Tutorial: Deep Learning Inference Optimizations for CPU

Guokai Ma
Intel
Notices & Disclaimers

- Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.
- Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.
- Your costs and results may vary.
- Intel technologies may require enabled hardware, software or service activation.
- © Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.
Agenda

- Deep learning inference system
- Deep learning inference optimizations
Deep learning inference system

Model implementation

- Model Checkpoint
- Optimization
  - Fusion
  - Low precision
  - Vectorize
- Optimized model
- SUT
  - Load balance
  - Warmup
  - Batching
  - Multi-instance
- loadgen
  - Query
  - Result

https://github.com/mlcommons/inference/tree/master/loadgen#integration-example-and-flow
Deep learning inference system

Model implementation

Model Checkpoint

Optimization

Fusion
Low precision
Vectorize

Optimized model

SUT

Load balance
Warmup
Batching
Multi-instance

Query

Result

loadgen

https://github.com/mlcommons/inference/tree/master/loadgen#integration-example-and-flow
Deep learning inference system

Model implementation

Optimization

- Fusion
- Low precision
- Vectorize
- 

SUT

- Load balance
- Warmup
- Batching
- Multi-instance

Result

Query

loadgen

https://github.com/mlcommons/inference/tree/master/loadgen#integration-example-and-flow
Deep learning inference system

Model implementation

- Model Checkpoint
- Optimization
  - Fusion
  - Low precision
  - Vectorize
  - ...
- Optimized model
- SUT
  - Load balance
  - Warmup
  - Batching
  - Multi-instance
- loadgen
  - Query
  - Result

https://github.com/mlcommons/inference/tree/master/loadgen#integration-example-and-flow
Deep learning inference system

- **Offline inference scenario – Throughput**
  - All samples are packed into a single query, throughput of SUT is measured

- **Server inference scenario – Throughput under latency bound**
  - Samples are sent from loadgen with random interval. SUT needs to ensure 99% of the samples are respond within latency bound
  - Throughput of SUT is measured
Deep learning inference optimizations

- **SUT**
  - Multi-instance
  - Warmup
  - Batching
    - Reduce padding
    - Dynamic batching
    - Constant batching

- **Framework/Model**
  - Op fusion
  - Low precision
  - Sparsity
  - Vectorize
Deep learning inference optimizations

- **SUT**
  - Multi-instance
  - Warmup
  - Batching
    - Reduce padding
    - Dynamic batching
    - Constant batching

- **Framework/Model**
  - Op fusion
  - Low precision
  - Sparsity
  - Vectorize
Multi-instance inference

Processor

L3 cache

Memory
Multi-instance inference

Processor

| Core | Core | Core | Core | Core | Core | Core | Core | Core | Core | Core | Core | Core | Core | Core | Core | Core | Core | Core | Core | Core | Core | Core | Core | Core |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| L2    | L2    | L2    | L2    | L2    | L2    | L2    | L2    | L2    | L2    | L2    | L2    | L2    | L2    | L2    | L2    | L2    | L2    | L2    | L2    | L2    | L2    | L2    | L2    |

L3 cache
Multi-instance inference

Processor

L3 cache

Core Core Core Core Core Core Core Core Core Core Core Core Core Core Core Core
L2 L2 L2 L2 L2 L2 L2 L2 L2 L2 L2 L2 L2 L2 L2 L2 L2
Multi-instance inference
Multi-instance inference

- Multi-instance inference allows us to control CPU computation resource in a fine grain manner
- Few cores per instance:
  - Reduce synchronization overhead
  - Increase parallelism for serial part of model
  - Ideal when activation can fully fit in L2 cache
- Instance must be bound to specific set of cores
  - numactl
  - taskset
  - OMP_NUM_THREADS
  - KMP_AFFINITY
Multi-instance inference

- Multi-instance inference allows us to control CPU computation resource in a fine grain manner
- Few cores per instance:
  - Reduce synchronization overhead
  - Increase parallelism for serial part of model
  - Ideal when activation can fully fit in L2 cache
- Instance must be bound to specific set of cores
  - `numactl`
  - `taskset`
  - `OMP_NUM_THREADS`
  - `KMP_AFFINITY`

