S21501 Tuning GPU Server for Deep Learning Performance

Dell EMC HPC & AI Innovation Lab

Frank Han : frank.han@dell.com
Rengan Xu : rengan.xu@dell.com

DELL EMC

Agenda

- The goal of this session
- About us/HPC innovation lab
- MLPerf
- Our testing bed
- Single nodes training finding
- Multiple nodes training
- Inference

Goal of this session

- Show some possible tuning knobs
- Share results from our tuning
- More RFPs are using MLPerf
- Next version submission

HPC and DL Engineering - what we do

- **Design and build** systems for HPC and Deep Learning workloads.
- **Systems** include compute, storage, network, software, services, support.
- **Integration** with factory, software, services.
- Power and performance analysis, tuning, **best practices**, trade-offs.
- Focus on **application** performance.
- **Vertical** solutions.
- Research and **proof of concept** studies.
- **Publish** white papers, blogs, conference papers (www.hpcatdell.com)
- **Access** to the systems in **the lab**

World-class infrastructure in the Innovation Lab

13K ft.^{2} lab, 1,300+ servers, ~10PB storage dedicated to HPC in collaboration with the community

Zenith

- TOP500-class system based on Intel Scalable Systems Framework (OPA, KNL, Xeon, OpenHPC)
- 424 nodes dual Intel Xeon Gold processors, Omni-Path fabric.
- +160 Intel Xeon Phi (KNL) servers.
- **Over 1 PF combined performance!**
- #265 on Top500 June 2018, 1.86 PF theoretical peak
- Lustre, Isilon H600, Isilon F800 and NSS storage
- Liquid cooled and air cooled

Rattler

- Research/development system with Mellanox, NVIDIA and Bright Computing
- 88 nodes with EDR InfiniBand and Intel Xeon Gold processors
- 32x PowerEdge C4140 nodes with 4x NVIDIA GPUs

Other systems

- 32 node AMD cluster, storage solutions, etc.

DELL EMC

MLPerf

MLPerf training Introduction

Benchmark	Dataset	Quality Target	Reference Implementation Model
Image classification	ImageNet (224x224)	75.9% Top-1 Accuracy	Resnet-50 v1.5
Object detection (light weight)	COCO 2017	23% mAP	SSD-ResNet34
Object detection (heavy weight)	COCO 2017	0.377 Box min AP, 0.339 Mask min AP	Mask R-CNN
Translation (recurrent)	WMT English-German	24.0 BLEU	GNMT
Translation (non-recurrent)	WMT English-German	25.0 BLEU	Transformer
Recommendation	Undergoing modification		
Reinforcement learning	N/A	Pre-trained checkpoint	Mini Go

- A broad ML benchmark suite for measuring performance of ML frameworks, ML hardware accelerators, and ML cloud platforms.
- Cover different DL domains
- Proper metrics (training time, accuracy)
- Real datasets

Who did what in MLPerf training v0.6

DELL

- Dell server optimization
  - CPU binding
  - BIOS HT
  - Batch Size/Learning rate/Warm-up steps/etc.
  - NCCL P2P & Tree/Ring vs DGX-1 optimized

NVIDIA/Others

- Select framework
- Code optimization by each team
  - NCCL 2.3 -> 2.4.7
  - DALI
  - Active function
  - Multiple GPU(Horovod)
- DGX optimization
  - Batch Size
  - etc.

MLPerf Community

- Define the Sub-benchmarks
  - Accuracy
  - Models
  - Datasets
- Github sample codes 1xP100

MLPerf v0.5 and v0.6 difference – MLPerf community

- Raises quality targets:
  - Image classification (ResNet) to 75.9% (v0.5 was 74.9%)
  - light-weight object detection (SSD) to 23% mAP (v0.5 was 21.2%)
  - recurrent translation (GNMT) to 24 Sacre BLEU (v0.5 was 21.8)
- Allows use of the LARS optimizer for ResNet, enabling additional scaling
- Experimentally allows a slightly larger set of hyperparameters to be tuned
  - Enabling faster performance and some additional scaling
- Changes timing to start the first time the application accesses the training dataset, thereby excluding startup overhead
  - This change was made because the large scale systems measured are typically used with much larger datasets than those in MLPerf, and hence normally amortize the startup overhead over much greater training time
- Improves the MiniGo benchmark in two ways
  - First, it now uses a standard C++ engine for the non-ML compute, which is substantially faster than the prior Python engine.
  - Second, it now assesses quality by comparing to a known-good checkpoint, which is more reliable than the previous very small set of game data
- Suspends the Recommendation benchmark while a larger dataset and model are being created

Resnet-50 v0.6 improvement - NVIDIA

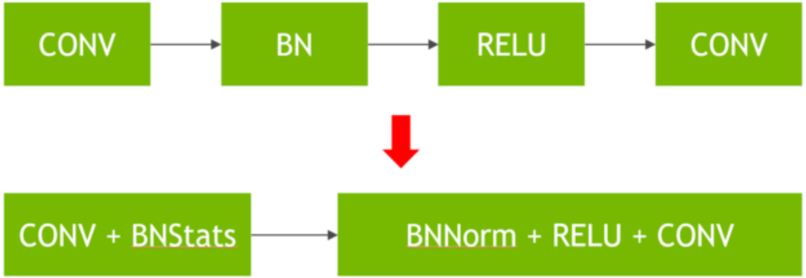
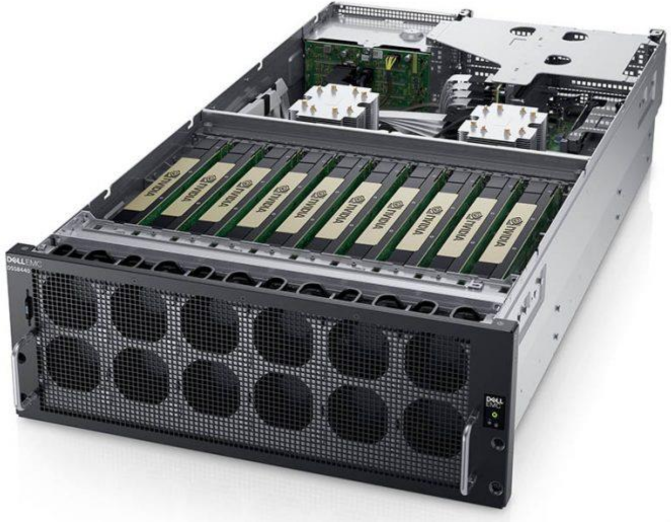


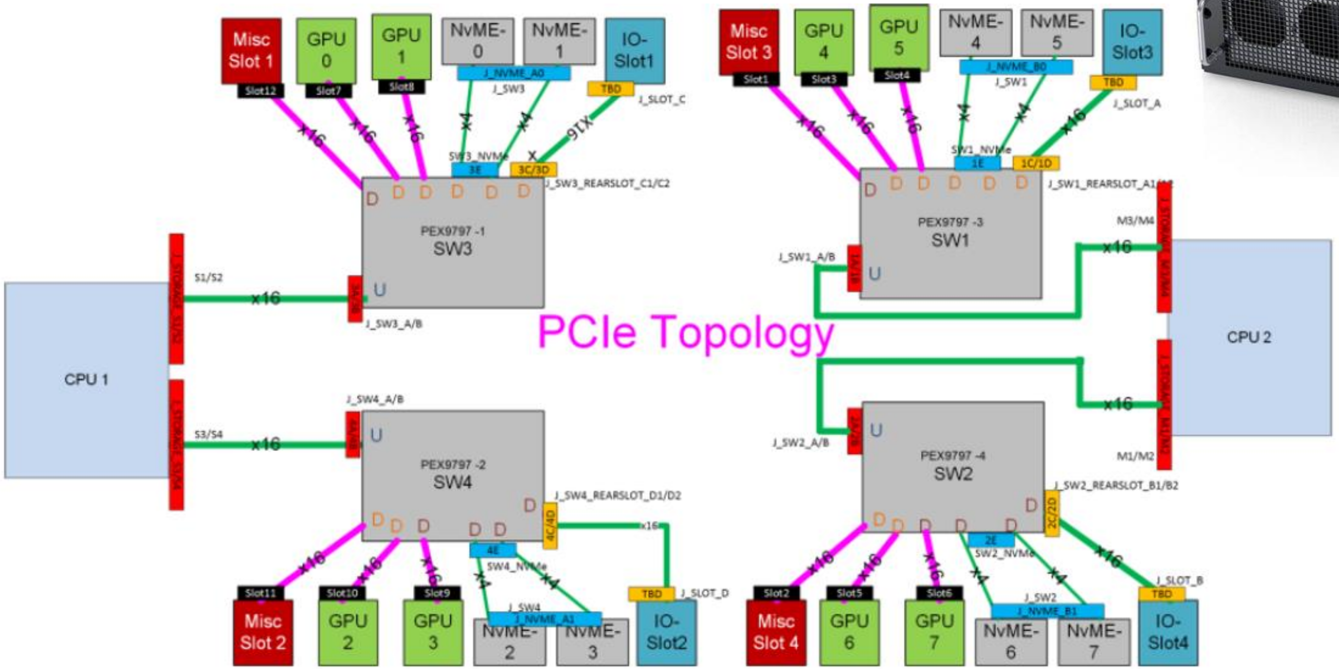
Figure 1. New fused convolution + batchnorm kernels make better use of Tensor Cores and halve the number of discrete kernels that need to run

