AI CONFERENCE

Multi GPU Training

Hyungon Ryu (유현곤 부장)
Sr. Solutions Architect at NVIDIA

DGX SuperPOD

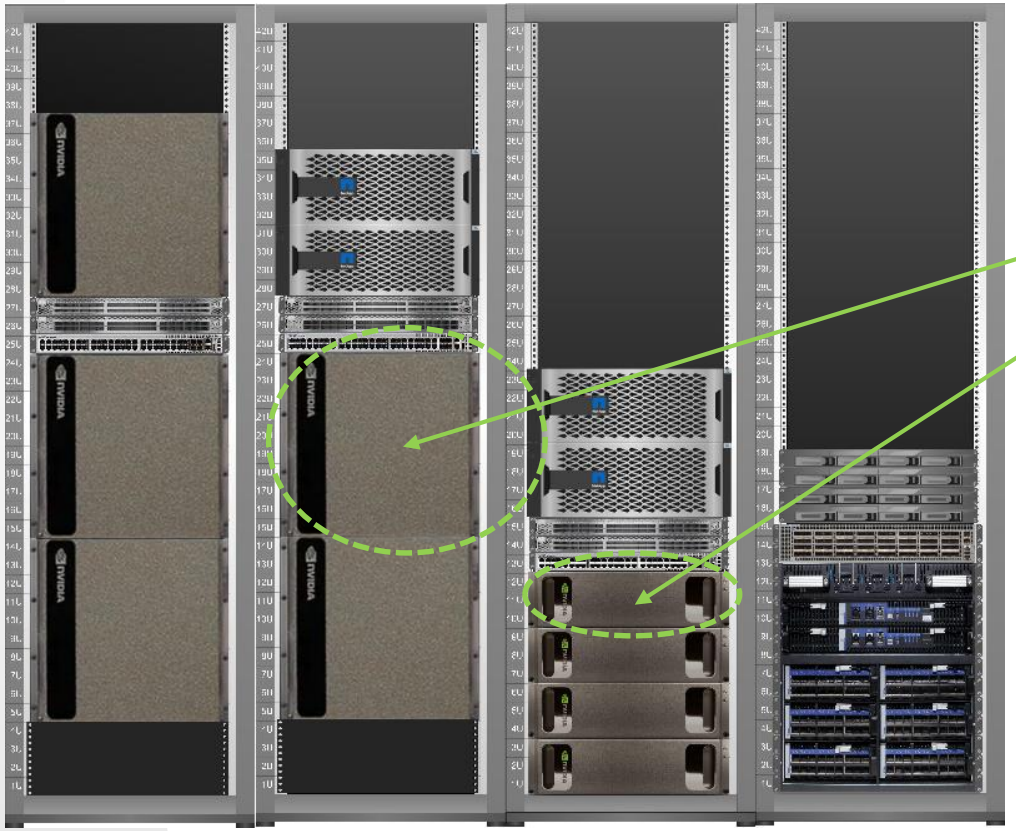
- 22 Rank TOP500.org
- 96 node DGX-2H
- 1536 Tesla V100
- 11.2PFLOPS (DP)
- 144 PFLOPS(TensorCore)

Rmax (TFlop/s)	Rpeak (TFlop/s)
9,444,	11,209,

https://devblogs.nvidia.com/dgx-superpod-world-record-supercomputing-enterprise/

Saturn V - DGX POD

제가 사용한 시스템



Allocated Resources are
- DGX-2
- DGX-1 (16GB)

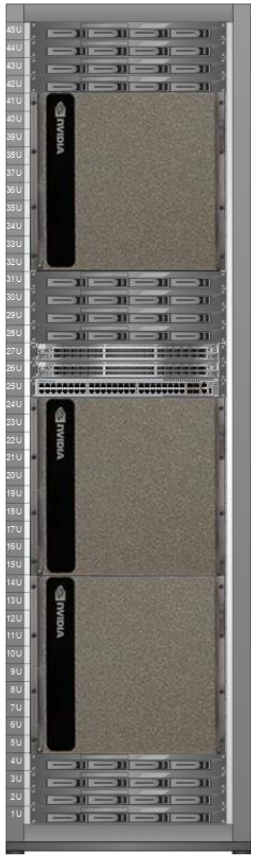
Using NGC Docker Images with custom build for utility

DGX Data Center Reference Design Whitepaper

DGX-2 POD (35KW) and DGX-1 POD (18KW)

Fit within a standard-height 42RU data center rack, 35kW

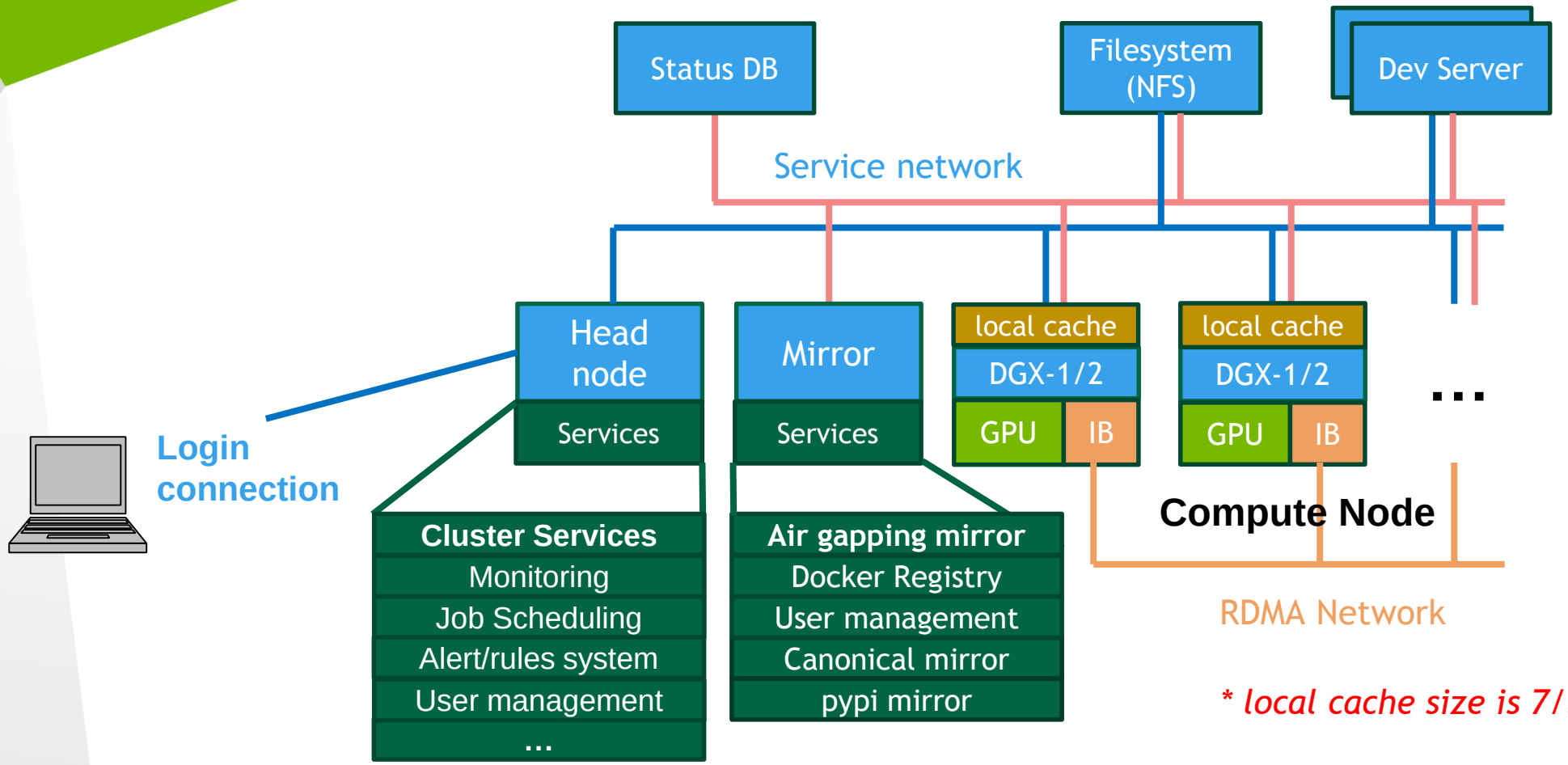
- **Three DGX-2 servers (3 x 10 RU = 30 RU)**
- **Twelve storage servers (12 x 1 RU = 12 RU)**
- **10 GbE (min) storage and management switch (1 RU)**
- **Mellanox 100 Gbps intra-rack high speed network switches (1 or 2 RU)**



