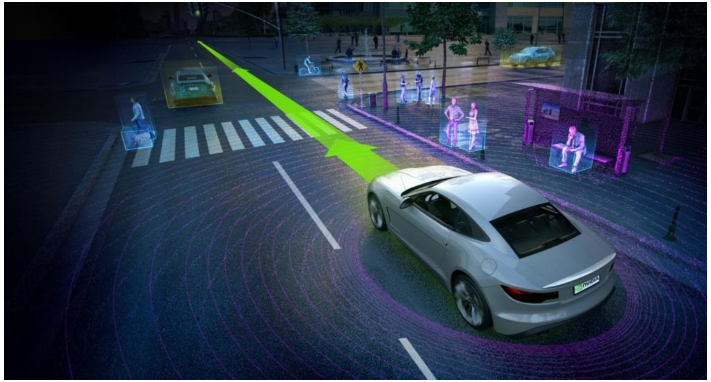
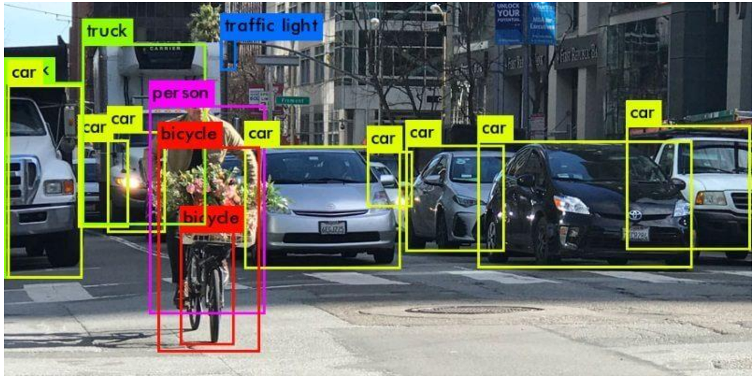
Machine Learning with PyTorch

Smeet Chheda
Stony Brook University
Ookami User Group Meeting – 02/10/2022

Machine Learning

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

Compiler	BLAS Library	Horovod	Compiler Flags
Fujitsu v4.5 (Clang mode)	SSL2	v0.20.3 (Fujitsu patch)	*-Kfast -Knolargepage -lpthread*
Arm v21.0	Arm Performance Libraries	v0.23.0	*-Ofast -pthread -mcpu=a64fx*
GNU v10.3	Arm Performance Libraries	v0.23.0 (eigen patch)	*-Ofast -pthread -mcpu=a64fx*
GNU v10.3	OpenBLAS v0.3.19	v0.23.0 (eigen patch)	*-Ofast -pthread -mcpu=a64fx*
GNU v10.3	BLIS 0.8.1	v0.23.0	*-Ofast -pthread -mcpu=a64fx*

Build Process

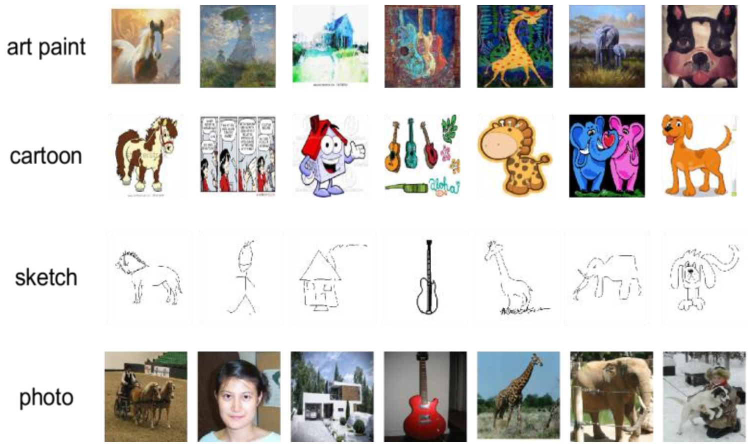
- Python v3.8.2 is built from source with O3 optimization for all compilers.
  - The python object file is re-compiled with the respective C++ compiler and linked against the BLAS libraries.

