Machine Learning with PyTorch

Smeet Chheda
Stony Brook University
Ookami User Group Meeting – 02/10/2022
Machine Learning has been applied to production systems in many areas: Self-driving cars, smart assistants on cellular devices, traffic camera systems, etc.
PyTorch and other Frameworks/Libraries

- PyTorch is an open source machine learning library based on the Torch library.
  - It provides tensor computations with gpu acceleration and reverse-mode automatic differentiation through autograd
  - Version 1.10.0 is cloned from GitHub and kept consistent across all builds

<table>
<thead>
<tr>
<th>Compiler</th>
<th>BLAS Library</th>
<th>Horovod</th>
<th>Compiler Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fujitsu v4.5 (Clang mode)</td>
<td>SSL2</td>
<td>v0.20.3 (Fujitsu patch)</td>
<td>-Kfast -Knolargepage -lpthread</td>
</tr>
<tr>
<td>Arm v21.0</td>
<td>Arm Performance Libraries</td>
<td>v0.23.0</td>
<td>-Ofast -pthread -mcpu=a64fx</td>
</tr>
<tr>
<td>GNU v10.3</td>
<td>Arm Performance Libraries</td>
<td>v0.23.0 (eigen patch)</td>
<td>-Ofast -pthread -mcpu=a64fx</td>
</tr>
<tr>
<td>GNU v10.3</td>
<td>OpenBLAS v0.3.19</td>
<td>v0.23.0 (eigen patch)</td>
<td>-Ofast -pthread -mcpu=a64fx</td>
</tr>
<tr>
<td>GNU v10.3</td>
<td>BLIS 0.8.1</td>
<td>v0.23.0</td>
<td>-Ofast -pthread -mcpu=a64fx</td>
</tr>
</tbody>
</table>
Build Process

- Python v3.8.2 is built from source with O3 optimization for all compilers.
  - The python object file is re-compiled with the respective C++ compiler and linked against the BLAS libraries.

- PyTorch is built with oneDNN v2.4.3 support (formerly known as MKL-DNN)
  - Fujitsu achieved this by creating an aarch64 version of xbyak JIT assembler.
  - `xbyak_aarch64` and `xbyak_translator_aarch64` have been primarily developed to enable assembly coding with full SVE support and porting oneDNN to aarch64.
  - Their work has been upstreamed and can be used directly (original scripts require building xed (Intel's x86 Decoder) prior to installing oneDNN for A64FX).
  - A patch is applied to cmake files (FindBLAS) to search & recognize SSL2, ArmPL, OpenBLAS and BLIS.

- Horovod, a distributed deep learning framework, is built with openMPI
  - v4.0.1 (modified) for Fujitsu compilers and v4.1.1 for ARM and GNU compilers.
Single Node Training on A64FX

- **Task:** Image Classification
- **Dataset:** Photo, Art Painting, Cartoon, Sketch (PACS*) - 4 domains, 7 classes, 9991 images
- **Deep Neural Network:** ResNet50

Training: 6101 images  
Transforms: Resizing, Horizontal Flips, Color Jittering, Gray scaling, tensor conversion, normalization

Evaluation: 3942 images  
Transforms: Resizing, tensor conversion, normalization

- Model training & inference can be improved by using different memory formats (NCHW default, NHWC, nChw16c - mkldnn block format)
- Some of the variables used to optimize the runs - 
  OMP_NUM_THREADS=48  and  XOS_MMM_L_HPAGE_TYPE=none
- Using TCMalloc for memory allocation.
Single Node Training on A64FX

**Block Memory Format**

<table>
<thead>
<tr>
<th>Compiler + BLAS library</th>
<th>Batch Size=256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fujitsu+SSL2</td>
<td>215.025</td>
</tr>
<tr>
<td>ARM+ArmPL</td>
<td>221.217</td>
</tr>
<tr>
<td>GNU+ArmPL</td>
<td>193.97</td>
</tr>
<tr>
<td>GNU+OpenBLAS</td>
<td><strong>190.29</strong></td>
</tr>
<tr>
<td>GNU+BLIS</td>
<td><strong>189.99</strong></td>
</tr>
</tbody>
</table>

