td>
<td>SE, SN, EC, MP, BI</td>
<td>Offline analytics</td>
<td>Two-dimensional matrix</td>
<td>Hadoop, Spark, MPI</td>
</tr>
<tr>
<td>Read</td>
<td>Set</td>
<td>SE, SN, EC</td>
<td>NoSQL</td>
<td>ProfSearch resumes</td>
<td>HBase, MongoDB</td>
</tr>
<tr>
<td>Write</td>
<td>Set</td>
<td>SE, SN, EC</td>
<td>NoSQL</td>
<td>ProfSearch resumes</td>
<td>HBase, MongoDB</td>
</tr>
<tr>
<td>Scan</td>
<td>Set</td>
<td>SE, SN, EC</td>
<td>NoSQL</td>
<td>ProfSearch resumes</td>
<td>HBase, MongoDB</td>
</tr>
<tr>
<td>OrderBy</td>
<td>Set</td>
<td>EC</td>
<td>Data warehouse</td>
<td>E-commerce transaction</td>
<td>Hive, Spark-SQL, Impala</td>
</tr>
<tr>
<td>Aggregation</td>
<td>Set</td>
<td>EC</td>
<td>Data warehouse</td>
<td>E-commerce transaction</td>
<td>Hive, Spark-SQL, Impala</td>
</tr>
<tr>
<td>Project</td>
<td>Set</td>
<td>EC</td>
<td>Data warehouse</td>
<td>E-commerce transaction</td>
<td>Hive, Spark-SQL, Impala</td>
</tr>
<tr>
<td>Filter</td>
<td>Set</td>
<td>EC</td>
<td>Data warehouse</td>
<td>E-commerce transaction</td>
<td>Hive, Spark-SQL, Impala</td>
</tr>
<tr>
<td>Select</td>
<td>Set</td>
<td>EC</td>
<td>Data warehouse</td>
<td>E-commerce transaction</td>
<td>Hive, Spark-SQL, Impala</td>
</tr>
<tr>
<td>Union</td>
<td>Set</td>
<td>EC</td>
<td>Data warehouse</td>
<td>E-commerce transaction</td>
<td>Hive, Spark-SQL, Impala</td>
</tr>
<tr>
<td>Convolution</td>
<td>Transform</td>
<td>SN, EC, MP, BI</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Caffe, PyTorch, Pthread</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>Matrix</td>
<td>SN, EC, MP, BI</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Caffe, PyTorch, Pthread</td>
</tr>
<tr>
<td>Relu</td>
<td>Logic</td>
<td>SN, EC, MP, BI</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Caffe, PyTorch, Pthread</td>
</tr>
<tr>
<td>Sigmoid / Tanh</td>
<td>Matrix</td>
<td>SN, EC, MP, BI</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Caffe, PyTorch, Pthread</td>
</tr>
<tr>
<td>MaxPooling</td>
<td>Sampling</td>
<td>SN, EC, MP, BI</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Caffe, PyTorch, Pthread</td>
</tr>
<tr>
<td>AvgPooling</td>
<td>Sampling</td>
<td>SN, EC, MP, BI</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Caffe, PyTorch, Pthread</td>
</tr>
<tr>
<td>CosineNorm</td>
<td>Basic Statistics</td>
<td>SN, EC, MP, BI</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Caffe, PyTorch, Pthread</td>
</tr>
<tr>
<td>BatchNorm</td>
<td>Basic Statistics</td>
<td>SN, EC, MP, BI</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Caffe, PyTorch, Pthread</td>
</tr>
<tr>
<td>Dropout</td>
<td>Sampling</td>
<td>SN, EC, MP, BI</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Caffe, PyTorch, Pthread</td>
</tr>
</tbody>
</table>
Sort

- Sort the key value according to a certain order

- Data input
  - Wikipedia entries

- Software stacks
  - Hadoop, Spark, Flink, MPI
Grep

- Extract matching strings from text files and counts how many time they occurred

- Data input
  - Wikipedia entries

- Software stacks
  - Hadoop, Spark, Flink, MPI
WordCount

- Count the number of words in a document

- Data input
  - Wikipedia entries

- Software stacks
  - Hadoop, Spark, Flink, MPI
MD5

- A widely used hash function producing a 128-bit hash value
  - The input message is broken up into chunks of 512-bit blocks
    \[
    \begin{align*}
    F(B, C, D) &= (B \land C) \lor (\neg B \land D) \\
    G(B, C, D) &= (B \land D) \lor (C \land \neg D) \\
    H(B, C, D) &= B \oplus C \oplus D \\
    I(B, C, D) &= C \oplus (B \lor \neg D)
    \end{align*}
    \]