Fit within a standard-height 42RU data center rack, 18kW

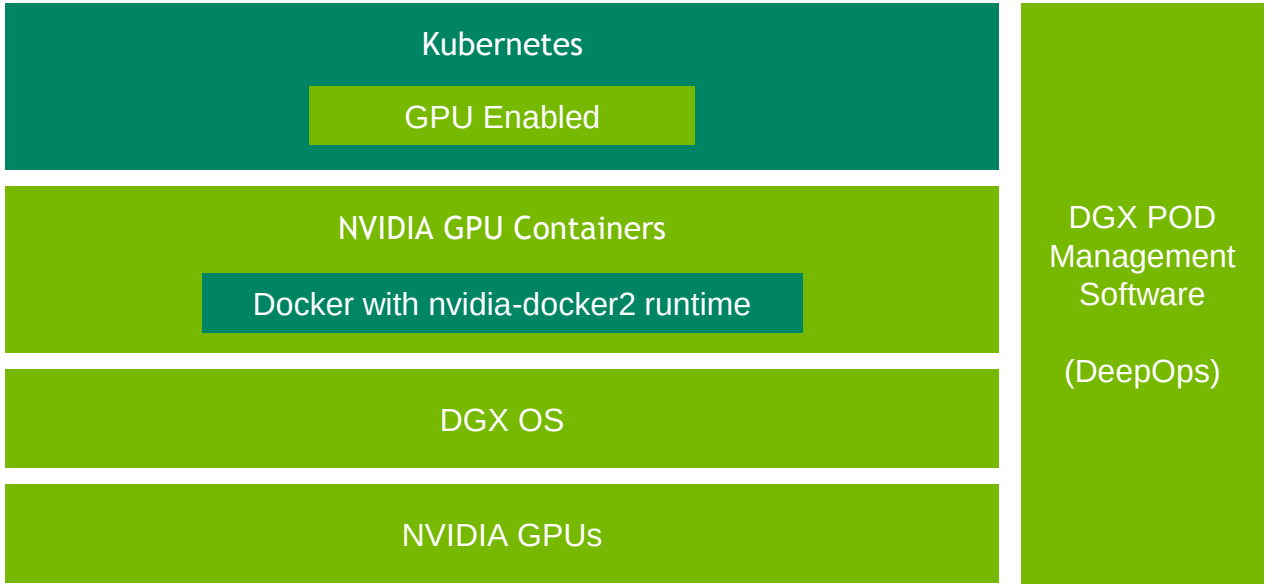
- **Four** DGX-1 servers (4 x 3 RU = 12 RU)
- **Six** storage servers (6 x 1 RU = 6 RU)
- 10 GbE (min) storage and management switch (1 RU)
- Mellanox 100 Gbps intra-rack high speed network switches (1 or 2 RU)

MultiNode DGX-1/DGX-2



** local cache size is 7/30TB each*

DeepOps



- Scale-up to thousands of GPUs
- Self-healing cluster orchestration
- GPU optimized out-of-the-box