- Image Classification / ResNet-50 (1.24x improvement). Implemented new fused convolution + batchnorm kernels through cuDNN 7.6.
  - This optimization drastically reduces the cost of batch normalization (a bandwidth-limited operation and does not benefit from Tensor Cores) by performing the normalization in adjacent convolution layers, as outlined in figure 1.
- A variety of DALI-related improvements accelerated the data input pipeline, enabling it to keep up with high-speed neural network processing.
  - These include using NVJPEG and ROI JPEG decode to limit the JPEG decode work to the region of the raw image actually used. We also used Horovod for data parallel execution, allowing us to hide the exchange of gradients between GPUs behind other back-propagation work happening on the GPU.

Testbed

Dell EMC DSS8440

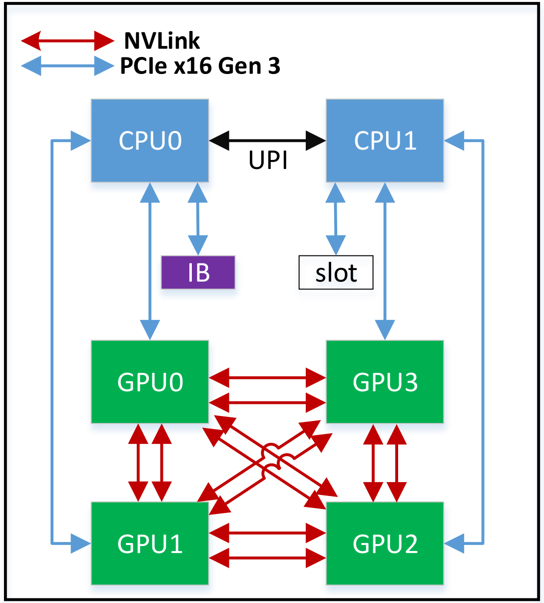


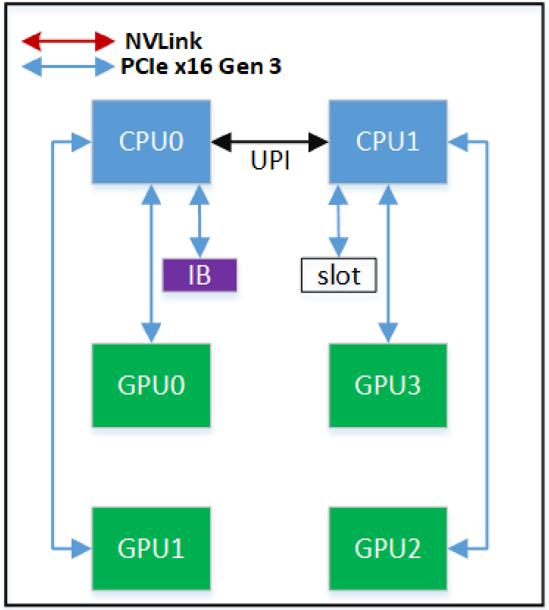


D&LLEMC

C4140M – NVLINK System

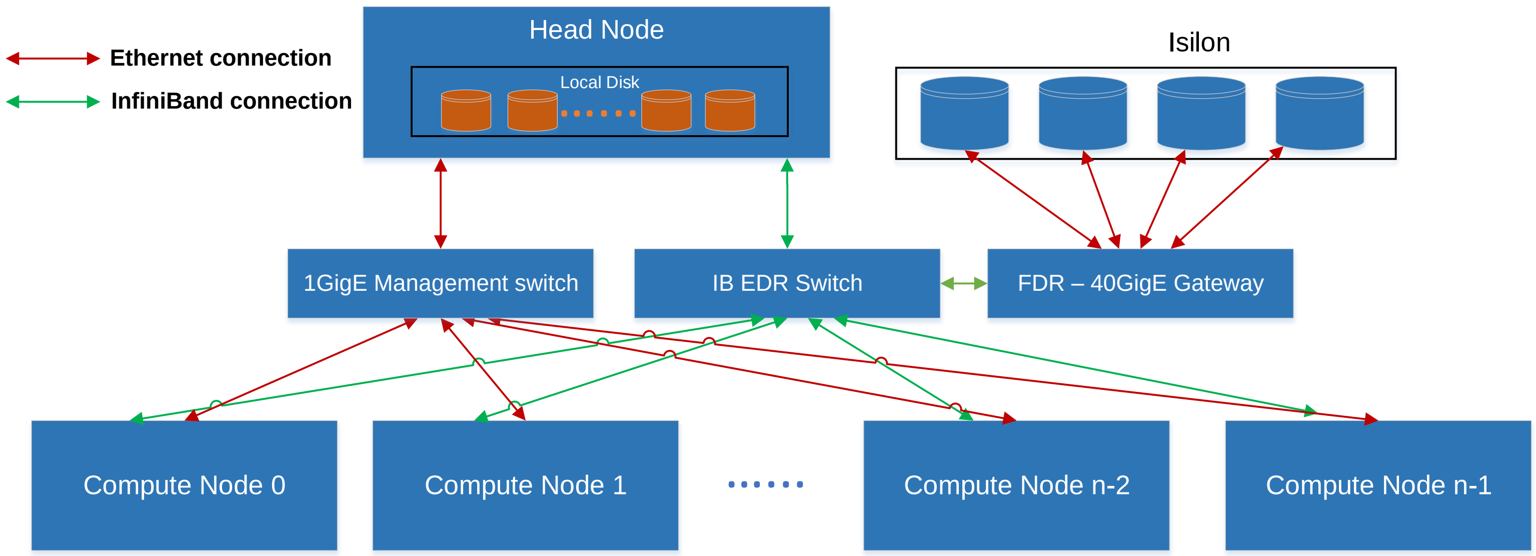
- All accelerators are put at the front
- The Only Dell system has NVLINK
- Smaller failure zone
- Dual redundant PSU





LLEMC

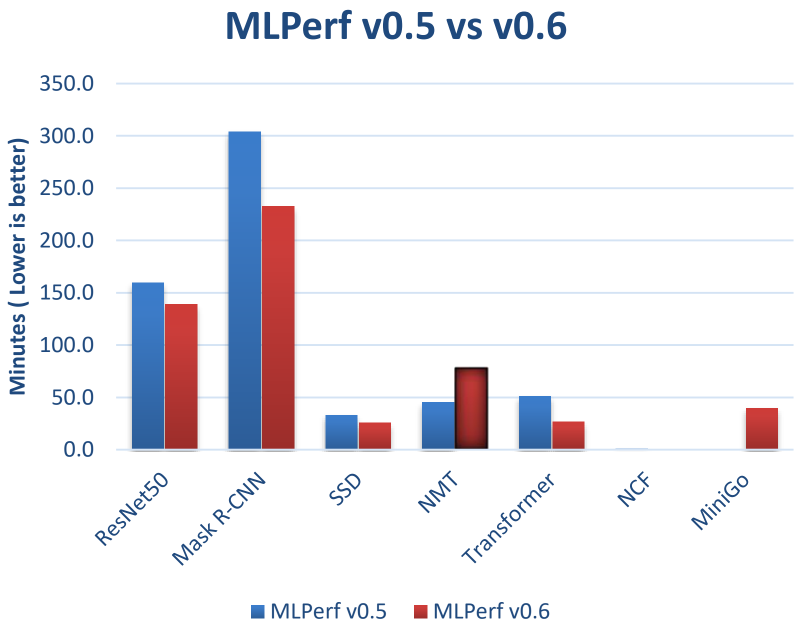
Our cluster



Ethernet connection
InfiniBand connection
Head Node
Local Disk
Isilon
1GigE Management switch
IB EDR Switch
FDR – 40GigE Gateway
Compute Node 0
Compute Node 1
Compute Node n-2
Compute Node n-1