<table>
<thead>
<tr>
<th>Workload</th>
<th>Cores per instance</th>
<th>Weight sharing (experimental)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLRM</td>
<td>7</td>
<td>Yes</td>
</tr>
<tr>
<td>RNN-T</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>BERT</td>
<td>8</td>
<td>No</td>
</tr>
<tr>
<td>MiniGo (selfplay)</td>
<td>1</td>
<td>No</td>
</tr>
</tbody>
</table>
Deep learning inference optimizations

- **SUT**
  - Multi-instance
  - **Warmup**
  - Batching
    - Reduce padding
    - Dynamic batching
    - Constant batching

- **Framework/Model**
  - Op fusion
  - Low precision
  - Sparsity
  - Vectorize
Warmup

- Allows primitives to be created at SUT initialization
- Workload with variable input size may need large primitive cache
  - BERT
  - DLRM
  - RNNT
- Need to ‘sweep’ input size from smallest to largest input size
Deep learning inference optimizations

- **SUT**
  - Multi-instance
  - Warmup
  - Batching
    - Reduce padding
    - Dynamic batching
    - Constant batching

- **Framework/Model**
  - Op fusion
  - Low precision
  - Sparsity
  - Vectorize
Batching – reduce padding

- Sort all inputs in offline queue according to input size, from largest input size to smallest input size
- Pad batch inputs to the largest input size in the batch
Constant batching

- DLRM: ensure the total number of user-item pairs is constant per batch (constant batching)
- E.g. DLRM has input with 100, 200, 300, 400, 500, 600, 700 data points
- A constant batching scheme could be:
  - 100: BS=420,000/100
  - 200: BS=420,000/200
  - 300: BS=420,000/300
  - 400: BS=420,000/400
  - 500: BS=420,000/500
  - 600: BS=420,000/600
  - 700: BS=420,000/700
Dynamic batching

- Select the right batch size for the input shape
- E.g. one-time calibration step on different BERT input shapes to get seqlen→BS lookup table
- Batching according to seqlen→BS lookup table from calibration step
Deep learning inference optimizations

- SUT (loadgen bridge)
  - Multi-instance
  - Warmup
  - Batching
    - Reduce padding
    - Dynamic batching
    - Constant batching

- Framework/Model
  - Op fusion
  - Low precision
  - Sparsity
  - Vectorize
Deep learning inference optimizations

- **SUT**
  - Multi-instance
  - Warmup
  - Batching
    - Reduce padding
    - Dynamic batching
    - Constant batching

- **Framework/Model**
  - Op fusion
  - Low precision
  - Sparsity
  - Vectorize
Op fusion

Left: 10 memory accesses and 2 quantization ops

Right: 6 memory accesses and 1 quantization op


Deep learning inference optimizations

- **SUT**
  - Multi-instance
  - Warmup
  - Batching
    - Reduce padding
    - Dynamic batching
    - Constant batching

- **Framework/Model**
  - Op fusion
  - Low precision
  - Sparsity
  - Vectorize
Low precision inference

- Dedicated low precision computing unit provide higher computation throughput than FP32
- 8bit (int8)
- 16 bit (FP16, BF16)
- Low precision is common practice of MLPerf inference workloads:

<table>
<thead>
<tr>
<th></th>
<th>ResNet50</th>
<th>SSD-RN34</th>
<th>3DUNet</th>
<th>BERT</th>
<th>DLRM</th>
<th>RNN-T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>Int8</td>
<td>Int8</td>
<td>Int8</td>
<td>Int8</td>
<td>Int8</td>
<td>Int8 encoder+16bit decoder</td>
</tr>
</tbody>
</table>

Lowest precision had been used by submitters in MLPerf inference v1.0, closed division [https://mlcommons.org/en/inference-datacenter-10/](https://mlcommons.org/en/inference-datacenter-10/)
Common low precision (int8) flow

1. Calibration – find out the range/distribution of each activation tensor
   ✗ Calibration dataset – a dataset sampled from train dataset

2. Quantization configuration – decide quantization scale/zero-point from calibration range/distribution
   ✗ Min-max quantization
   ✗ KL-divergence quantization
   ✗ Per tensor/per channel scale/zero-point

3. Quantize model weights offline or during model checkpoint loading
4. Quantize activation when input is FP32 and next op expect int8 activation
5. Dequantize activation when output is int8 and next op expect FP32 activation
6. Chain int8 op as much as possible – saves dequantization/quantization overhead

* 3, 4, 5 might be done automatically depending on the framework/inference engine
Low Precision Inference

The Intel DL Boost AVX512_VNNI VPDPBUSD instruction enables 8-bit multiplies with 32-bit accumulates with 1 instruction u8×s8→s32 providing a theoretical peak compute gain of 4x int8 OPS over fp32. Image credit to Israel Hirsh.