https://github.com/NVIDIA/deepops

Perf. in DGX-2

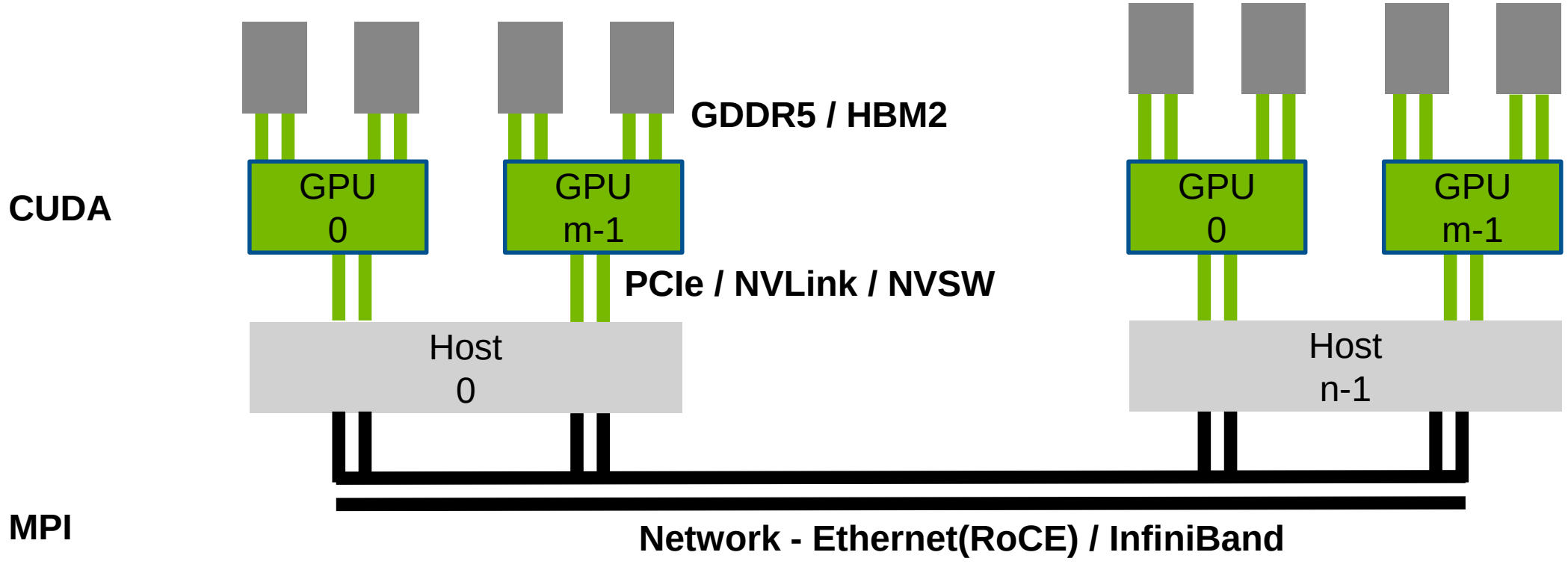
DGX-2 have 2 PetaFLOPS

..as One Giant GPU
81,920 CUDA Cores
512 GB HBM2

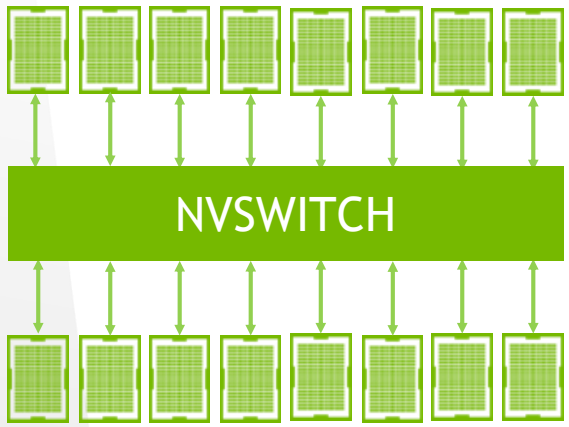
Tesla V100
5,120 CUDA Cores
32GB HBM2

x 16

multiGPU/multiNode



Comm. In DGX-2



6x NVLINK lanes

All 96 lanes are Non Blocking Fat Tree

16 GPUs Fully connected
2.4TB/s of bi-section bandwidth

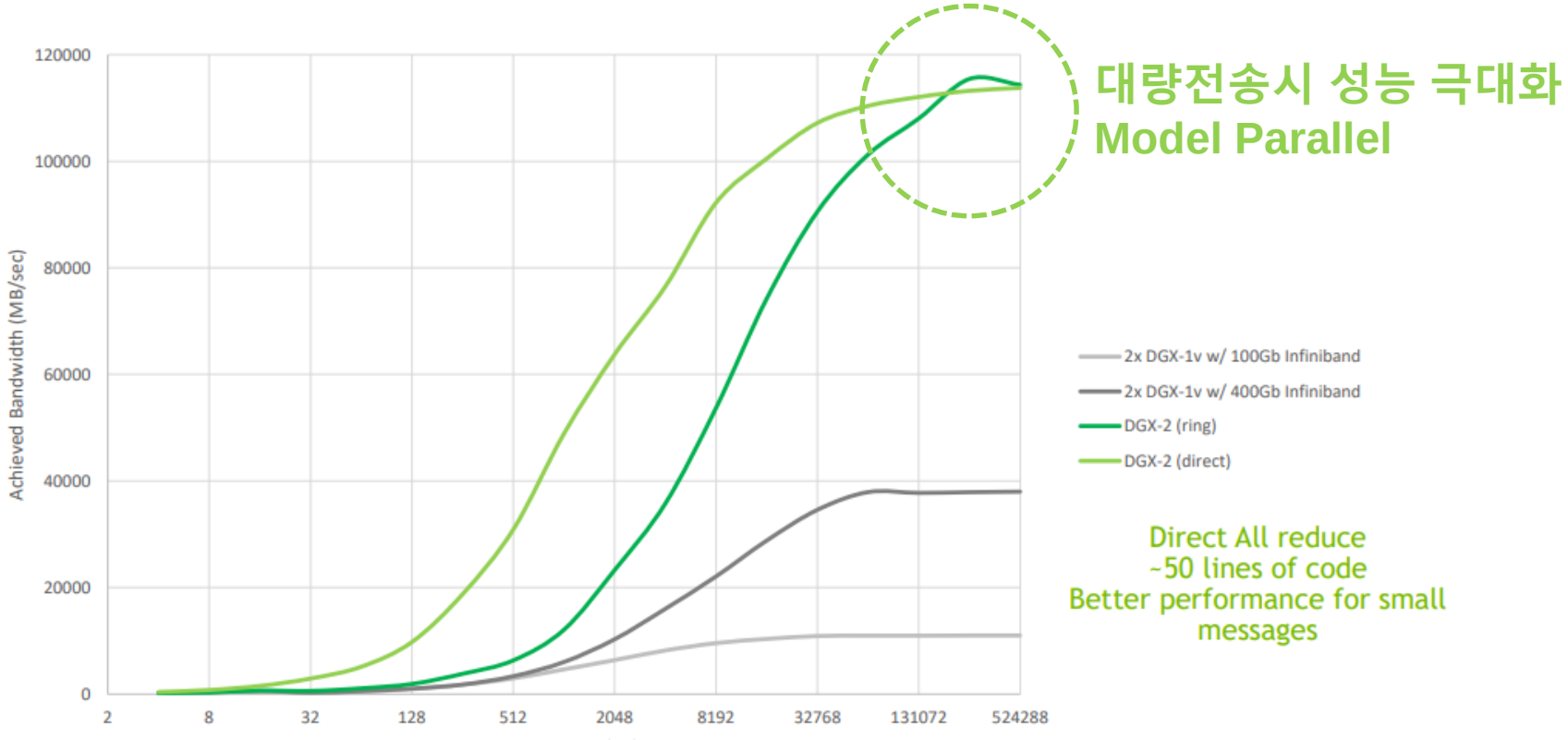
```
$ nvidia-smi topo -m
```

	G0	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14	G15	CPU Affinity
GPU0	X	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	0-23,48-71
GPU1	NV6	X	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	0-23,48-71
GPU2	NV6	NV6	X	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	0-23,48-71
GPU3	NV6	NV6	NV6	X	NV6	NV6	NV6	NV6	NV6	NV6							0-23,48-71
GPU4	NV6	NV6	NV6	NV6	X	NV6	NV6	NV6	NV6	NV6							0-23,48-71
GPU5	NV6	NV6	NV6	NV6	NV6	X	NV6	NV6	NV6	NV6							0-23,48-71
GPU6	NV6	NV6	NV6	NV6	NV6	NV6	X	NV6	NV6	NV6							0-23,48-71
GPU7	NV6	NV6	NV6	NV6	NV6	NV6	NV6	X	NV6	NV6							0-23,48-71
GPU8	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	X	NV6							24-47,72-95
GPU9	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	X							24-47,72-95
GPU10	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	X	NV6	NV6	NV6	NV6	NV6	24-47,72-95
GPU11	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	X	NV6	NV6	NV6	NV6	24-47,72-95
GPU12	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	X	NV6	NV6	NV6	24-47,72-95
GPU13	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	X	NV6	NV6	24-47,72-95
GPU14	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	X	NV6	24-47,72-95
GPU15	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	X	24-47,72-95

```
$ nvidia-smi nvlink -s
GPU 0: Tesla V100-SXM3-32GB
    Link 0: 25.781 GB/s
    Link 1: 25.781 GB/s
    Link 2: 25.781 GB/s
    Link 3: 25.781 GB/s
    Link 4: 25.781 GB/s
    Link 5: 25.781 GB/s
```

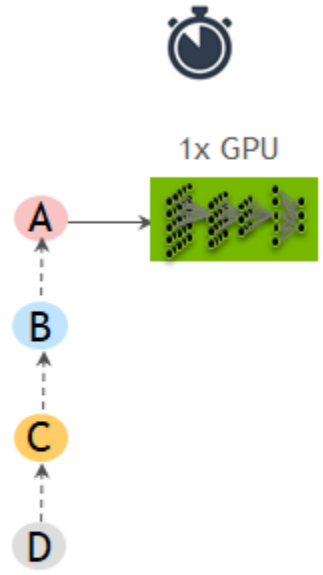
6. NV Switch Benefit
ALL Reduce Benchmark

Deep Learning의 파라미터 전송에 사용됨

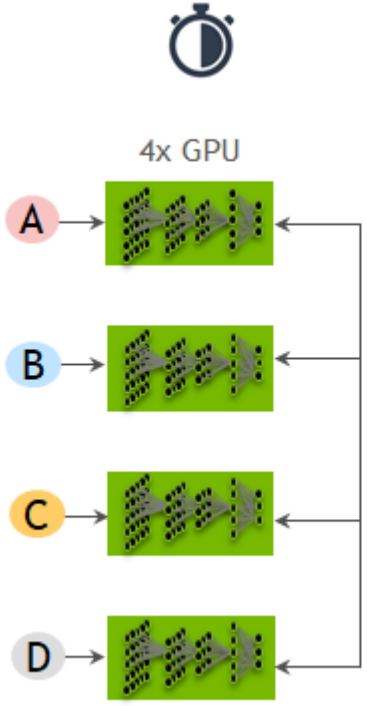


Distributed Deep Learning

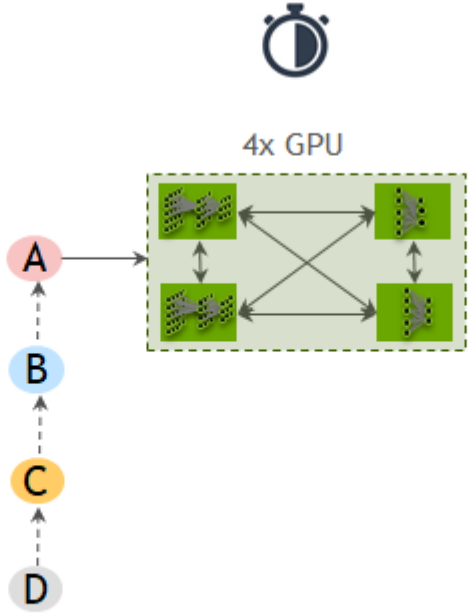
SINGLE GPU



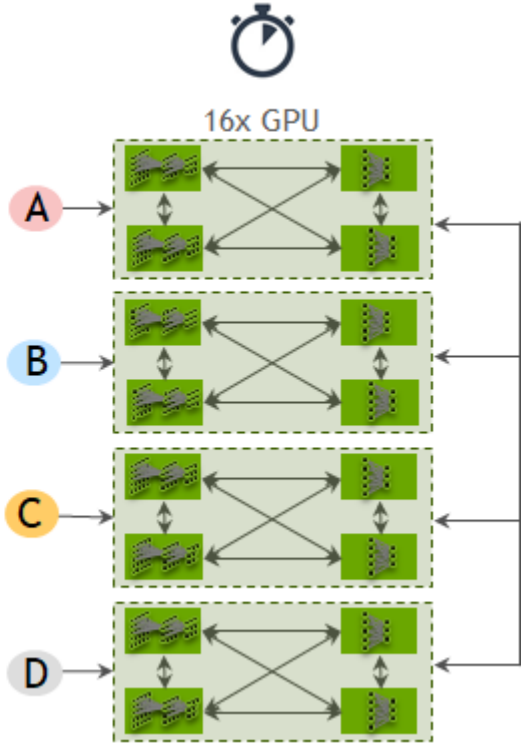
DATA PARALLEL



MODEL PARALLEL



DATA AND MODEL PARALLEL



병렬 101 수치해석 병렬 PI계산(적분법)

