Accelerating Large Scale Deep Learning Inference through DeepCPU at Microsoft

2019 USENIX Conference on Operational Machine Learning

Minjia Zhang, Samyam Rajbandari, Wenhan Wang, Elton Zheng, Olatunji Ruwase, Jeff Rasley, Jason Li, Junhua Wang, Yuxiong He

Microsoft AI and Research
Highlights

- **DeepCPU**, the fastest deep learning serving library for recurrent neural networks (RNNs) on CPUs
- SLT (Scenario, Library, Technique) driven methodology
- **10x lower latency and cost** than existing framework
- Ship DL models with great latency/cost reduction in Microsoft
Deep Learning Serving Challenges

• Long serving latency blocks deployment
• Support advance models while meeting latency SLA and saving cost

<table>
<thead>
<tr>
<th>DL Scenarios</th>
<th>Original Latency</th>
<th>Latency Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRC Model A</td>
<td>~100ms</td>
<td>&lt; 10ms</td>
</tr>
<tr>
<td>MRC Model B</td>
<td>~107ms</td>
<td>&lt; 10ms</td>
</tr>
<tr>
<td>Ranking Model</td>
<td>10ms for [query, 1 passage] x 150 passages</td>
<td>&lt; 5ms</td>
</tr>
<tr>
<td>Query rewriting</td>
<td>~51ms</td>
<td>&lt; 5ms</td>
</tr>
</tbody>
</table>
Methodology

• Improve existing general-purpose DL frameworks? 🚨
• Customized optimization with effective reuse ✔
• Co-development of Scenario, Library, and Technique (SLT)

• Scenario
  • Apply customized optimization, striking for best performance
  • Think out of box, not limited by existing framework

• Library
  • Collection of generic building blocks that speed up customized optimization
  • Framework independent -- can benefit multiple DL frameworks

• Technique
  • One technique could benefit multiple library components and many scenarios
  • Parallelism, scheduling, and locality optimization on CPU at no cost in accuracy
Outline

• Real-World Scenarios with DeepCPU-Powered RNN-Based Models

• Library Features

• Optimization Techniques

• How is DeepCPU Utilized?
Scenario 1: Question Answering

Bidirectional Attention Flow Model (BiDAF)
1. Tensorflow Serving latency: 107ms (non-shippable)
2. Target latency: < 10ms
RNN Performance Bottleneck

What is Atom?

Step 1: Limited Parallelism (small batch size)
Step 2: Poor Data Locality
Step 3: Poor Scalability and Performance due to reading data from slow memory

Performance Critical Factors | Implications
--- | ---
Limited Parallelism (small batch size) | Poor Scalability
Poor Data Locality | Poor Scalability and Performance due to reading data from slow memory

L3 Cache

L2 Cache

L2 Cache

Core 0

Core 1
Optimization Results

Bidirectional Attention Flow Model (BiDAF)
1. Tensorflow Serving latency: 107ms (non-shippable)
2. Target latency: < 10ms

Our Optimization
DeepCPU implementation for BiDAF

Same accuracy
Latency: 107ms to 4.1ms (>20 times speedup)
Non-shippable -> Shippable
Scenario 2: Text Similarity Ranking

• Generate text similarities using deep learning model
• Model: word embedding + encoding with GRUs + conv + max-pool
• Latency SLA: 5ms for <query, top 150 passages>
• Tensorflow serving latency
  • single <query, passage> pair: 10ms
  • <query, 150 passages>: fan-out to 150 machines

• Our optimizations
  • <query, 150 passages>: 5ms, one machine (>100x throughput gain)
  • Reduce thousands of machines and millions of infrastructure costs
## Optimization Results

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Original Latency</th>
<th>Latency Target</th>
<th>Optimized Latency</th>
<th>Latency reduction</th>
<th>Throughput improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRC Model A</td>
<td>~100ms</td>
<td>10ms</td>
<td>9ms</td>
<td>&gt;10X</td>
<td>&gt;10X</td>
</tr>
<tr>
<td>MRC Model B</td>
<td>~107ms</td>
<td>10ms</td>
<td>4.1ms</td>
<td>&gt;20X</td>
<td>&gt;50X</td>
</tr>
<tr>
<td>Neural Ranking</td>
<td>10~12ms for [query, 1 doc] x 33 docs</td>
<td>6ms</td>
<td>1.5ms for [query, 1 doc]; &lt;6ms for [query, 33 docs]</td>
<td>&gt;6X</td>
<td>&gt;30X</td>
</tr>
<tr>
<td>Model A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neural Ranking</td>
<td>10ms for [query, 1 passage] x 150 passages</td>
<td>5ms</td>
<td>&lt;1ms for [query, 1 passage]; &lt;5ms for [query, 150 passages]</td>
<td>&gt;10X</td>
<td>&gt;100X</td>
</tr>
<tr>
<td>Model B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query rewriting</td>
<td>51ms</td>
<td>5ms</td>
<td>4ms</td>
<td>&gt;10X</td>
<td>&gt;3X</td>
</tr>
</tbody>
</table>
## Optimization Results Continued