The ISA Fusion VPDPBUSD executes on both Port 0 and Port 5 in 1 cycle.

<table>
<thead>
<tr>
<th>SRC 1 8-bit</th>
<th>(A_0)</th>
<th>(A_1)</th>
<th>(A_2)</th>
<th>(A_3)</th>
<th>...</th>
<th>(A_{63})</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC 2 8-bit</td>
<td>(B_0)</td>
<td>(B_1)</td>
<td>(B_2)</td>
<td>(B_3)</td>
<td>...</td>
<td>(A_{63})</td>
</tr>
<tr>
<td>SRC 3 / DEST 32-bit</td>
<td>(C_0)</td>
<td>...</td>
<td>...</td>
<td>(C_{15})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(A_0B_0 + A_1B_1 + A_2B_2 + A_3B_3 + C_0)</td>
<td>...</td>
<td>(A_{60}B_{60} + A_{61}B_{61} + A_{62}B_{62} + A_{63}B_{63} + C_{15})</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Deep learning inference optimizations

- **SUT**
  - Multi-instance
  - Warmup
  - Batching
    - Reduce padding
    - Dynamic batching
    - Constant batching

- **Framework/Model**
  - Op fusion
  - Low precision
  - **Sparsity**
  - Vectorize
Sparsity

- Tile-based sparsity: blocks of consecutive zeros and non-zeros
- Training with sparsity:
  - magnitude-based pruning
  - 99% sparsity ratio for the sparse General Matrix Multiplications (GEMMs)
- MLPerf DLRM benchmark: 0.93% accuracy loss & 1.4X performance gain


Baseline FP32 DLRM dense model
source: https://github.com/mlcommons/inference/tree/master/recommendation/dlrm/pytorch
Optimized FP32 DLRM sparse model reproductions steps can be found at
https://github.com/mlcommons/submissions_inference_1_0/tree/master/open/Intel/code/dlrm-99 (link active on 4/21). Both
the dense model and the sparse DLRM model are tested on the same hardware with the server scenario: 1-node, 2x Intel
Xeon Platinum 8380 processor on Coyote Pass with 1 TB (16 slots/ 64GB/3200) total DDR4 memory, ucode 0x8d05a260, HT
on, Turbo on, Ubuntu 20.04.2 LTS, 5.4.0-66-generic. Please see the additional hardware and framework detail in v1.0
Deep learning inference optimizations

- **SUT**
  - Multi-instance
  - Warmup
  - Batching
    - Reduce padding
    - Dynamic batching
    - Constant batching

- **Framework/Model**
  - Op fusion
  - Low precision
  - Sparsity
  - Vectorize
RNN-T inference -- vectorize greedy decoder

- Speech recognition task
- Inference contains three steps
  - Wave to input feature
  - Encoder
  - Greedy decoder
- Greedy decoder has low efficiency
  - Control flow amongst tensor operation
  - Single batch single time step LSTM

Sound wav input

Preprocessing

LSTM encoder

Greedy decoder

“Hello, world!”
RNN-T inference -- vectorize greedy decoder

Batch greedy decoder in RNN-T: 3.3X overall performance gain vs. sequential greedy decoder

HW used to measure performance gain can be found in the following link (https://www.intel.com/content/www/us/en/artificial-intelligence/posts/3rd-gen-xeon-mlperf-performance-gains.html)
Summary

- **SUT**
  - Multi-instance
  - Warmup
  - Batching
    - Reduce padding
    - Dynamic batching
    - Constant batching

- **Framework/Model**
  - Op fusion
  - Low precision
  - Sparsity
  - Vectorize
Summary

- **SUT**
  - Multi-instance (CPU)
  - Warmup (generic)
  - Batching
    - Reduce padding
    - Dynamic batching
    - Constant batching

- **Framework/Model**
  - Op fusion (generic)
  - Low precision (generic)
  - Sparsity (CPU)
  - Vectorize (CPU)
Q & A