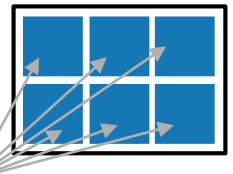
$$\int_0^x \frac{1}{1+x^2}dx = \arctan x$$

$$\int_0^1 \frac{1}{1+x^2}dx = \arctan 1 = \frac{\pi}{4}$$

$$\left(\because \tan\frac{\pi}{4} = 1\right)$$

N 코어만큼
나눠서 계산하고
최종 결과는
각 코어에서 reduction

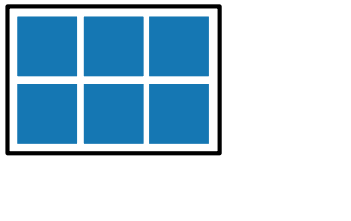
CPU

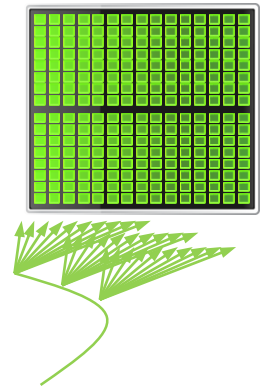


```
main() {
  double pi = 0.0; long i;

  #pragma omp parallel for reduction(+:pi)
  for (i=0; i<N; i++)
  {
    double t = (double)((i+0.05)/N);
    pi += 4.0/(1.0+t*t);
  }

  printf("pi = %f\n", pi/N);
}
```

OpenMP

CPU **Parallel Hardware**





```
main() {
  double pi = 0.0; long i;

  #pragma acc kernels
  for (i=0; i<N; i++)
  {
    double t = (double)((i+0.05)/N);
    pi += 4.0/(1.0+t*t);
  }

printf("pi = %f\n", pi/N);
}
```

OpenACC

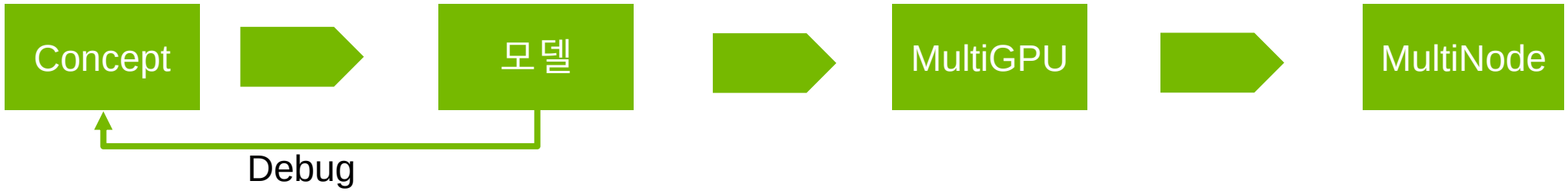
DL 학습 병렬화 전략

Concept → 모델 → MultiGPU → MultiNode

GPU 병렬화 되어있음

Tensorflow
Horovod
APEX/AMP on Pytorch
MXNET
Chainer
C2

DL 학습 병렬화 전략



Step1.

아이디어
모델로 구현
수많은(?) 시행착오 이후
코드 완성

Step2.

완성된 모델로
연구 진행

Step3.

연구 진행시
성능이슈 발생시
multiGPU 병렬화 시도

DL 학습 병렬화 전략

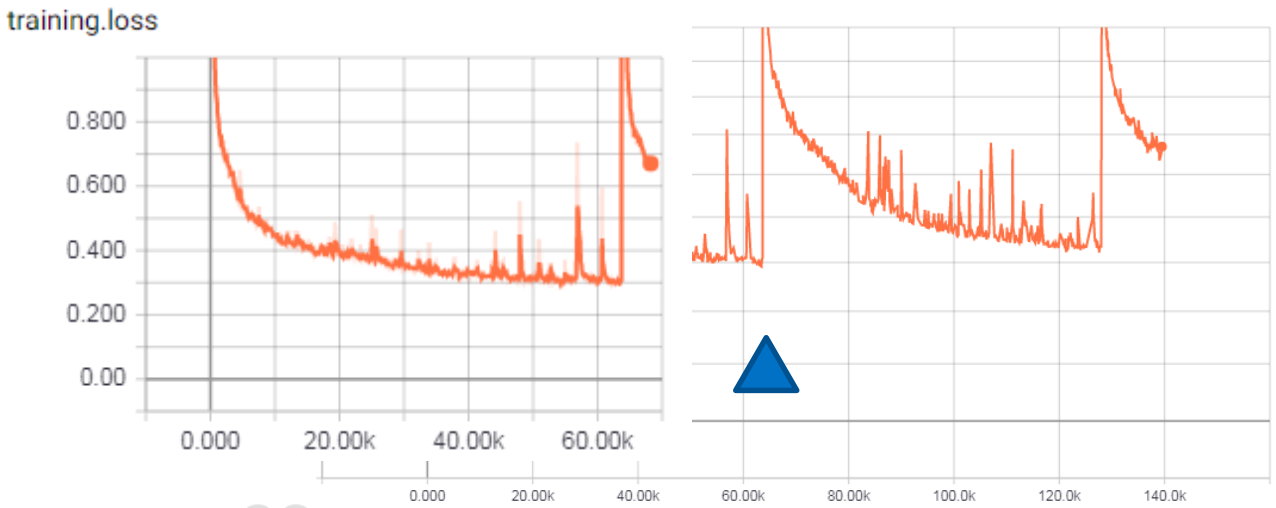
Concept → 모델 → MultiGPU → MultiNode

Step2. 연구 진행시

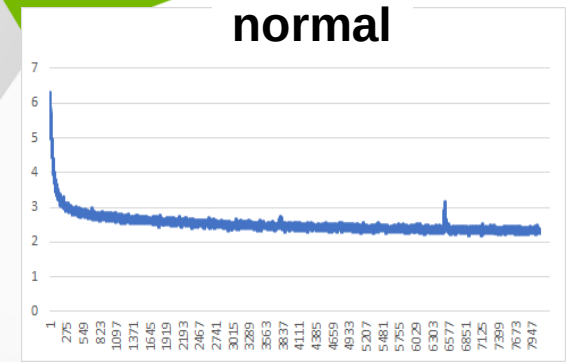
많은 데이터로 인하여 실연구 성과 확인에 시간이 오래 소요(TAT)

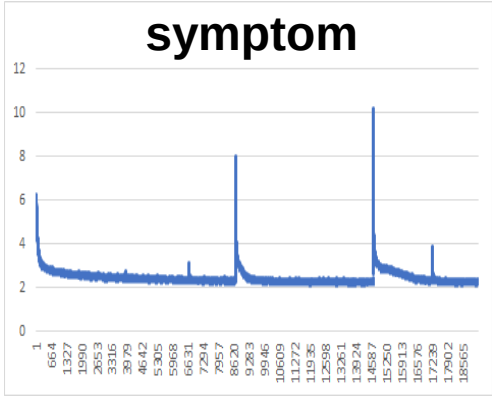
Size	Used	Avail	Use%	Mounted on
2.6T	875G	1.8T	34%	/home/scratch.hryu_sw