Single nodes

MLPerf results – Original NVIDIA docker run on DSS8440



Network	Server - DSS8440	
	MLPerf v0.5	MLPerf v0.6
ResNet50-v1.5	159.6	139.0
Mask R-CNN	304.0	232.6
SSD	33.2	25.7
NMT	45.3	78.0
Transformer	51.0	26.7
NCF	1.0	N/A
MiniGo	N/A	34.0

- Time taken to converge GNMT from v0.5 to v0.6 is significantly high.
- Analysis in the next slides

GNMT Profiling results

```
======== Error: no application specified.
[root@dss01 rnn_translator]# nvprof -i gnmt.nvvp20190724182831.499
======== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   59.65%  534.420s       970  550.95ms  96.596ms  848.15ms  ncclAllReduceRingLLKernel_sum_f16(ncclColl)
                   10.35%  92.7072s    674052  137.54us  85.567us  393.18us  volta_fp16_s884gemm_fp16_128x64_ldg8_f2f_nn
                    6.35%  56.8967s    169372  335.93us  125.22us  33.244ms  volta_fp16_s884gemm_fp16_128x128_ldg8_f2f_nn
                    5.79%  51.8429s    166814  310.78us  190.97us  4.8022ms  volta_fp16_s884gemm_fp16_256x128_ldg8_f2f_nn
                    4.50%  40.2976s     12484  3.2279ms  165.53us  9.2248ms  volta_fp16_s884gemm_fp16_256x128_ldg8_f2f_nt
                    3.06%  27.4563s     16616  1.6524ms  78.047us  31.857ms  volta_fp16_s884gemm_fp16_128x128_ldg8_f2f_nt
                    1.75%  15.6660s      3225  4.8577ms  44.831us  31.040ms  volta_fp16_s884gemm_fp16_128x128_ldg8_f2f_tn
                    1.20%  10.7461s    335902  31.991us  2.6880us  75.679us  void LSTM_elementWise_fp<__half, __half, float, cudnnRN
e_t=2>(int, int, int, int, __half const *, __half const *, __half const *, __half const *, cudnn::reduced_divisor, __half*, __half*,
, __half const *, __half*, bool, int, cudnnRNNClipMode_t, cudnnNanPropagation_t, float, float)
                    1.11%  9.96056s    334286  29.796us  17.184us  72.319us  void LSTM_elementWise_bp1<__half, __half, float>(int, i
```

- Analysis using “nvprof” profiling tool show most of the time (524.42s) is spent on *ncclAllReduceRing* communication

```
[root@dss01 rnn_translator]# head -7 *.nvprof
==> Ring_p2p1.nvprof <==
======== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   27.96%  140.717s       970  145.07ms  94.456ms  1.11446s  ncclAllReduceRingLLKernel_sum_f16(ncclColl)
                   18.47%  92.9812s    674052  137.94us  85.983us  393.31us  volta_fp16_s884gemm_fp16_128x64_ldg8_f2f_nn
                   11.36%  57.1649s    169372  337.51us  124.70us  33.029ms  volta_fp16_s884gemm_fp16_128x128_ldg8_f2f_nn
                   10.33%  51.9953s    166814  311.70us  195.49us  4.8308ms  volta_fp16_s884gemm_fp16_256x128_ldg8_f2f_nn
                    8.01%  40.2951s     12484  3.2277ms  170.65us  9.1541ms  volta_fp16_s884gemm_fp16_256x128_ldg8_f2f_nt

==> Ring_p2p3.nvprof <==
======== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   58.95%  519.359s       970  535.42ms  95.981ms  787.84ms  ncclAllReduceRingLLKernel_sum_f16(ncclColl)
                   10.49%  92.4277s    674052  137.12us  85.727us  397.18us  volta_fp16_s884gemm_fp16_128x64_ldg8_f2f_nn
                    6.46%  56.8969s    169372  335.93us  124.09us  33.208ms  volta_fp16_s884gemm_fp16_128x128_ldg8_f2f_nn
                    5.92%  52.1251s    166814  312.47us  194.53us  4.8313ms  volta_fp16_s884gemm_fp16_256x128_ldg8_f2f_nn
                    4.58%  40.3089s     12484  3.2288ms  169.73us  9.4176ms  volta_fp16_s884gemm_fp16_256x128_ldg8_f2f_nt

==> Ring_p2p5.nvprof <==
======== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   61.79%  588.464s       970  606.66ms  90.532ms  1.04102s  ncclAllReduceRingLLKernel_sum_f16(ncclColl)
                    9.77%  93.0078s    674052  137.98us  85.503us  398.40us  volta_fp16_s884gemm_fp16_128x64_ldg8_f2f_nn
                    6.00%  57.1875s    169372  337.64us  125.12us  33.147ms  volta_fp16_s884gemm_fp16_128x128_ldg8_f2f_nn
                    5.50%  52.4149s    166814  314.21us  195.33us  4.8320ms  volta_fp16_s884gemm_fp16_256x128_ldg8_f2f_nn
                    4.26%  40.6027s     12484  3.2524ms  170.02us  9.4077ms  volta_fp16_s884gemm_fp16_256x128_ldg8_f2f_nt

==> Tree_p2p1.nvprof <==
======== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   40.49%  247.853s       970  255.52ms  107.21ms  1.06158s  ncclAllReduceTreeLLKernel_sum_f16(ncclColl)
                   15.34%  93.8892s    674052  139.29us  85.983us  405.12us  volta_fp16_s884gemm_fp16_128x64_ldg8_f2f_nn
                    9.39%  57.4789s    169372  339.36us  123.52us  33.342ms  volta_fp16_s884gemm_fp16_128x128_ldg8_f2f_nn
                    8.48%  51.9241s    166814  311.27us  199.20us  4.8339ms  volta_fp16_s884gemm_fp16_256x128_ldg8_f2f_nn
                    6.63%  40.5821s     12484  3.2507ms  172.86us  9.3337ms  volta_fp16_s884gemm_fp16_256x128_ldg8_f2f_nt

==> Tree_p2p5_2.nvprof <==
======== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   49.95%  363.391s       970  374.63ms  93.316ms  988.59ms  ncclAllReduceTreeLLKernel_sum_f16(ncclColl)
                   12.82%  93.2443s    674052  138.33us  85.407us  398.01us  volta_fp16_s884gemm_fp16_128x64_ldg8_f2f_nn
                    7.89%  57.3899s    169372  338.84us  122.82us  33.187ms  volta_fp16_s884gemm_fp16_128x128_ldg8_f2f_nn
                    7.22%  52.5207s    166814  314.85us  199.61us  4.8807ms  volta_fp16_s884gemm_fp16_256x128_ldg8_f2f_nn
                    5.58%  40.5722s     12484  3.2499ms  173.34us  9.3915ms  volta_fp16_s884gemm_fp16_256x128_ldg8_f2f_nt

==> Tree_p2p5.nvprof <==
======== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   52.71%  406.117s       970  418.68ms  92.187ms  819.33ms  ncclAllReduceTreeLLKernel_sum_f16(ncclColl)
                   12.10%  93.2616s    674052  138.36us  85.663us  401.72us  volta_fp16_s884gemm_fp16_128x64_ldg8_f2f_nn
                    7.44%  57.2983s    169372  338.30us  125.31us  33.220ms  volta_fp16_s884gemm_fp16_128x128_ldg8_f2f_nn
                    6.80%  52.3837s    166814  314.02us  198.78us  4.8538ms  volta_fp16_s884gemm_fp16_256x128_ldg8_f2f_nn
                    5.27%  40.5771s     12484  3.2503ms  173.31us  9.1597ms  volta_fp16_s884gemm_fp16_256x128_ldg8_f2f_nt
[root@dss01 rnn_translator]#
```

https://devblogs.nvidia.com/massively-scale-deep-learning-training-nccl-2-4/

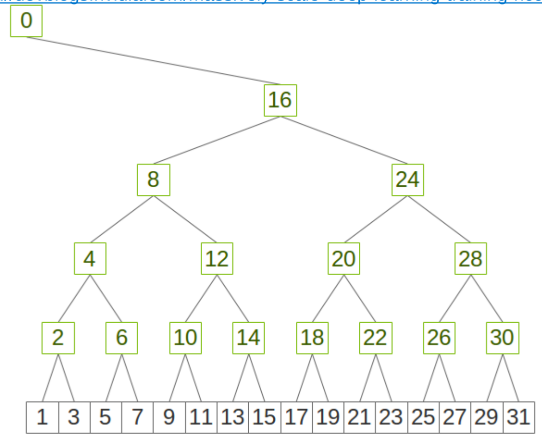


Figure 1. Binary tree using a power-of-two pattern

NCCL_P2P_LEVEL

(since 2.3.4)