**NCHW format, oneDNN disabled**

<table>
<thead>
<tr>
<th>Compiler + BLAS library</th>
<th>Batch Size=128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fujitsu+SSL2</td>
<td>347.26</td>
</tr>
<tr>
<td>ARM+ArmPL</td>
<td>703.83</td>
</tr>
<tr>
<td>GNU+ArmPL</td>
<td>609.92</td>
</tr>
<tr>
<td>GNU+OpenBLAS</td>
<td><strong>1022.47</strong></td>
</tr>
</tbody>
</table>

*Lower is better*
Importance of (A64FX supported) vectorized libraries

- BLIS was built with a generic configuration on A64FX (-mcpu flag added in {C,CXX}FLAGS explicitly)
- During configuration, PyTorch does not find LAPACK support. The graph shows speedup for other library builds.
- These libraries are important because oneDNN does not have optimized implementations for all operators provided by PyTorch. In that case, we must convert the outputs from prior layers to the dense (NCHW) representation, perform the unsupported operation and then convert the output back to the block format.

** This can also be seen in OpenBLAS v0.3.10 (sve-enabled sgemm & dgemm kernels added in v0.3.19)

Epoch time for GNU + BLIS**: 10162.18s
Distributed Training with Horovod

- Dataset: CIFAR-10* - 10 classes, 60000 images (50,000 train & 10,000 test)
- Model: ResNet50
- Train batch size = 512 images

- Horovod built with OpenMPI to run distributed code
- Process mapping achieved by --map-by flag
- Two mappings tested:
  - 1 process per NUMA region
    - XOS_MMM_L_PAGING_POLICY=demand:demand:demand
    - --map-by ppr:1:numa:pe=12
  - 1 process per node
    - XOS_MMM_L_HPAGE_TYPE=none
    - --map-by ppr:1:node:pe=48

[Image of CIFAR-10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck]
Distributed Training with Horovod

Images processed at 128 nodes per node: 30.75%

Images processed at 128 nodes per node: 67.14%
In the prior slide, we see a discrepancy in distributing the workload over multiple nodes with the GNU compiler. After checking the trace files (Horovod timeline), we see that there is a large communication bottleneck compared to the ARM and Fujitsu compilers. Still trying to figure out what could be causing this…

This issue is reproducible and similar results are seen with different versions ---
- GNU v10.3.0 + OpenMPI (v4.1.1, v4.1.2) and GNU v11.2.0 + openMPI v4.1.2
Sidebar: GNU/ARM compilers with SSL2

● With some effort, compiling oneDNN and PyTorch with GNU/ARM compilers and Fujitsu’s SSL2 BLAS library is possible.
  ○ For oneDNN, elementwise operations run successfully, but forward and backward passes for convolution operations run into a Segmentation Fault.
  ○ For PyTorch, MLP networks can run end-to-end without running into any errors (and no significant changes in precision). CNNs run into a fault, as mentioned above.

● If anyone tries to see the dependencies of shared libraries provided by Fujitsu….

```
[schheda@fj-debug2 lib64]$ pwd
/opt/FJSVtclamga/cp-1.0.20.06/lib64
[schheda@fj-debug2 lib64]$ ldd libbfjlapackexsve.so
statatically linked
[schheda@fj-debug2 lib64]$  
```
Summary

- Single node training runs show significant improvements by leveraging a block memory format and mkl-dnn fused operations compared to disabled mkl-dnn.
- GNU compiler builds outperform Fujitsu compiler build with mkl-dnn enabled.
- Fujitsu SSL2 shows higher performance when using native operations defined in ATen – tensor core library.

- Distributed training runs with Horovod show better run time per epoch for mkl-dnn fused operations and block format when using 1 process per node and 48 threads per process compared to 1 process per NUMA region and 12 threads per NUMA. The same is not true when mapping processes by NUMA regions.
  - Fused operations lead to lower kernel launches.
  - However, the scaling efficiency of using 1 process per node is much lower than the efficiency of using 1 process per NUMA region.
- Fujitsu compiler outperforms ARM and GNU compilers w.r.t. scaling.
Acknowledgement

I would like to thank Tony, Eva and the entire Ookami Team for their help and sharing invaluable knowledge.
Thank you