DeepCPU: Serving RNN-based Deep Learning Models 10x Faster

Minjia Zhang*, Samyam Rajbhandari*, Wenhan Wang, Yuxiong He
(*Equal contribution)

Microsoft AI and Research
Highlights

• DeepCPU, the fastest deep learning serving library for recurrent neural networks (RNNs) on CPUs
• 10x lower latency and cost than TensorFlow and CNTK
• Empower CPU to beat GPU for RNN serving
• 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>Attention sum reader</td>
<td>~100ms</td>
<td>&lt; 10ms</td>
</tr>
<tr>
<td>Bidirectional attention flow model</td>
<td>~107ms</td>
<td>&lt; 10ms</td>
</tr>
<tr>
<td>Text similarity model</td>
<td>10ms for [query, 1 passage]  x 150 passages</td>
<td>&lt; 5ms</td>
</tr>
<tr>
<td>Seq2seq model</td>
<td>~51ms</td>
<td>&lt; 5ms</td>
</tr>
</tbody>
</table>
Outline

• Overview of Recurrent Neural Network (RNN)
• DeepCPU-Powered Real-World RNN-Based Models
• Library Features
• Performance Optimizations
• Performance Results
RNN Serving Performance Challenges

Language Modeling
Machine Translation
Machine Reading Comprehension
Speech Recognition
Conversation Bot
...

Limited Parallelism
- Small batch size
- Sequential dependency

Limited Bandwidth
- Vector-matrix multiplication
- Low data reuse
Case 1: Question Answering

**Direct answer**

**Good quality**
Model Complexity and Latency

Bidirectional Attention Flow Model (BiDAF)

1. Tensorflow Serving latency: 107ms (non-shippable)
2. Target latency: < 10ms
Optimization Results

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**
Case 2: Text Similarity Ranking

• Generate text similarities using deep model
• Model: word embedding + encoding with GRUs + conv + max-pool + scoring
• 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 to serve Bing traffic
DeepCPU: Fast DL Serving Library on CPUs

- **RNN family**
  - GRU cell and GRU sequence
  - LSTM cell and LSTM sequence
  - 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 ……
Performance Critical Factors | Implications
--- | ---
Limited Parallelism | Poor Scalability
Poor Data Locality | Poor Scalability and Performance due to reading data from slow memory

Deep Dive: RNN Performance Bottleneck

Step 1: What is W1?
Step 2: is W1 W1 W2
Step 3: Atom? W1 W2
Deep Dive: DeepCPU RNN Optimizations

1. MM-DAG
2. Valid Phased Schedule Generator
3. MM-Fusion
4. Reuse-Aware Parallelism Generator
5. Cache-Aware Partitioning
6. Weight-Centric Streamlining

3. **MM-fusion:**
   fuses smaller MMs into larger ones, improving data reuse and parallelism degree;

4. **Reuse-aware parallelism generator:**
   identifies best parallelism degree within and across MMs through auto-tuning, jointly considering locality;

5. **Private-cache-aware-partitioning (PCP):**
   optimizes data movement between shared L3 cache and private L2 cache with a novel and principled partitioning method;

6. **Weight centric streamlining (WCS):**
   maps the partitions produced by PCP to compute cores in a way that enables reuse of weights across the sequence.
3. MM-Fusion

1. MM-DAG
2. Valid Phased Schedule Generator
3. MM-Fusion
4. Reuse-Aware Parallelism Generator
5. Cache-Aware Partitioning
6. Weight-Centric Streamlining

Execute and get Execution Time

More Parallelism?
Yes

More Schedules?
Yes

Fastest Schedule

Step 1
What
W1

Step 2
is
W1

Step 3
Atom?
W1

Step 4
What is Atom?
W1

L3 Cache
L2 Cache
L1 Core 0
L1 Core 1

time
4. Resue-Aware Parallelism Generator

- 1. MM-DAG
- 2. Valid Phased Schedule Generator
- 3. MM-Fusion
- 4. Reuse-Aware Parallelism Generator
- 5. Cache-Aware Partitioning
- 6. Weight-Centric Streamlining