The `NCCL_P2P_LEVEL` variable allows the user to finely control when to use the peer to peer (P2P) transport between GPUs. The level defines the maximum distance between GPUs where NCCL will use the P2P transport.

Values accepted

0 : Never use P2P. (always disabled)

1 : Use P2P when GPUs are on the same PCI switch.

2 : Use P2P when GPUs are connected through PCI switches (potentially multiple hops).

3 : Use P2P when GPUs are on the same PCI root complex, potentially going through the CPU.

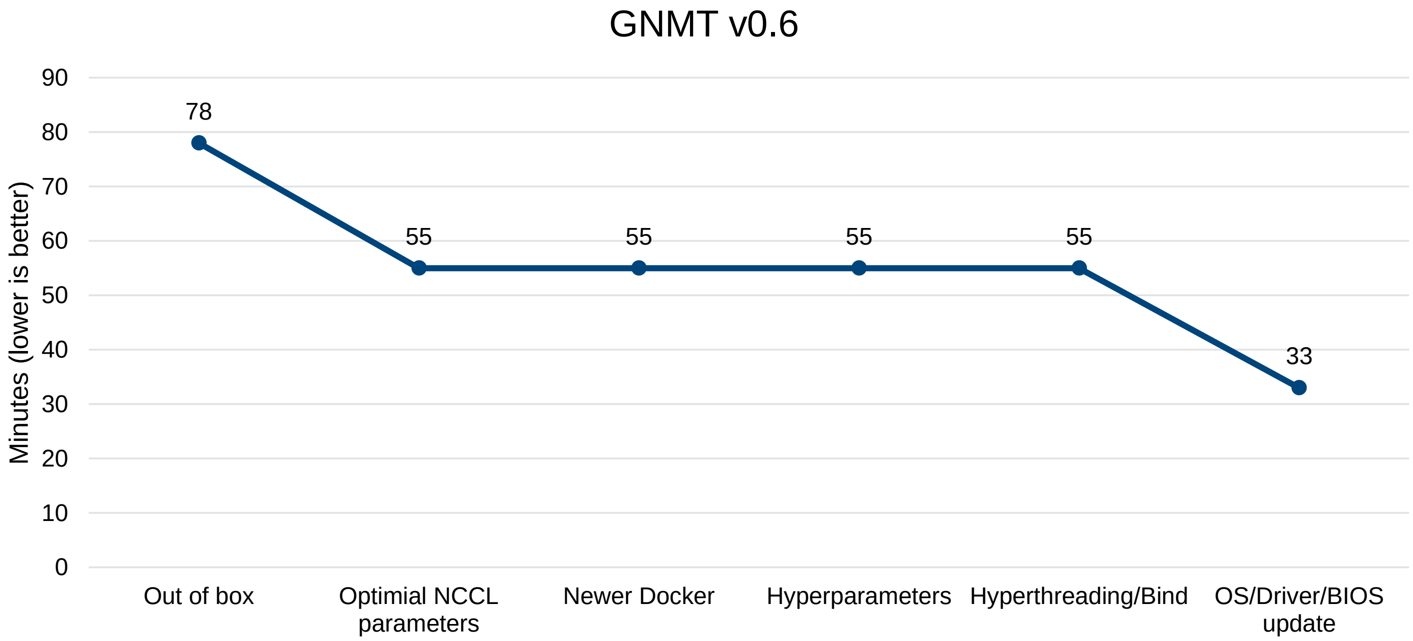
4 : (Since 2.4.7) Use P2P even across PCI root complexes, as long as the GPUs are within the same NUMA node. (Before 2.4.7) Use P2P even across PCI root complexes, regardless of whether the GPUs are within the same NUMA node (always enabled).

5 : Use P2P even across the SMP interconnect between NUMA nodes (e.g., QPI/UPI). (always enabled)

The default value is 3.

https://docs.nvidia.com/deeplearning/sdk/nccl-developer-guide/docs/env.html

GNMT on DSS8440



- NVIDIA docker is optimal based on DGX-1
- PCIe system need fine tune to make sure software matches with hardware topology
- We are able to reduce the training time from 77 min to 33 min on DSS8440
- Reduces overall time to 70%

Tuning Knobs - HW

- CPU cores frequency
  - 6230 2.1 GHz/20cores
  - 6248 2.5 GHz/20cores
  - 6252 2.1 GHz/24cores
- Memory 2666 vs 2933
- PCIe vs NVLink
  - Except GNMT and Transformer ~30%
  - ~10% both single and multiple nodes
  - TDP 250 vs 300 W & Higher frequency
- Storage
  - Local SSD
  - U.2 NVMe
  - Isilon
  - Lustre

- BIOS
  - Custom profile based on HPC workload profile
  - HyperThreading
  - SubNumaCluster
  - ADDDC
- GPUs
  - V100-PCIe/SXM2
  - V100S
  - RTX 6000/8000
- Server
  - C4140
  - DSS8440
  - R740

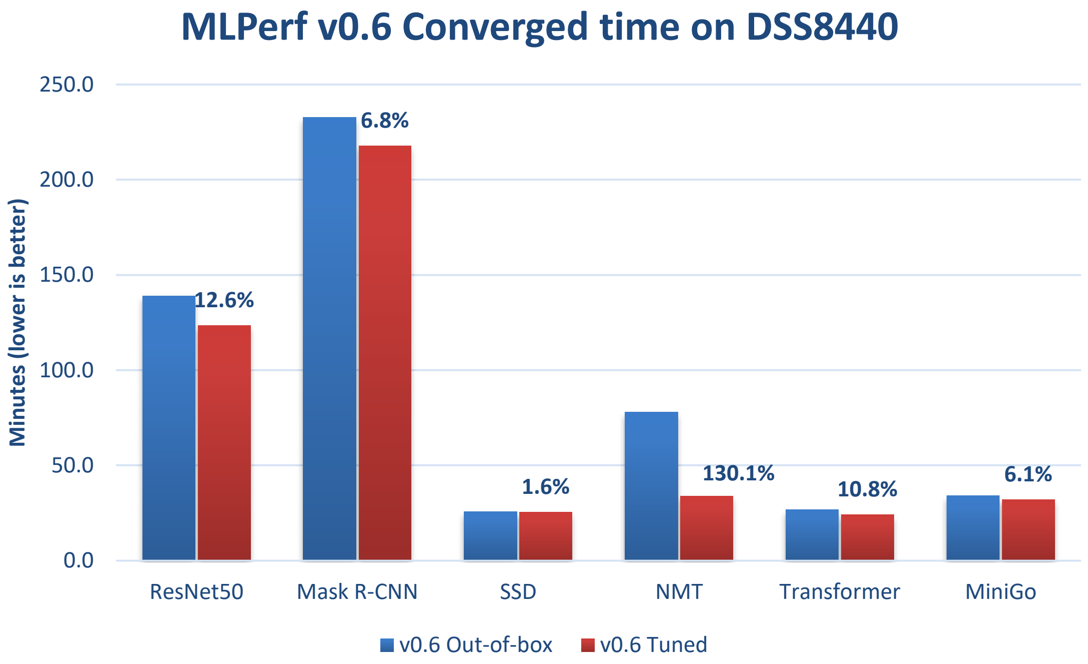
Tuning Knobs - SW

- Docker version
- Binding with CPU cores
- OS
  - Spectre/Meltdown patches
  - Dist/Release/Kernel
- GPU Driver
- CUDA toolkit version
- NCCL
  - Tree vs Ring
  - P2P
- CuDNN
- DALI

- Hyperparameters
  - Batch size
  - Learning rate
  - Etc.