- Data input
  - Wikipedia entries

- Software stacks
  - Hadoop, Spark, MPI
Connected Component

- A subgraph in which any two vertices are connected to each other by paths
  - Easily computed in linear time using either breadth-first search or depth-first search

- Data input
  - Facebook social network

- Software stacks
  - Hadoop, Spark, Flink, GraphLab, MPI
RandSample

- Select a subset samples randomly
  - Using a random data generator to determine whether the data is selected or not

- Data input
  - Wikipedia entries

- Software stacks
  - Hadoop, Spark, MPI
FFT

- Cooley–Tukey algorithm
  - radix-2 decimation-in-time (DIT) FFT
    \[ X_k = E_k + e^{-\frac{2\pi i}{N} k} O_k \]
    \[ X_{k+N/2} = E_k - e^{-\frac{2\pi i}{N} k} O_k \]

- Data input
  - Two-dimensional matrix

- Software stacks
  - Hadoop, Spark, MPI
Matrix Multiply

- Compute a matrix from two matrices
  \[ c_{ij} = a_{i1}b_{1j} + \cdots + a_{im}b_{mj} = \sum_{k=1}^{m} a_{ik}b_{kj} \]

- Data input
  - Two-dimensional matrix

- Software stacks
  - Hadoop, Spark, MPI
NoSQL --- Read, Write, Scan

- **Benchmarks**
  - Read records randomly
  - Write new records
  - Scan records in order

- **Data input**
  - ProfSearch resumes
    - a semi-structured data set from a vertical search engine for scientists

- **Software stacks**
  - Hbase, MongoDB
**OrderBy**

- Order the data according to specific item
  
  ```sql
  create table tmp36 as SELECT * FROM bigdatabench_dw_item order by ITEM_ID;
  ```

- Data input
  - E-commerce transaction
- Software stacks
  - Hive, Spark-SQL, Impala
Aggregation

- Gather information and aggregate in a summary form
  
  ```sql
  create table tmp27 as select GOODS_ID, sum(GOODS_NUMBER) from bigdatabench_dw_item group by GOODS_ID;
  ```

- Data input
  - E-commerce transaction

- Software stacks
  - Hive, Spark-SQL, Impala
Project

- Retrieve specified attributes (columns)

```sql
create table tmp37 as SELECT ORDER_ID FROM bigdatabench_dw_order;
```

- Data input
  - E-commerce transaction

- Software stacks
  - Hive, Spark-SQL, Impala
Filter

- Select partial records that match certain criteria

```sql
create table tmp35 as select * from bigdatabench_dw_item where GOODS_AMOUNT > 750000;
```

- Data input
  - E-commerce transaction

- Software stacks
  - Hive, Spark-SQL, Impala
Select

- Select a set of records from one or more tables

```
create table tmp26 as select GOODSPRICE, GOODS_AMOUNT from bigdatabench_dw_item where GOODS_AMOUNT > 224000;
```

- Data input
  - E-commerce transaction

- Software stacks
  - Hive, Spark-SQL, Impala
Union

- Combine the result of two or more SELECT statements

```sql
create table tmp38 as select * from
  (select * from bigdatabench_dw_item where GOODS_AMOUNT > 750000
  union all
  select * from bigdatabench_dw_item where GOODS_AMOUNT < 5
 ) temp;
```

- Data input
  - E-commerce transaction

- Software stacks
  - Hive, Spark-SQL, Impala
Convolution

- The general expression

\[ g(x, y) = (\omega * f)(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x - s, y - t) \]

Note: \( g(x,y) \) is the filtered image, \( f(x,y) \) is the original image, \( \omega \) is the filter kernel

- Data input
  - Image dataset---Cifar, ImageNet
  - Convolution kernel

- Software stacks
  - TensorFlow, Caffe2, PyTorch, Pthread
Fully Connected

- Have connections to all neurons in the previous layer
  - Matrix multiplication followed by a bias offset
    \[ H(X) = (x. W^t) + b \]

- Data input
  - Image dataset---Cifar, ImageNet

- Software stacks
  - TensorFlow, Caffe2, PyTorch, Pthread
Relu

- Abbreviation of rectified linear unit
  - Is defined as the positive part of its argument
    \[ f(x) = \max(0, x) \quad x \text{ is the input to a neuron} \]