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Original Latency</th>
<th>Latency Target</th>
<th>Optimized Latency</th>
<th>Latency reduction</th>
<th>Throughput improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoder Model A</td>
<td>~29ms</td>
<td>10ms</td>
<td>5.4ms</td>
<td>5X</td>
<td>5X</td>
</tr>
<tr>
<td>MRC Model C</td>
<td>~45ms for 1 [query, passage]</td>
<td>10ms</td>
<td>4.0ms for 1 [query, passage]; &lt;8.5ms for 20 [query, passage]</td>
<td>11X</td>
<td>&gt; 100X</td>
</tr>
<tr>
<td>Query tagging</td>
<td>9~16ms</td>
<td>3ms</td>
<td>0.95ms</td>
<td>10X</td>
<td>&gt; 10X</td>
</tr>
<tr>
<td>Encoder Model B</td>
<td>~25ms for [query, 1 title url]</td>
<td>7ms for a batch size of 33</td>
<td>5.4ms for [query, 33 title url];</td>
<td>10X</td>
<td>&gt; 100X</td>
</tr>
<tr>
<td>Classifier A</td>
<td>60ms</td>
<td>3ms</td>
<td>3ms</td>
<td>20X</td>
<td>20X</td>
</tr>
<tr>
<td>Classifier B</td>
<td>8ms</td>
<td>3ms</td>
<td>1ms</td>
<td>8X</td>
<td>8X</td>
</tr>
</tbody>
</table>

**Latency:** 5x – 20x faster, from impossible to ship to well fitting SLA  
**Capacity:** serving 5x – 20x bigger models under the same latency SLA  
**Throughput:** 5x – 100x higher  
**Cost:** reduced to 1% - 20% of original cost
DeepCPU: Fast DL Serving Library on CPUs

- **RNN family**
  - GRU cell and GRU sequence
  - LSTM cell and LSTM sequence
  - Bidirectional and stacked RNN networks

- **Fundamental building blocks and common DL Layers**
  - Matrix multiplication kernels, activation functions
  - high-way network, max pool layer, MLP layer ......

- **DL layers for MRC and conversation models**
  - Variety of attention layers
  - seq2seq decoding with beam search ......
## Optimization Techniques

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Our optimized library on CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix computation</td>
<td>Cache-aware matrix kernels + Intel MKL</td>
</tr>
<tr>
<td>Activation functions</td>
<td>Vectorization + parallelization</td>
</tr>
<tr>
<td>Operation Fusing</td>
<td>Fuse operations to reduce data read/write</td>
</tr>
<tr>
<td>Affinity</td>
<td>Bind app thread to hardware thread cross-socket awareness</td>
</tr>
<tr>
<td>Locality</td>
<td>Private-cache-aware partitioning + weight-centric streamlining</td>
</tr>
<tr>
<td>Parallelism</td>
<td>Judicious parallelism considering workload, parallelism efficiency and load balancing</td>
</tr>
<tr>
<td>Task Scheduling</td>
<td>Priority over critical path</td>
</tr>
<tr>
<td></td>
<td>Global optimization of DAG</td>
</tr>
</tbody>
</table>
How is DeepCPU Utilized?

- Customized Model Client
- Performance Hyperparameter Tuning
- Customized Serving Runtime
- Even Faster Latency

Customized Optimization

More Development Work

- DeepCPU Operators
- Replace Nodes in Model Graph
- Existing Framework Serving Engine
- Faster Latency

Framework Integration (TensorFlow, WinML, ONNX)

Less Development Work

Optimized Techniques

DeepCPU Library

Critical Scenario Owners

Framework Users
DeepCPU: Make DL Serving Faster & More Efficient

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Models</th>
<th>Usage</th>
<th>Impact</th>
</tr>
</thead>
</table>
| • Question Answering
  • Machine Reading Comprehension
  • Ranking
  • Query Rewriting
  • Query Tagging | • GRU/LSTM
  • Stacked RNN
  • Seq2Seq
  • Attention layers
  • Convolution
  • Highway network
  • MLP ...... | • Customized optimization
  • Framework integration | • 10x faster
  • 10x larger models
  • 10x - 100x more throughput
  • 10x - 100x less cost |
Thank You!

Questions?