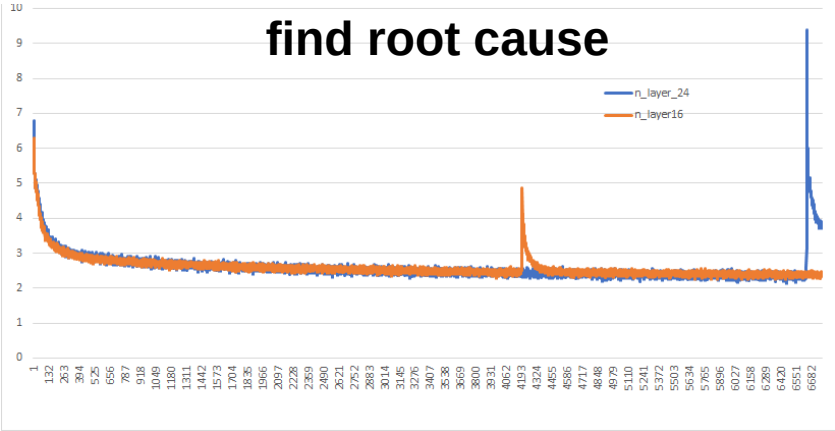
16일 차 학습 이후 loss 터지는 문제 발견

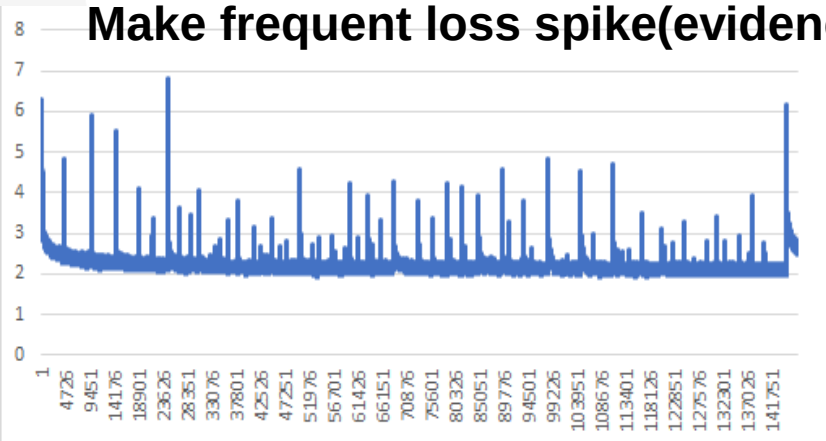


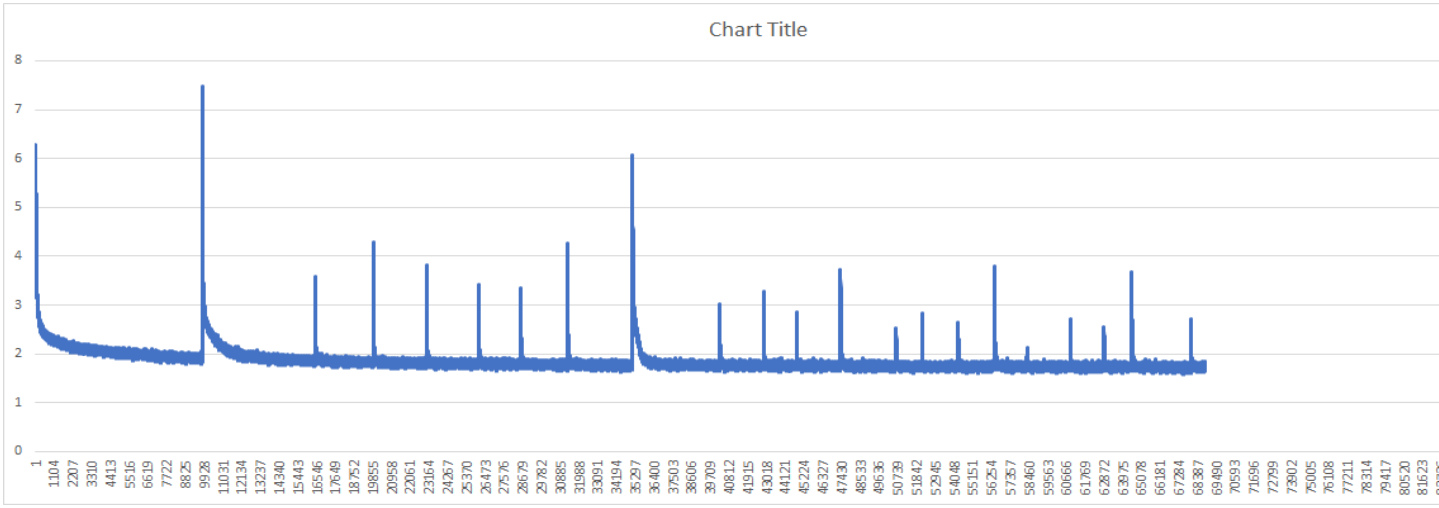
More 700 GPU hours to debug one issue











DL 학습 병렬화 전략

Concept → 모델 → MultiGPU → MultiNode

Step3. 병렬화 진행시

이슈1. First GPU allocate Large Memory

```
| N/A   46C    P0   103W / 350W |  29200MiB / 32510MiB
| N/A   44C    P0    82W / 350W |  11244MiB / 32510MiB
| N/A   52C    P0    87W / 350W |  11244MiB / 32510MiB
```

이슈2. GPU idle

```
| N/A   48C    P0   273W / 350W |  19492MiB / 32510MiB |    100%      Default |
| N/A   47C    P0   252W / 350W |  16434MiB / 32510MiB |    100%      Default |
| N/A   56C    P0   301W / 350W |  16434MiB / 32510MiB |    100%      Default |
| N/A   48C    P0   313W / 350W |  16434MiB / 32510MiB |    100%      Default |
| N/A   42C    P0    83W / 350W |  19492MiB / 32510MiB |      0%      Default |
| N/A   41C    P0    81W / 350W |  16434MiB / 32510MiB |      0%      Default |
| N/A   49C    P0    87W / 350W |  16434MiB / 32510MiB |      0%      Default |
| N/A   48C    P0    85W / 350W |  16434MiB / 32510MiB |      0%      Default |
| N/A   41C    P0    83W / 350W |  19492MiB / 32510MiB |      0%      Default |
| N/A   40C    P0    81W / 350W |  16434MiB / 32510MiB |      0%      Default |
| N/A   48C    P0    86W / 350W |  16434MiB / 32510MiB |      0%      Default |
| N/A   47C    P0    84W / 350W |  16434MiB / 32510MiB |      0%      Default |
| N/A   40C    P0   179W / 350W |  19492MiB / 32510MiB |      0%      Default |
```