Execute and get Execution Time

- Yes
- More Parallelism?
- No
- More Schedules?
- Yes
- No
- Fastest Schedule
5. Cache-Aware Partitioning

**Theorem**  Consider $P$ cores on a CPU, and an MM $C[i,j] + = \sum_k A[i,k] \times B[k,j]$, where $|A| + |B| + |C| \leq |L3Cache|$ and $\min(|A|, |B|, |C|) + H + 1 \leq |L2Cache|$. $H$ is a constant upper bounded by $\max(I, J, K)$, where $I$, $J$ and $K$ are the sizes of indices $i$, $j$ and $k$. For a $P$-way partitioning $\langle X_i, X_j, X_k \rangle$ where $X_i \times X_j \times X_k = P$, a tight bound on the data movement between L3 and L2 cache is given by $X_j|A| + X_i|B| + 2X_k|C|$.
6. Weight-Centric Streamlining

1. MM-DAG
2. Valid Phased Schedule Generator
3. MM-Fusion
4. Reuse-Aware Parallelism Generator
5. Cache-Aware Partitioning
6. Weight-Centric Streamlining

Execute and get Execution Time

More Parallelism?
Yes
No

More Schedules?
Yes
No

Fastest Schedule

Input \times \text{Weight} = \text{Output}

Data Movement per Thread
- Input/2 + |weight| + |output|/2

Step 1
What is Atom? W1
Step 2
W2
Step 3
W2
Step 4
W2

Step 1
What is Atom? W1
Step 2
W2
Step 3
W2
Step 4
W2

18
Deep Dive: Summary

1. MM-DAG
2. Valid Phased Schedule Generator
3. MM-Fusion
4. Reuse-Aware Parallelism Generator
5. Cache-Aware Partitioning
6. Weight-Centric Streamlining

---

**Naïve Schedule**

- **Step 1**: What is Atom? W1
- **Step 2**: What? W1
- **Step 3**: Atom? W1

---

**Schedule Generator + MM-Fusion + Parallelism**

- **Step 1**: What is Atom? W1
- **Step 2**: What? W1
- **Step 3**: Atom? W1
- **Step 4**: W2

---

**+ Cache-Aware Partitioning + Weight-Centric Streamlining**

- **Step 1**: What is Atom? W1
- **Step 2**: What? W1
- **Step 3**: Atom? W1
- **Step 4**: W2
Performance : DeepCPU vs TF vs CNTK

• Average LSTM speedup
  • DeepCPU is 23x faster than Tensorflow
  • DeepCPU is 31x faster than CNTK

• Average GRU speedup
  • DeepCPU is 16x faster than Tensorflow
  • DeepCPU is 25x faster than CNTK

<table>
<thead>
<tr>
<th>Model parameters</th>
<th>LSTM speedup</th>
<th>GRU speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>input hidden seq. len.</td>
<td>TF</td>
<td>CNTK</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>100</td>
</tr>
<tr>
<td>256</td>
<td>64</td>
<td>100</td>
</tr>
<tr>
<td>1024</td>
<td>64</td>
<td>100</td>
</tr>
<tr>
<td>64</td>
<td>256</td>
<td>100</td>
</tr>
<tr>
<td>64</td>
<td>1024</td>
<td>100</td>
</tr>
<tr>
<td>1024</td>
<td>1024</td>
<td>100</td>
</tr>
<tr>
<td>256</td>
<td>256</td>
<td>1</td>
</tr>
<tr>
<td>256</td>
<td>256</td>
<td>10</td>
</tr>
<tr>
<td>256</td>
<td>256</td>
<td>100</td>
</tr>
</tbody>
</table>
DeepCPU vs GPU

Batch Size = 1, Sequence Length = 100

Batch Size = 1, Input/Hidden Dimension = 256
Summary of DeepCPU

- **DeepCPU**, the fastest deep learning serving library for recurrent neural networks (RNNs) on CPUs
- **10x lower latency and cost** than Tensorflow and CNTK
- Empower CPU to beat GPU for RNN serving
- Ship DL models in Microsoft with great latency/cost reduction
Thank You!

Questions?