- Frameworks
  - Tensorflow
  - Pytorch
  - MXNet
  - Horovod

MLPerf training v6.0 tuning result



- Haven't try everything on every subtests, some of them applied, and it is good enough for showing the difference before and after tuning

Know when to stop

- Tuning is time consuming
- Set expectation
- Compare with known results from MLPerf website
  - Hyperparameters
  - Converged with less or equal epoch_num
  - Average results
  - Refer to tokens/s, images/s
- Watch GPU utilization >90 TDP > 200/250 W
- Profiling
- Run with multiple systems will help to speed up

Suggestions for single node tuning

- Run on latest OS or newer kernel
- Use NCCL matching your hardware layout
- Use new CUDA libraries
  - Improved performance
  - Additional parameters
- Explorer settings in Dockerfile, run.sub, run_and_time, config_DGX1.sh, compare with DGX2's
- Use submitted results files as reference

Build docker with latest libraries

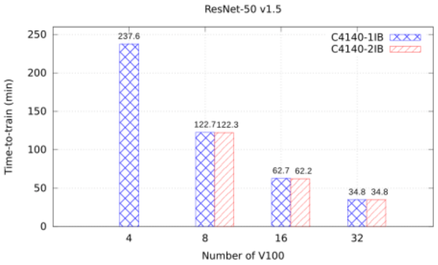
```
[root@node009 gnmt]# cat Dockerfile
ARG FROM_IMAGE_NAME=nvcr.io/nvidia/pytorch:19.05-py3
FROM ${FROM_IMAGE_NAME}

# Install dependencies for system configuration logger
RUN wget https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1604/x86_64/nvidia-machine-learning-repo-ubuntu1604_1.0.0-1_amd64.deb && dpkg -i nvidia-machine-learning-repo-ubuntu1604_1.0.0-1_amd64.deb

RUN apt-get update && apt-get install -y --no-install-recommends \
        infiniband-diags \
        pciutils \
        libnccl2 libnccl-dev libcudnn7 libcudnn7-dev && \
    rm -rf /var/lib/apt/lists/* #&& rm nvidia-machine-learning-repo-ubuntu1604_1.0.0-1_amd64.deb


# Rebuild PyTorch
WORKDIR /opt/pytorch
RUN cd pytorch && \
TORCH_CUDA_ARCH_LIST="5.2 6.0 6.1 7.0 7.5+PTX" \
CMAKE_PREFIX_PATH="$(dirname $(which conda))/../" \
NCCL_INCLUDE_DIR="/usr/include/" \
NCCL_LIB_DIR="/usr/lib/" \
python setup.py install && python setup.py clean

# Install Python dependencies
WORKDIR /workspace/rnn_translator

COPY requirements.txt .
RUN pip install --no-cache-dir https://github.com/mlperf/training/archive/6289993e1e9f0f5c4534336df83ff199bd0cdb75.zip#subdirectory=compliance \
 && pip install --no-cache-dir -r requirements.txt

# Copy & build extensions
COPY seq2seq/csrc seq2seq/csrc
COPY setup.py .
RUN pip install .

# Copy GNMT code
COPY . .

# Configure environment variables
ENV LANG C.UTF-8
ENV LC_ALL C.UTF-8
```

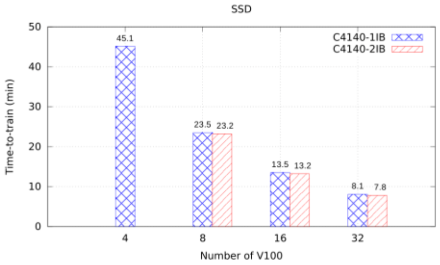
Multiple nodes

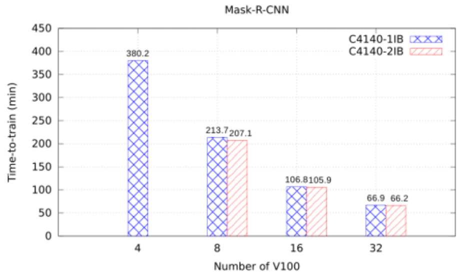
MLPerf multiple nodes training

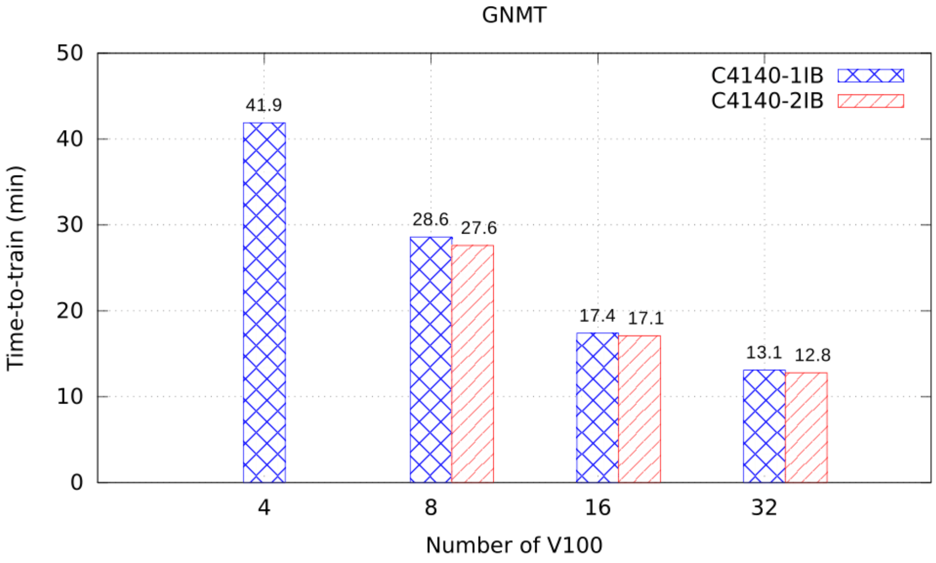
- Not easy to use Docker on multi-node with InfiniBand
  - The InfiniBand driver version within Docker container may not match the version on the host
  - The docker container may not update the IB driver automatically
- Solution:
  - convert the base Docker container into Singularity container
  - build MLPerf benchmarks within Singularity container
  - run Singularity container with Slurm