- Data input
  - Image dataset---Cifar, ImageNet

- Software stacks
  - TensorFlow, Caffe2, PyTorch, Pthread
Sigmoid

- Sigmoid activation function

\[ S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1} \]

- Data input
  - Image dataset---Cifar, ImageNet

- Software stacks
  - TensorFlow, Caffe2, PyTorch, Pthread
Tanh

- Tanh activation function

\[ f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \]

- Data input
  - Image dataset---Cifar, ImageNet

- Software stacks
  - TensorFlow, Caffe2, PyTorch, Pthread
MaxPooling

- Non-linear down-sampling
  - Dividing the input image into a set of non-overlapping rectangles
  - Outputs the *maximum* for each sub-rectangle

- Data input
  - Image dataset---Cifar, ImageNet

- Software stacks
  - TensorFlow, Caffe2, PyTorch, Pthread
**AvgPooling**

- Non-linear down-sampling
  - Dividing the input image into a set of non-overlapping rectangles
  - Outputs the *average* value for each sub-rectangle

- Data input
  - Image dataset---Cifar, ImageNet

- Software stacks
  - TensorFlow, Caffe2, PyTorch, Pthread
Batch Normalization

- A normalization method/layer for neural networks
  - For a layer with d-dimensional input \( x = (x(1) \ldots x(d)) \)

\[
\hat{x}(k) = \frac{x(k) - E[x(k)]}{\sqrt{\text{Var}[x(k)]}}
\]

- Data input
  - Image dataset---Cifar, ImageNet

- Software stacks
  - TensorFlow, Caffe2, PyTorch, Pthread

Cosine Normalization

■ Using Cosine Similarity in Neural Networks
  ■ Instead of Dot Product
    \[ o = f(\text{net}_{\text{norm}}) = f(\cos \theta) = f\left(\frac{\vec{w} \cdot \vec{x}}{|\vec{w}| |\vec{x}|}\right) \]
    where \text{net}_{\text{norm}} is the normalized pre-activation, \vec{w} is the incoming weight vector and \vec{x} is the input vector; (\cdot) indicates dot product, \( f \) is nonlinear activation function

■ Data input
  ■ Image dataset---Cifar, ImageNet

■ Software stacks
  ■ TensorFlow, Caffe2, PyTorch, Pthread

Dropout

- A regularization technique for reducing overfitting in neural networks

- Data input
  - Image dataset---Cifar, ImageNet

- Software stacks
  - TensorFlow, Caffe2, PyTorch, Pthread
Software Stacks

AI
- TensorFlow
- Caffe2
- PyTorch
- Pthread

Offline Analytics
- Hadoop
- Spark
- Flink
- MPI

Graph Analytics
- Hadoop
- Spark
- Flink
- GraphLab
- MPI

Streaming
- Spark streaming
- JStorm

Data Warehouse
- Hive
- SparkSQL
- Impala

NoSQL
- HBase
- MongoDB
Outline

- Summary of Micro Benchmark

- Micro Benchmark Characterization

- Conclusion
Experiment Setups

- Three-node cluster

<table>
<thead>
<tr>
<th>Hardware Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td><strong>CPU Type</strong></td>
</tr>
<tr>
<td>Intel ®Xeon E5-2620 V3</td>
</tr>
<tr>
<td><strong>Intel CPU Core</strong></td>
</tr>
<tr>
<td>12 <a href="mailto:cores@2.40G">cores@2.40G</a></td>
</tr>
<tr>
<td><strong>L1 DCache</strong></td>
</tr>
<tr>
<td>12 × 32 KB</td>
</tr>
<tr>
<td><strong>L1 ICache</strong></td>
</tr>
<tr>
<td>12 × 32 KB</td>
</tr>
<tr>
<td><strong>L2 Cache</strong></td>
</tr>
<tr>
<td>12 × 256 KB</td>
</tr>
<tr>
<td><strong>L3 Cache</strong></td>
</tr>
<tr>
<td>15MB</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
</tr>
<tr>
<td>64GB,DDR4</td>
</tr>
<tr>
<td><strong>Disk</strong></td>
</tr>
<tr>
<td>SATA@7200RPM</td>
</tr>
<tr>
<td><strong>Ethernet</strong></td>
</tr>
<tr>
<td>1Gb</td>
</tr>
<tr>
<td><strong>Hyper-Threading</strong></td>
</tr>
<tr>
<td>Disabled</td>
</tr>
</tbody>
</table>
Data Configuration