- 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 Encoder 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: **P**hoto, **A**rt Painting, **C**artoon, **S**ketch (PACS*) - 4 domains, 7 classes, 9991 images
- Deep Neural Network - ResNet50

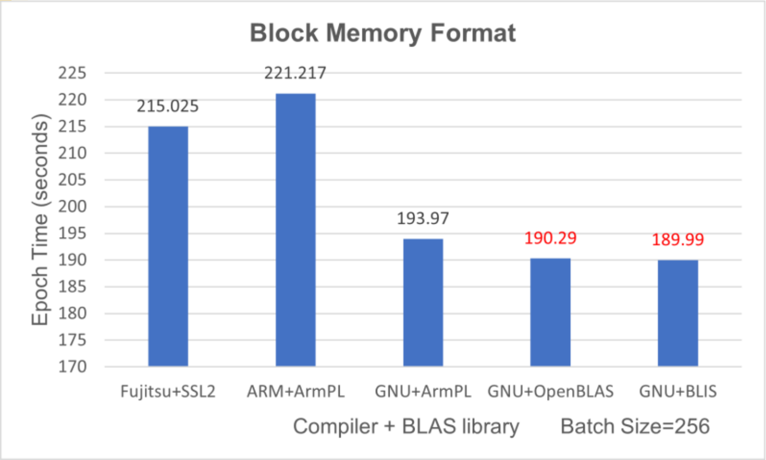


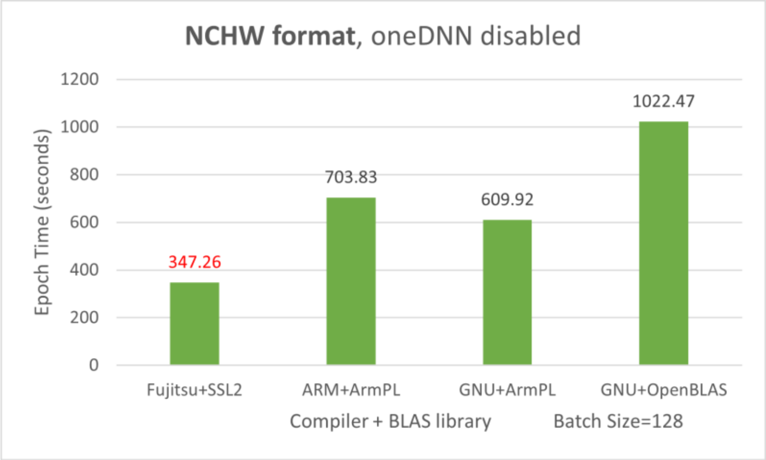
Training : 6101 images
Transforms : Resizing, Horizontal Flips, Color Jittering, Gray scaling, tensor conversion, normalization

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

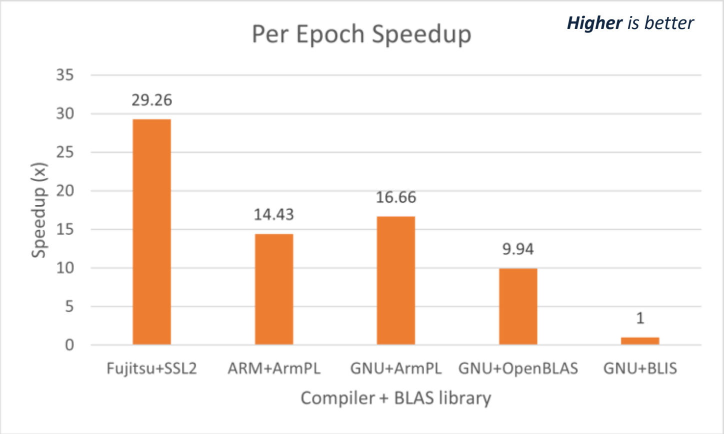




Lower is better

iACS INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

Importance of (A64FX supported) vectorized libraries



Epoch time for GNU + BLIS** : 10162.18s

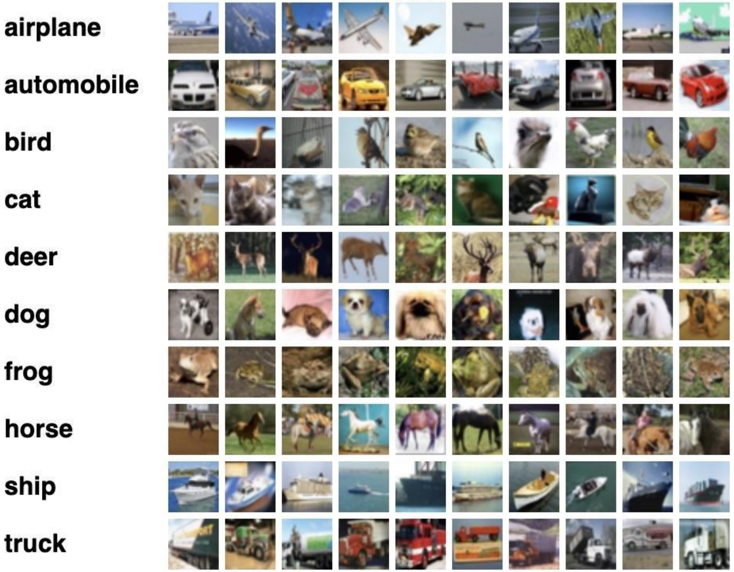
- BLIS was built with a generic configuration on A64FX (-mcpu flag added in {C,CXX}FLAGS explicitly)
- During configuration, PyTorch does not find LAPACK support. The graph shows speedup for other library builds.