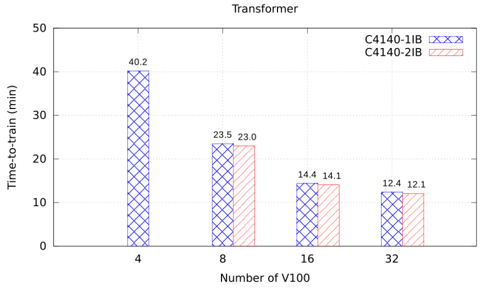
MLPerf Training on C4140





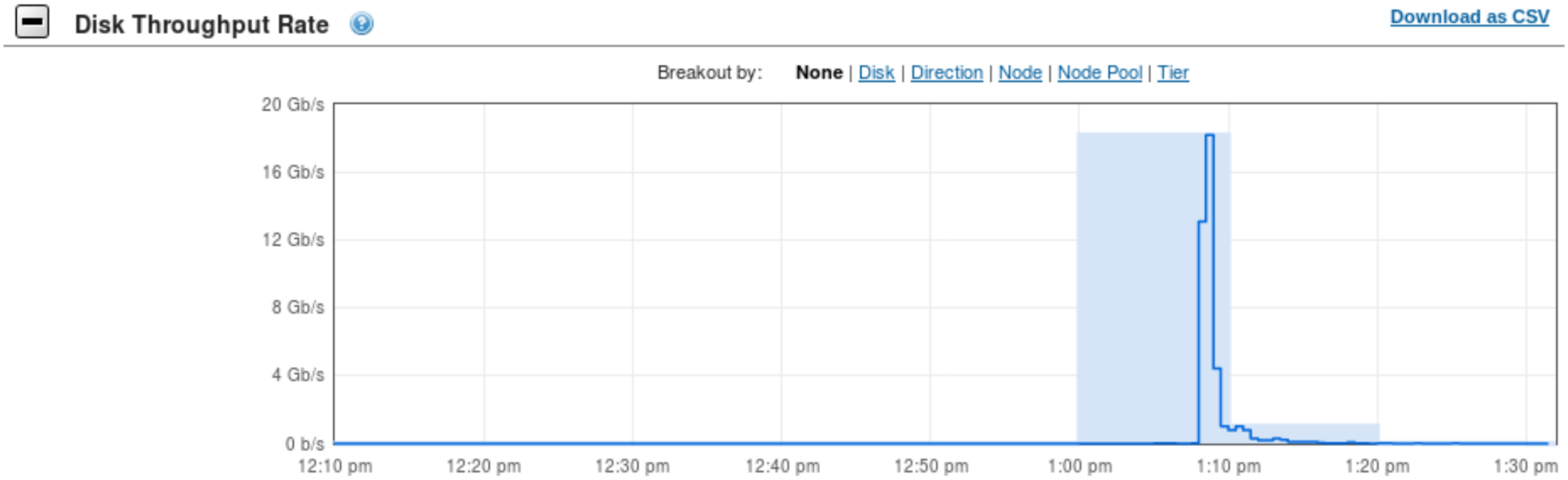




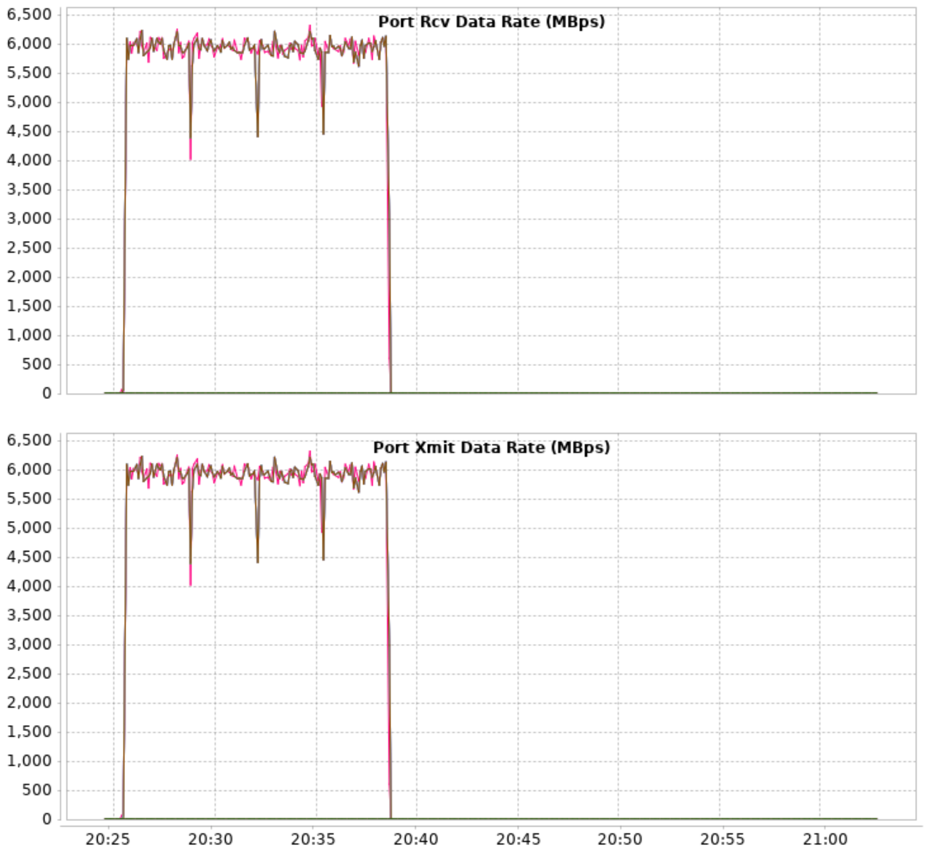


- Model parallelism requires higher bandwidth than data parallelism
- 2 IB adapters doesn't help a lot for performance improvement

Storage Profiling



Network Profiling



DELL EMC

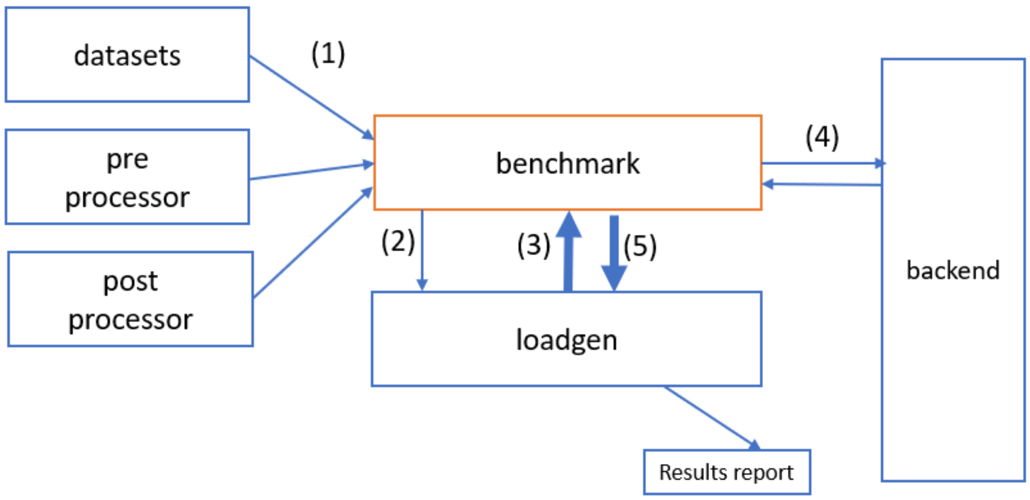
Storage and Network Profiling

		ResNet-50	SSD	Mask-R-CNN	GNMT	Transformer
Storage (Gb/s)	2 nodes	7.4	3.9	0.2	0.19	0.26
	4 nodes	16	3.1	0.42	0.28	0.34
	8 nodes	**18**	4.8	1.2	0.3	0.26
Network (Gb/s)	2 nodes	3.8	1.4	4.4	12	11.6
	4 nodes	4	4.8	7.6	36	24.8
	8 nodes	4.2	7.6	7.2	**48**	**24.8**

DELL EMC

Inference v0.5

MLPerf Inference v0.5 Benchmark



1 – benchmark knows dataset
2 – benchmark hands content id's to loadgen
3 – loadgen starts generating queries
4 – benchmark creates requests to backend
5 – benchmark post processes response and completes query
6 – after all queries finished, loadgen writes report

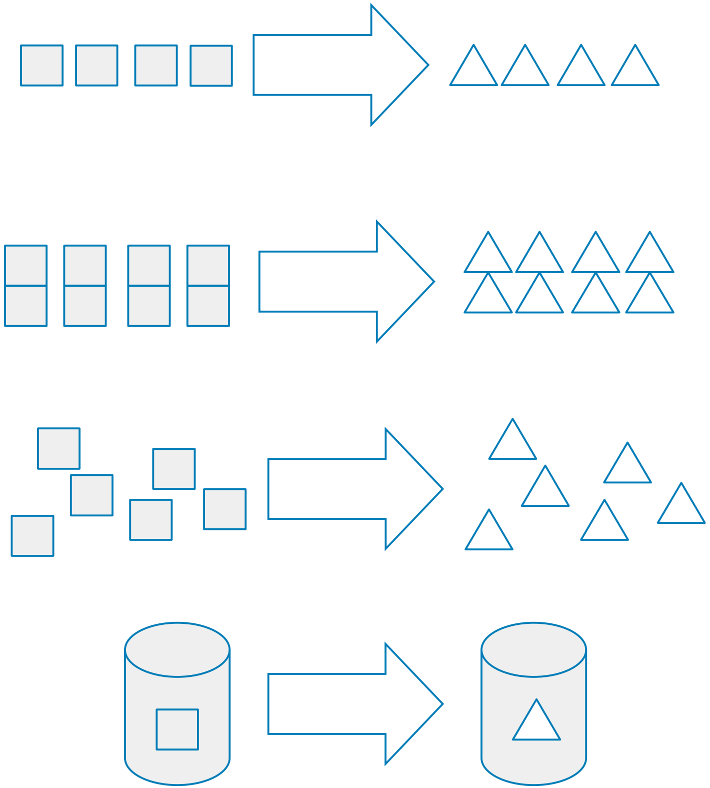
Source: https://arxiv.org/pdf/1911.02549.pdf