이슈3. Use MultiGPU but no speed up.

```
nvidia-smi -i 0 -l 1 | grep MiB
| N/A   42C    P0    64W / 163W |  12634MiB / 32480MiB |      2%      Default |
| N/A   42C    P0    66W / 163W |  14430MiB / 32480MiB |    100%      Default |
| N/A   43C    P0   117W / 163W |  13156MiB / 32480MiB |     95%      Default |
| N/A   45C    P0   160W / 163W |  13156MiB / 32480MiB |     99%      Default |
| N/A   43C    P0    65W / 163W |  12388MiB / 32480MiB |      4%      Default |
| N/A   43C    P0    65W / 163W |  10606MiB / 32480MiB |      4%      Default |
| N/A   42C    P0    48W / 163W |   8656MiB / 32480MiB |      7%      Default |
| N/A   41C    P0    48W / 163W |   7232MiB / 32480MiB |      2%      Default |
| N/A   41C    P0    48W / 163W |   6622MiB / 32480MiB |      2%      Default |
| N/A   41C    P0    47W / 163W |   5956MiB / 32480MiB |      2%      Default |
| N/A   41C    P0    48W / 163W |   5332MiB / 32480MiB |      2%      Default |
| N/A   41C    P0    47W / 163W |   5039MiB / 32480MiB |      5%      Default |
| N/A   41C    P0    47W / 163W |   4068MiB / 32480MiB |      3%      Default |
| N/A   41C    P0    47W / 163W |   3370MiB / 32480MiB |      3%      Default |
| N/A   41C    P0    64W / 163W |   2712MiB / 32480MiB |      4%      Default |
| N/A   41C    P0    63W / 163W |   3444MiB / 32480MiB |      3%      Default |
| N/A   41C    P0    65W / 163W |   4178MiB / 32480MiB |      3%      Default |
| N/A   42C    P0    62W / 163W |   4938MiB / 32480MiB |      3%      Default |
| N/A   41C    P0    63W / 163W |   5674MiB / 32480MiB |      3%      Default |
| N/A   41C    P0    64W / 163W |   6348MiB / 32480MiB |      2%      Default |
| N/A   41C    P0    62W / 163W |   6968MiB / 32480MiB |      2%      Default |
| N/A   41C    P0    62W / 163W |   7588MiB / 32480MiB |      2%      Default |
| N/A   41C    P0    62W / 163W |   8234MiB / 32480MiB |      3%      Default |
| N/A   41C    P0    63W / 163W |   8960MiB / 32480MiB |      3%      Default |
| N/A   41C    P0    64W / 163W |   9678MiB / 32480MiB |      3%      Default |
| N/A   41C    P0    63W / 163W |  10384MiB / 32480MiB |      3%      Default |
| N/A   41C    P0    62W / 163W |  11102MiB / 32480MiB |      2%      Default |
| N/A   41C    P0    62W / 163W |  11832MiB / 32480MiB |      3%      Default |
| N/A   41C    P0    65W / 163W |  12532MiB / 32480MiB |      3%      Default |
| N/A   41C    P0    61W / 163W |  13188MiB / 32480MiB |      2%      Default |
| N/A   41C    P0    57W / 163W |  15252MiB / 32480MiB |     93%      Default |
| N/A   42C    P0    66W / 163W |  15252MiB / 32480MiB |    100%      Default |
| N/A   42C    P0    66W / 163W |  15252MiB / 32480MiB |    100%      Default |
| N/A   43C    P0   136W / 163W |  13966MiB / 32480MiB |     97%      Default |
| N/A   45C    P0   159W / 163W |  13966MiB / 32480MiB |     99%      Default |
| N/A   43C    P0    67W / 163W |  13312MiB / 32480MiB |      4%      Default |
| N/A   42C    P0    66W / 163W |  11536MiB / 32480MiB |      4%      Default |
```

Issue3. 전처리

```
DEBUG :  epoch:0   iter:1   tat: 41.161 /sec
DEBUG :  epoch:0   iter:2   tat: 39.368 /sec
```

```
DEBUG :  epoch:15   iter:11   tat: 3.779 /sec
```

```
| N/A   50C    P0   128W / 163W |  21770MiB / 32480MiB |     99%      De
| N/A   51C    P0   160W / 163W |  21770MiB / 32480MiB |     99%      De
| N/A   50C    P0   119W / 163W |  21770MiB / 32480MiB |     37%      De
| N/A   51C    P0   120W / 163W |  21770MiB / 32480MiB |     45%      De
| N/A   51C    P0   160W / 163W |  21770MiB / 32480MiB |     99%      De
| N/A   50C    P0    77W / 163W |  21770MiB / 32480MiB |     97%      De
| N/A   50C    P0   120W / 163W |  21770MiB / 32480MiB |     41%      De
| N/A   50C    P0   135W / 163W |  21770MiB / 32480MiB |    100%      De
| N/A   51C    P0   154W / 163W |  21770MiB / 32480MiB |     99%      De
| N/A   50C    P0   113W / 163W |  21770MiB / 32480MiB |     40%      De
| N/A   49C    P0    63W / 163W |  21770MiB / 32480MiB |     90%      De
| N/A   50C    P0   137W / 163W |  21770MiB / 32480MiB |     99%      De
```

DL 학습 병렬화 전략

2. 모델 설계단계부터 multiGPU고려

기존 접근

Concept → 모델 → MultiGPU → MultiNode

Debug

개발 기간단축

제안

Concept → MultiGPU 고려 모델 → MultiGPU → MultiNode

Debug

Template for train script

1. **Data loader**
2. **Load model**
3. **Load checkpoint**
4. **Iteration**
   a. **Evaluate**
   b. **Save checkpoint**

Template for train script

1. **Data loader(Sampler)**
2. **Load model**
3. **Load checkpoint**
4. **Iteration**
   a. **Evaluate**
   b. **Save checkpoint**

DL 학습 병렬화 전략

2. 모델 설계 단계부터 multiGPU구현

데이터가 큰 경우
학습시간이 오래 걸리는 경우

제안

모델 구현 단계에
검증 작업이 빈번하게 발생시 시간절약
TAT 단축을 통한 빠른 결과 도출

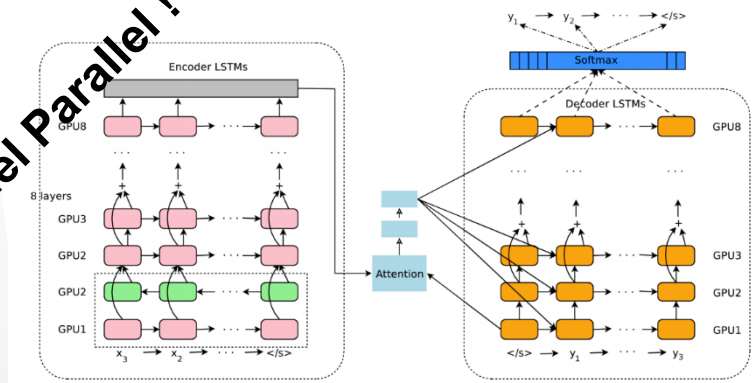
규모있는 문제 도전 가능

DL 학습 병렬화 전략

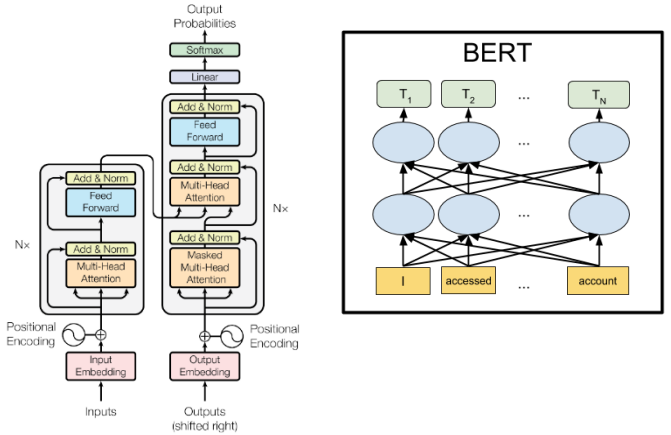
모델 설계단계부터 multiGPU가 고려된 모델들

Google's GNMT

Model Parallel !



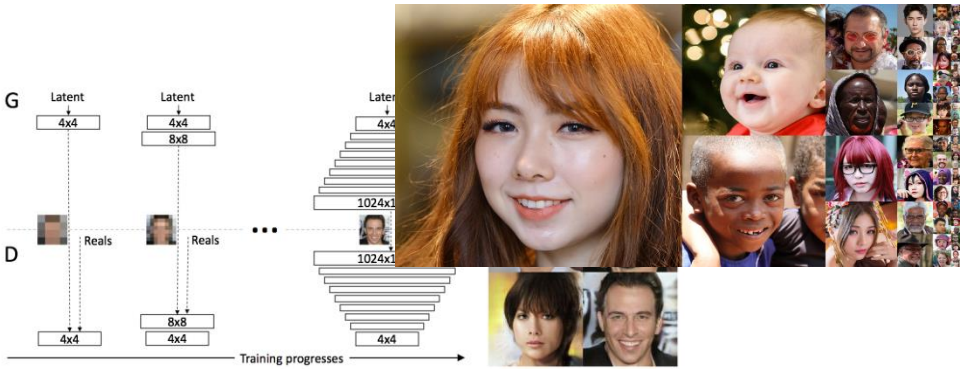
Transformer, BERT, GPT-2, XLNet



OpenAI's GLOW

40 GPU

NVIDIA's PGGAN, StyleGAN, WaveGlow



그럼 어떻게?