- To fully utilize the memory resources
  - Big data micro benchmarks
    - 100 GB text data
    - $2^{26}$-vertex graph data
    - 65536 two-dimension matrix data
  - AI micro benchmarks
    - Input dimension 224*224, channels 64
    - 100K images from ImageNet
System Behaviors

- **CPU Utilization & I/O Wait**
  - Hadoop have higher CPU utilization and less I/O wait than Spark
  - AI micro benchmarks have lower I/O wait than big data
  - Some of AI micro benchmarks are CPU intensive
  - Pthread benchmarks have less CPU utilization and I/O Wait in general

![Chart showing CPU Utilization and I/O Wait for various benchmarks](chart.png)
I/O Behaviors

- Disk I/O Bandwidth & Network I/O Bandwidth
  - Spark stack has much larger network I/O pressure than that of Hadoop stack
    - More data shuffles, so it needs transferring data from one node to another one frequently
Execution Performance

- The overall running efficiency of the workloads
  - Instruction level parallelism (ILP)
    - Retired instructions per cycle (IPC)
  - Memory level parallelism (MLP)
    - Dividing L1D_PEND_MISS.PENDING by L1D_PEND_MISS.PENDING_CYCLES
Execution Performance

- ILP & MLP

- Cover a wide range of ILP and MLP behaviors
  - Distinct computation and memory access patterns

- software stack changes computation and memory access patterns
  - Hadoop FFT v.s. Spark FFT
Top-Down Method

Issue point as the dividing point

Whether the micro operation is retired?

Only retiring is “useful work”

Not ready with more uops

From “A Top-Down Method for Performance Analysis and Counters Architecture”
Pipeline Efficiency

**Top-Down Methodology**

- Retiring, Frontend bound, Backend bound, Bad speculation
  - Hadoop: notable stalls due to frontend bound and bad speculation
  - Spark: Higher backend bound
  - AI reflects different bottlenecks
Frontend Bound

- Frontend latency bound > Frontend bandwidth bound
  - Latency bound: notable stalls due to frontend bound and bad speculation
  - Bandwidth bound: delivering insufficient uops comparing to the theoretical value
Data Motif – Frontend Bound

- Frontend Bound Breakdown
  - Top 3: branch reesteers, instruction cache miss, MS switch
    - The first reason is the delays to obtain the correct instructions
    - MS switch: big data and AI systems use many CISC instructions that cannot be decoded by default decoder
Data Motif – Backend Bound

- Memory bound (data movement delays) > Core bound
  - Memory bound: L1, L2, L3, external memory bound
  - Core bound: the lack of hardware resources or port under-utilization
Overview

■ Looking back at history
■ What is Data Motif
■ Characterization of Data Motif
■ Impact of Data Input
■ Conclusion
Impact of Data Input

Size

Pattern

Type & Source
Similarity Analysis

- Three data configurations
  - Small, Medium, Large

- Sixty metrics spanning system and micro-architecture

- Measuring Similarity
  - PCA
  - Hierarchical clustering
I/O Bandwidth

Using the I/O bandwidth of Small data size as baseline, we normalize the I/O bandwidth of Medium and Large data size.
Size Impact on Pipeline Behavior

- Data size increases $\rightarrow$ frontend bound decrease, backend bound increase
Impact of Data Input

Size  Pattern  Type & Source
Impact of Data Pattern

- Dense matrix V.S. Sparse matrix
  - I/O Bandwidth: Sparse < Dense
  - Frontend Stalls: Sparse > Dense

(a) System Behavior with Different Patterns.

(b) Micro-architecture Behavior with Different Patterns.
Impact of Data Input

Size, Pattern, Type & Source
Impact of Data Type and Source

- Un-structured text data & Semi-structured sequence data
  - System: 1.12-7.29 differences
  - Architecture: text format incurs more backend bound

(a) System Behavior with Different Types.

(b) Micro-architecture Behavior with Different Types.
Outline

- Summary of Micro Benchmark
- Micro Benchmark Characterization
- Conclusion
Conclusion

Website:
- http://www.benchcouncil.org/benchmarks.html
- http://www.benchcouncil/BigDataBench
- http://prof.ict.ac.cn/BigDataBench

Micro benchmark
- Single data motif implementation