MLPerf Inference v0.5

Area	Task	Model	Dataset
Vision	Image classification	ResNet50-v1.5	ImageNet (224x224)
Vision	Image classification	MobileNets-v1 224	ImageNet (224x224)
Vision	Object detection	SSD-ResNet34	COCO (1200x1200)
Vision	Object detection	SSD-MobileNets-v1	COCO (300x300)
Language	Machine translation	GNMT	WMT16

Source: https://arxiv.org/pdf/1911.02549.pdf

MLPerf Inference v0.5



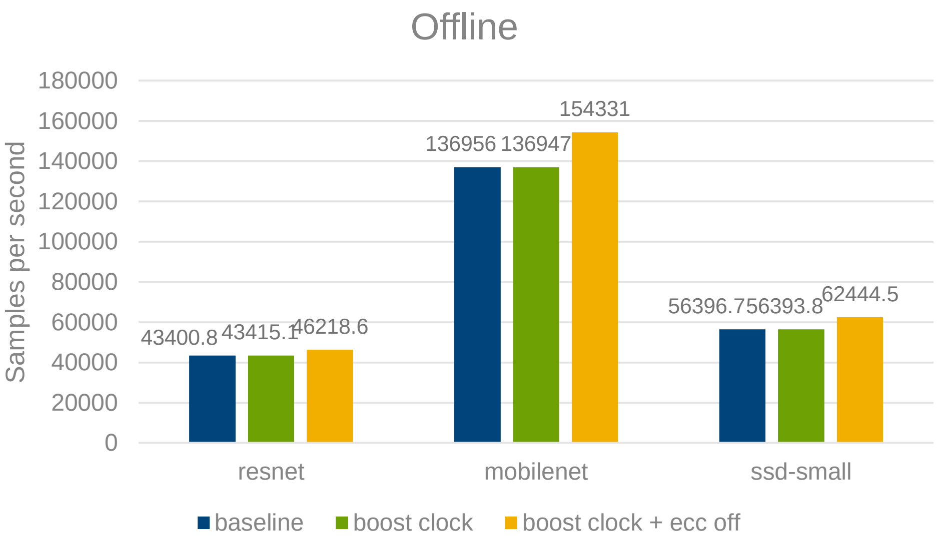
Single stream e.g. cell phone augmented vision	**Latency**
Multiple stream e.g. multiple camera driving assistance	**Number streams** subject to latency bound
Server e.g. translation site	**QPS** subject to latency bound
Offline e.g. photo sorting	**Throughput**

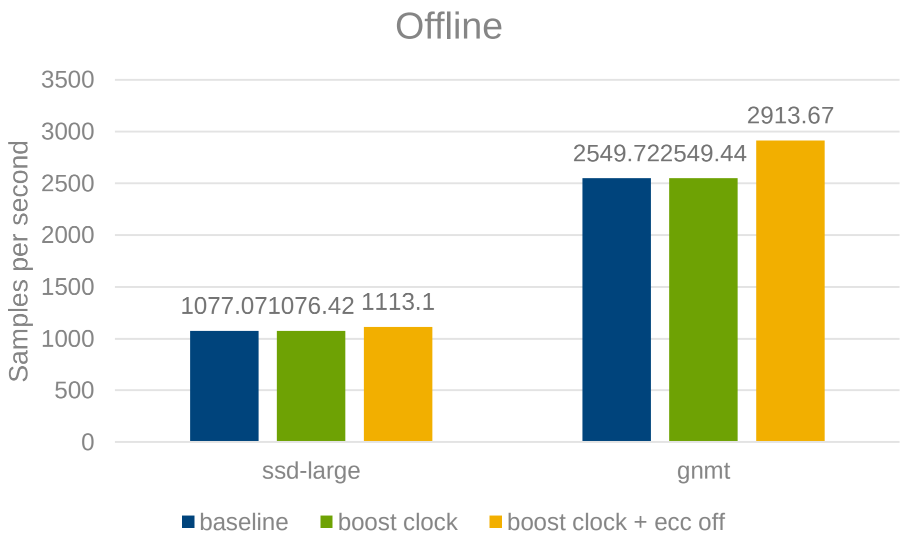
Source: https://arxiv.org/pdf/1911.02549.pdf

MLPerf Inference v0.5

- Test Server: R740/C4140
- GPUs: 4x T4/ 3x RTX6000 /4x V100-SXM2 16GB
- Inference Backend: TensorRT 6.0
- Benchmarks:
  - ResNet50-v1.5
  - MobileNet
  - SSD-MobileNets-v1
  - SSD-ResNet34
  - GNMT
- Inference Scenarios: Server, Offline

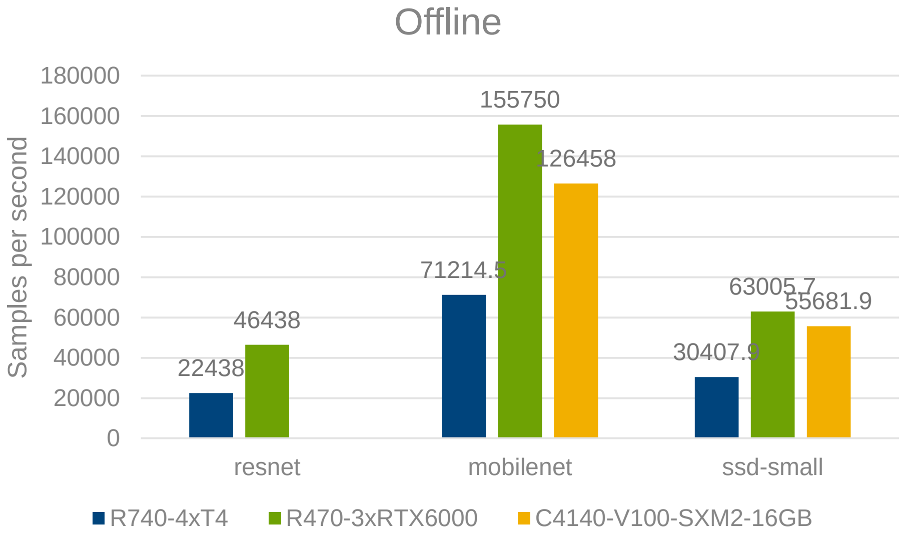
MLPerf Inference v0.5

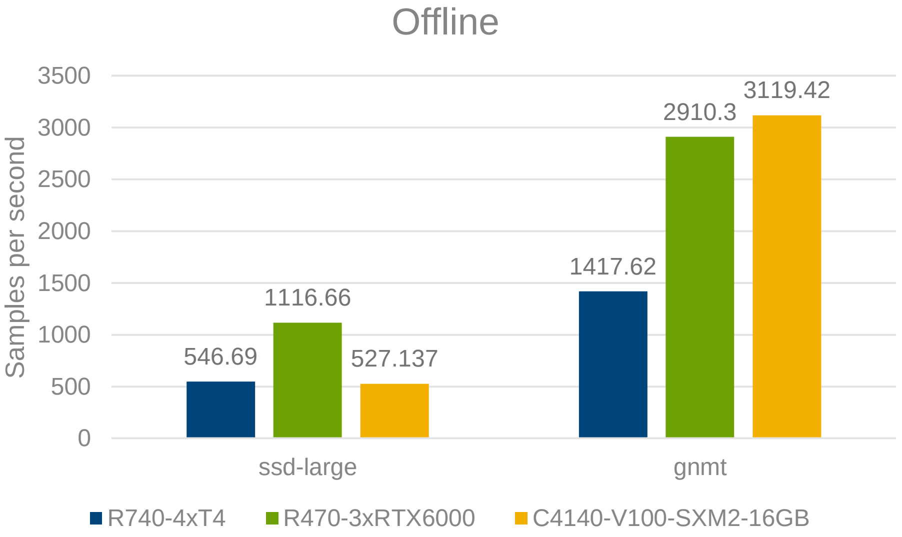




- Tested with 3x RTX 6000
  - Boost clock does not improve the performance
  - ECC off can improve the performance by 3.41% to 14.29%