NGC

NVIDIA GPU Cloud

Focus Model

- **NGC Model Script** https://ngc.nvidia.com/catalog/model-scripts
- **NGC (Containers)** https://ngc.nvidia.com/
- **DeepOps (DL+ DevOps)** https://github.com/NVIDIA/deepops
- **DGX POD design** https://www.nvidia.com/ko-kr/data-center/dgx-pod-reference-architecture/
- **DGX Station DGX-1 DGX-2**

예 : Multi-node DL train on NGC

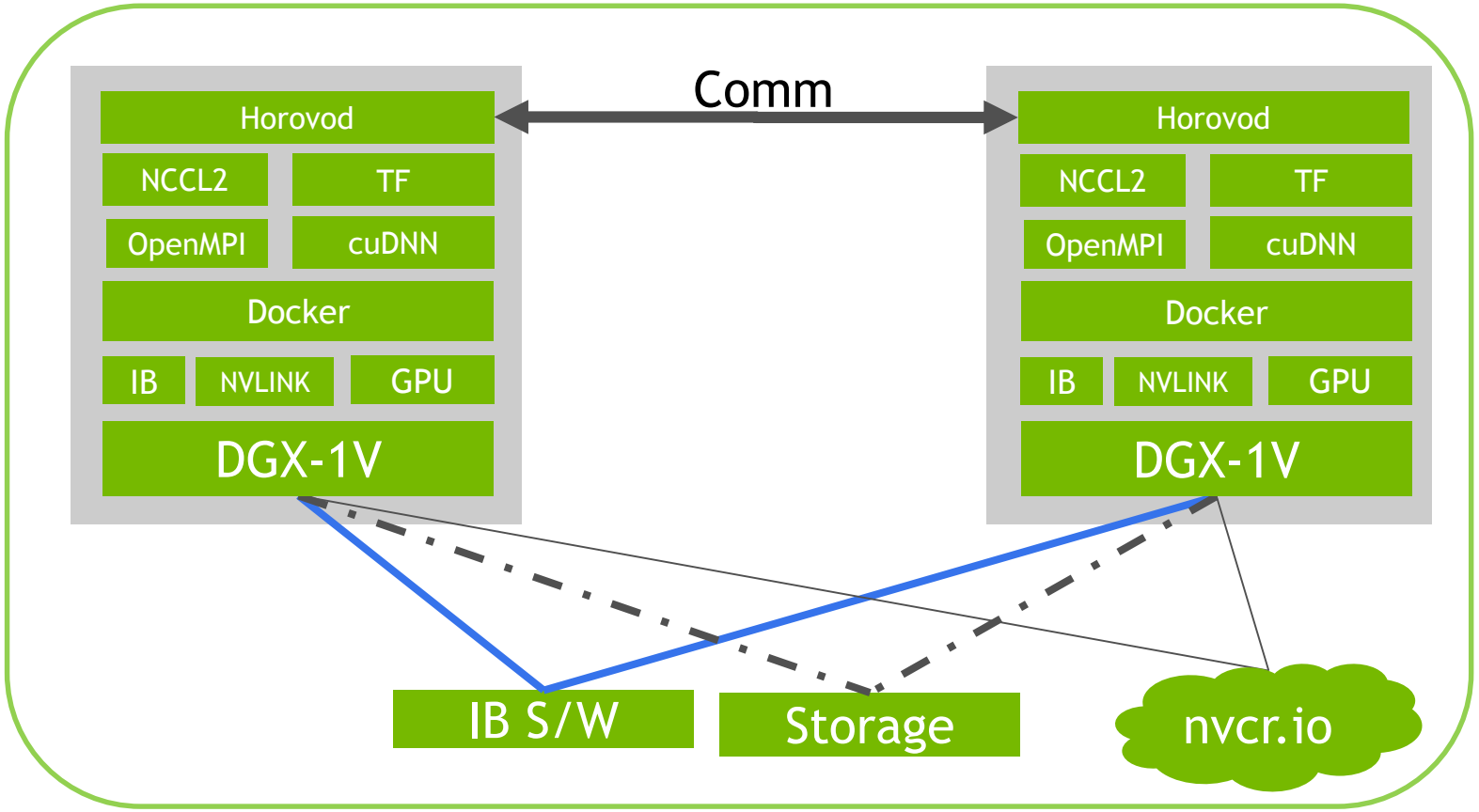
Horovod 이용시
복잡한 부분은 오히려 DevOps/Infrastructure 파트

2017.12

System :
- 3EA DGX-1
- IB EDR : ½ Blocking
- Storage : NFS
- OS : Ubuntu 16.04

DevOps :
- Slurm 16.05
- NVIDIA-Docker 1.0
- Singularity 2.4

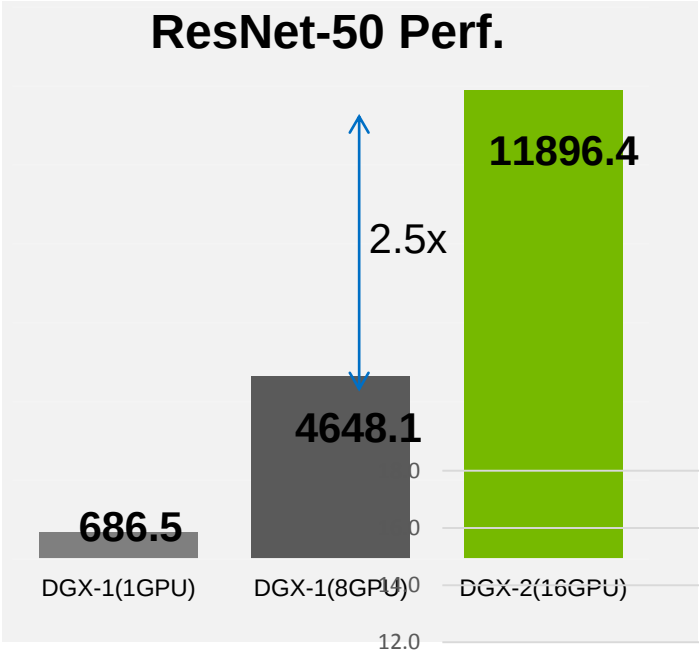
S/W :
- Horovod plug-in
- Tensorflow v1.5
- NCCL v2.1.2
- OpenMPI v3.0
- Base Container : nvidia/cuda:9.0-devel
  - CUDA v9.0
  - CUDNN v7.0