- 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*)

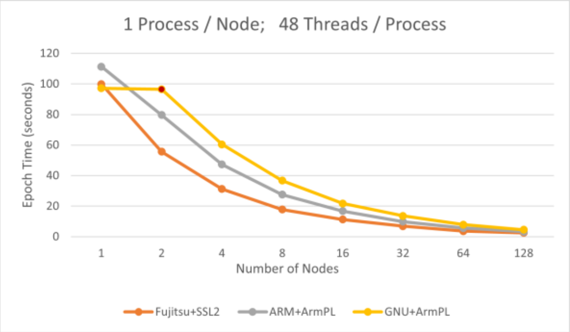
Distributed Training with Horovod

- 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`

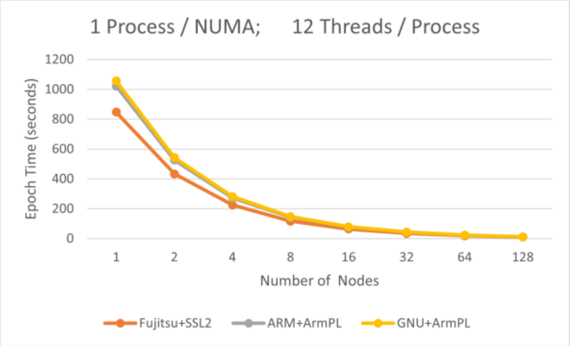
Distributed Training with Horovod



	Epoch Time (seconds)							
# Nodes	1	2	4	8	16	32	64	128
Fujitsu+SSL2	99.93	55.67	31.25	17.8	11.25	6.89	3.67	2.54
ARM+ArmPL	111.26	79.61	47.31	27.52	16.78	9.86	5.68	3.27
GNU+ArmPL	97.14	96.48	60.44	36.7	21.7	13.65	7.94	4.58

Images processed at 128 nodes per node: 30.75%

Lower is better



	Epoch Time (seconds)							
# Nodes	1	2	4	8	16	32	64	128
Fujitsu+SSL2	846.79	433.2	224.2	116.9	63.4	35.04	18.4	9.85
ARM+ArmPL	1022.6	526.1	271.8	142.3	76.61	42.77	22.1	11.5
GNU+ArmPL	1055.9	542.1	280.3	146.8	78.81	44.07	22.6	11.7

Images processed at 128 nodes per node: 67.14%

Scaling Discrepancy with GNU compiler

- 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… ⚠

Name	Wall Duration	Self time	Average Wall Duration	Occurrences
NEGOTIATE_ALLREDUCE	1,994.478 ms	1,994.478 ms	40.704 ms	49
ALLREDUCE	4,362.944 ms	5.900 ms	89.040 ms	49
MEMCPY_IN_FUSION_BUFFER	0.760 ms	0.760 ms	0.020 ms	38
MPI_ALLREDUCE	4,355.308 ms	4,355.308 ms	88.884 ms	49
MEMCPY_OUT_FUSION_BUFFER	0.976 ms	0.976 ms	0.026 ms	38
Totals	**10,714.466 ms**	**6,357.422 ms**	**48.047 ms**	**223**

ARM compiler

Name	Wall Duration	Self time	Average Wall Duration	Occurrences
NEGOTIATE_ALLREDUCE	35,801.265 ms	35,801.265 ms	730.638 ms	49
ALLREDUCE	6,344.522 ms	4.804 ms	129.480 ms	49
MEMCPY_IN_FUSION_BUFFER	5.465 ms	5.465 ms	0.114 ms	48
MPI_ALLREDUCE	6,328.858 ms	6,328.858 ms	129.160 ms	49
MEMCPY_OUT_FUSION_BUFFER	5.395 ms	5.395 ms	0.112 ms	48
Totals	**48,485.505 ms**	**42,145.787 ms**	**199.529 ms**	**243**

GNU compiler

2 node trace information over 1 epoch

- 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/FJSVstclanga/cp-1.0.20.06/lib64
[schheda@fj-debug2 lib64]$ ldd libfjlapackexsve.so
        statically 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