MLPerf Inference v0.5





- RTX is 2.7x – 2.9x faster than T4 per GPU
- RTX is 2.8x faster than V100 per GPU for ssd-large, and 1.2x – 1.6x for other models

MLPerf Inference v0.5

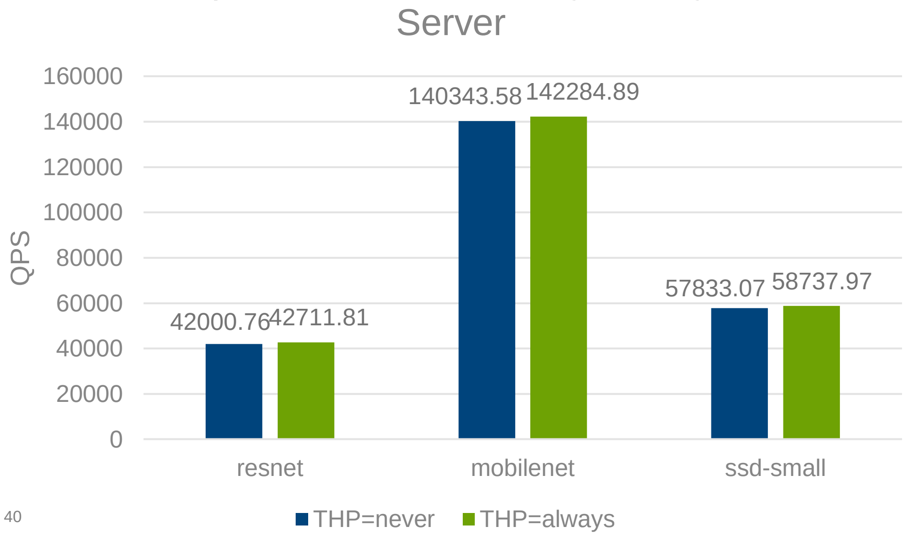
- The goal for Server scenario: find the maximum QPS subject to latency bound.
- Search strategy: binary search. The initial left boundary is valid QPS, the initial right boundary is invalid QPS.
- while(left < right){

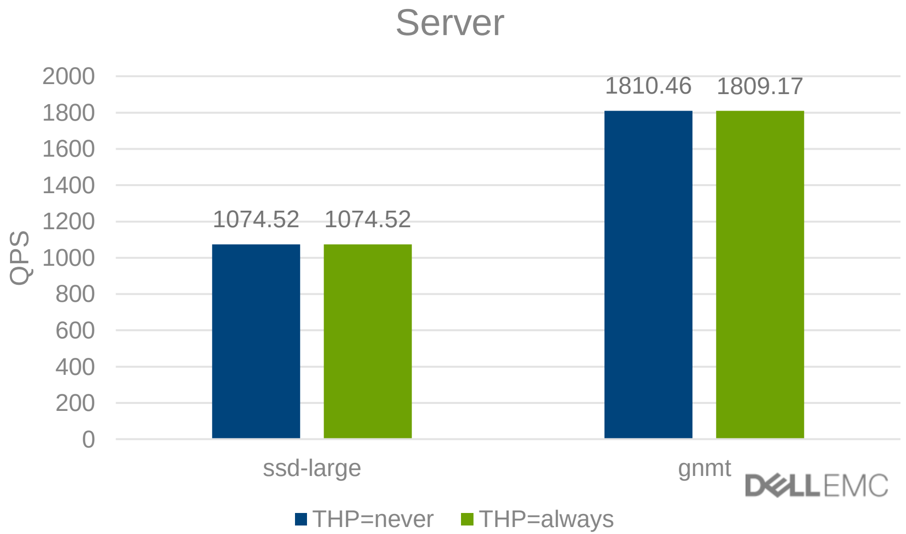
```
    mid = left + (right-left)/2;
    result = benchmark(system_id, model, Server scenario, qps=mid)
    if(result == "VALID")
            left = mid + 1;
    else
            right = mid;
```

- }
- right – 1 is the maximum QPS when the result is VALID

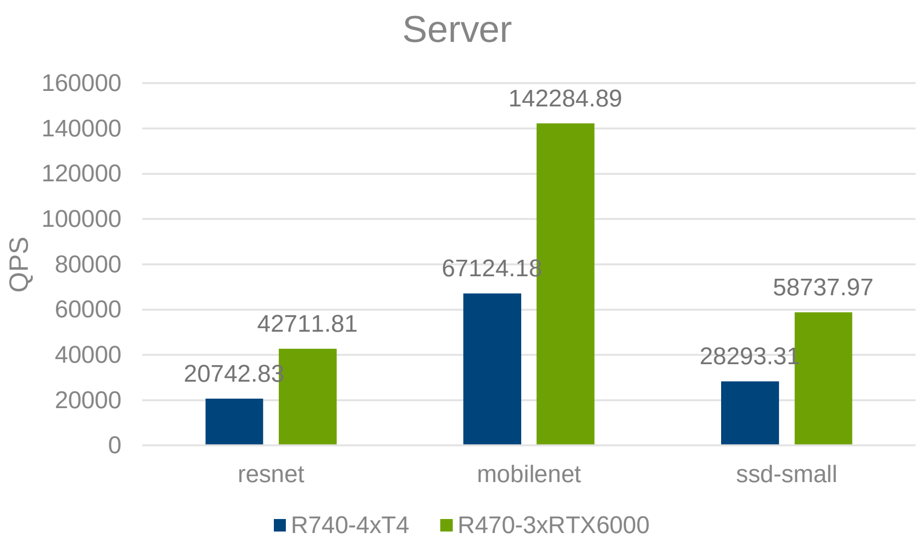
MLPerf Inference v0.5

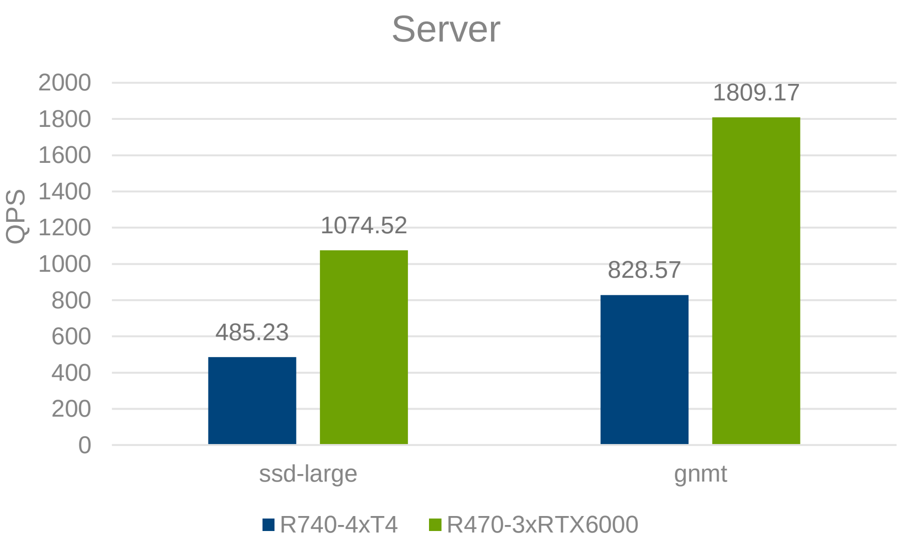
- Use jemalloc: scalable concurrency support and reducing memory fragmentation
  - LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libjemalloc.so.2
- Use Transparent Huge Pages (THP)
  - Performance improvement of 1.69%, 1.38%, and 1.56% for resnet, mobilenet, and ssd-small
  - No improvement for ssd-large and gnmt





MLPerf Inference v0.5





- RTX is 2.7x – 2.9x faster than T4 per GPU

Conclusions

- Training
  - Docker is not easy to use on multi-node with InfiniBand, an alternate is to use Singularity container
  - The performance scales well for ResNet-50 v1.5, SSD and Mask-R-CNN.
  - The machine translation models (GNMT and Transformer) have (or need) high network throughput.
  - Dual IB does not have significant performance improvement.
- Inference
  - Boost clock does not improve the performance
  - ECC off can improve the performance obviously
  - jemalloc library makes the Server scenario performance more stable
  - Transparent Huge Pages (THP) provide performance benefits for Server scenario
  - Binary search can search the right QPS for Server scenario efficiently

DELL EMC