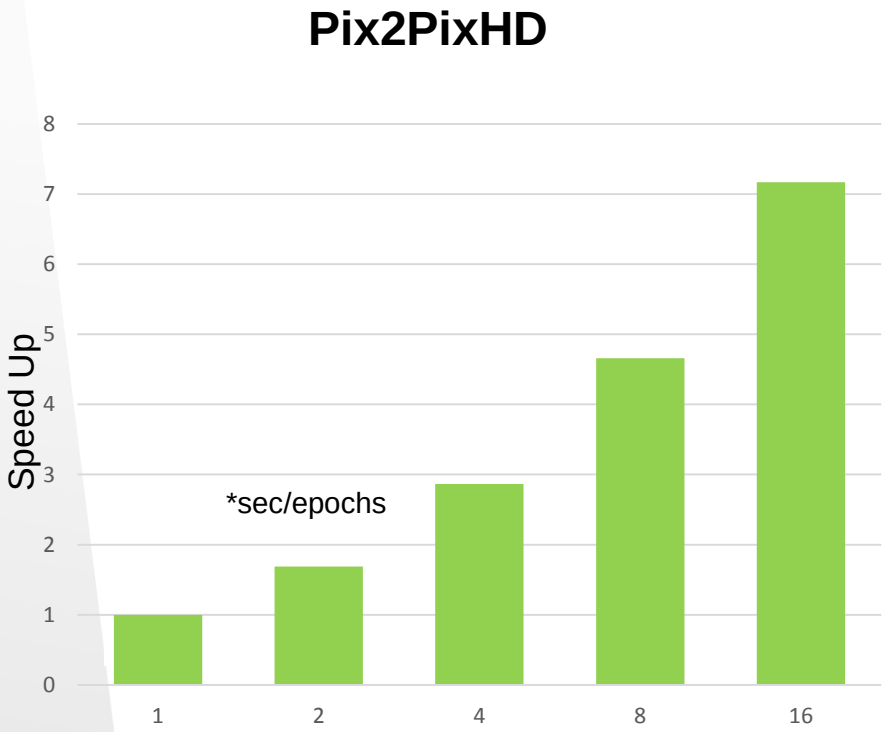


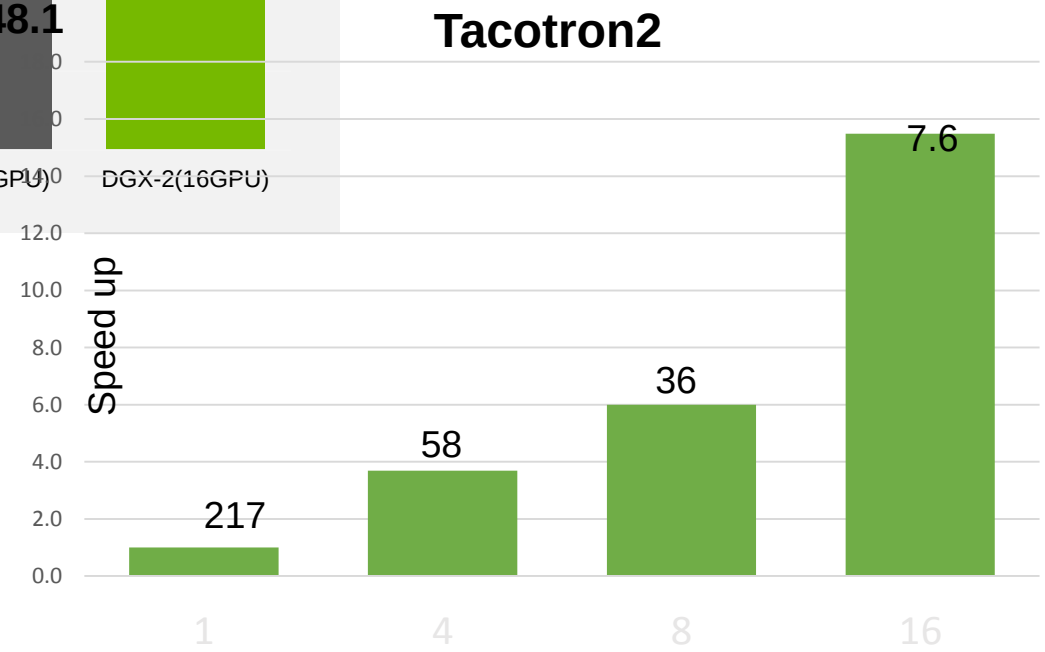
5분 만에 재현할 수 있도록 Dockerfile 제공
https://github.com/yhgon/horovod-tf-uber/blob/master/Dockerfile.c9d7tf15

DGX-1, DGX-2 MultiGPU Perf. on NGC

AI Conf. 2018







재현할 수 있도록 Dockerfile, Guide 제공

https://github.com/yhgon/KR_AICONF_2018

구체적인 접근1
Template

1. Template for New Project

```
jasper_pytorch
    requirement.txt
    Dockerfile
    README.md
    train.py
    model.py
    inference.py
    config.json
    modules
        my_layers.py
        SpecAug.py
    utils
        audio_processing.py(python)
        data_utils.py
    scripts
        dataset.sh(shell)
```

신규 프로젝트 작업시

Model, Data Loader, Train script 분리

Template 에 NV-Tacotron2 pytorch

filelists	changing structure for better organization	last year
text	text/: adding Keith Itos text pre-processing	last year
Dockerfile	Dockerfile: adding dockerfile	last year
LICENSE	Update license such that it appears on repo fron tpage	
README.md	README.md: describing how to load mel from disk	
audio_processing.py	adding python files	
data_utils.py	data_utils.py: adding support for loading mel from disk	
distributed.py	adding python files	
fp16_optimizer.py	adding python files	
hparams.py	hparams.py: adding load_mel_from_disk params	
inference.ipynb	ipynb typo	
layers.py	adding python files	
logger.py	adding python files	
loss_function.py	adding python files	
loss_scaler.py	loss_scaler.py: patching loss scaler for compatibility with current p...	last year
model.py	model.py: moving for better readibility	last year
multiproc.py	adding python files	last year
plotting_utils.py	adding python files	last year
requirements.txt	requirements.txt: updating tensorflow requirements	last year
stft.py	adding python files	last year
tensorboard.png	tensorboard.png: adding tensorboard image	last year
train.py	train.py: fixing typo	last year
utils.py	mask utils update for 0.4 cuda	last year

SOL

- **APEX의 예제 template사용(old)**
  multiproc.py
- **TF hparam 사용**
  hparams.py
- **Keith Ito의 Text 유틸 재활용**
  text/cleaner.py
- **Librosa 라이브러리 사용**
  stft.py

Template 에 NV-wavenet pytorch

..		
tacotron2 @ 064629c	Adding code for training your own models	last year
Makefile	updated PyTorch wrapper to use cpp_extension instead of ffi	8 months ago
README.md	updated PyTorch wrapper to use cpp_extension instead of ffi	
build.py	updated PyTorch wrapper to use cpp_extension instead of ffi	
config.json	Getting rid of magic numbers	
distributed.py	Getting rid of magic numbers	
inference.py	Added tests for A=512,1024 and fixed pytorch wrapper bug	
integration_test.py	Add pause/resume and CPU-asynchronous generation; integration test	
mel2samp_onehot.py	error in error string	
nv_wavenet.py	updated PyTorch wrapper to use cpp_extension instead of ffi	
nv_wavenet_test.py	Adding code for training your own models	last year
train.py	Making implementation a parameter	last year
utils.py	Getting rid of magic numbers	last year
wavenet.py	Getting rid of start end	last year
wavenet_infer.cu	Added tests for A=512,1024 and fixed pytorch wrapper bug	8 months ago
wavenet_infer.h	Adding PyTorch wrapper	last year
wavenet_infer_wrapper.cpp	updated PyTorch wrapper to use cpp_extension instead of ffi	8 months ago

SOL

- **APEX의 예제 template사용 distributed.py**
- **기존 utility 재활용 tacotron2/STFT.py**
- **타 데이터셋 검증 안함 16Khz CMU artic only**

Template 에 WaveGlow pytorch

tacotron2 @ fc0cf6a	waveglow	8 months ago
.gitmodules	waveglow	
LICENSE	LICENSE: adding license file	
README.md	Merge branch 'master' into master	
config.json	config.json: adding optional tensorboard	
convert_model.py	convert_model.py: script for converting pre fused res_skip to fuse	
denoiser.py	denoiser.py: adding code to remove model bias	
distributed.py	waveglow	
glow.py	glow.py: inverse only exists on fp32+ data	
glow_old.py	Resolve PyTorch deprecation warnings	7 months ago
inference.py	inference.py: using amp for mixed precision inference	3 months ago
mel2samp.py	mel2samp.py: adding fmin and fmax params	8 months ago
requirements.txt	torch in requirements	5 months ago
train.py	train.py: offseting tensorboard iterator count	29 days ago
waveglow_logo.png	waveglow	8 months ago

SOL

- **NV-Wavenet pytorch와 구조 동일**
- **APEX의 예제 template사용(최신)**
  distributed.py
- **기존 utility 재활용**
  tacotron2/STFT.py
- **데이터툴 재활용**
  load checkpoint from nv-wavenet

Good Start Point !!!

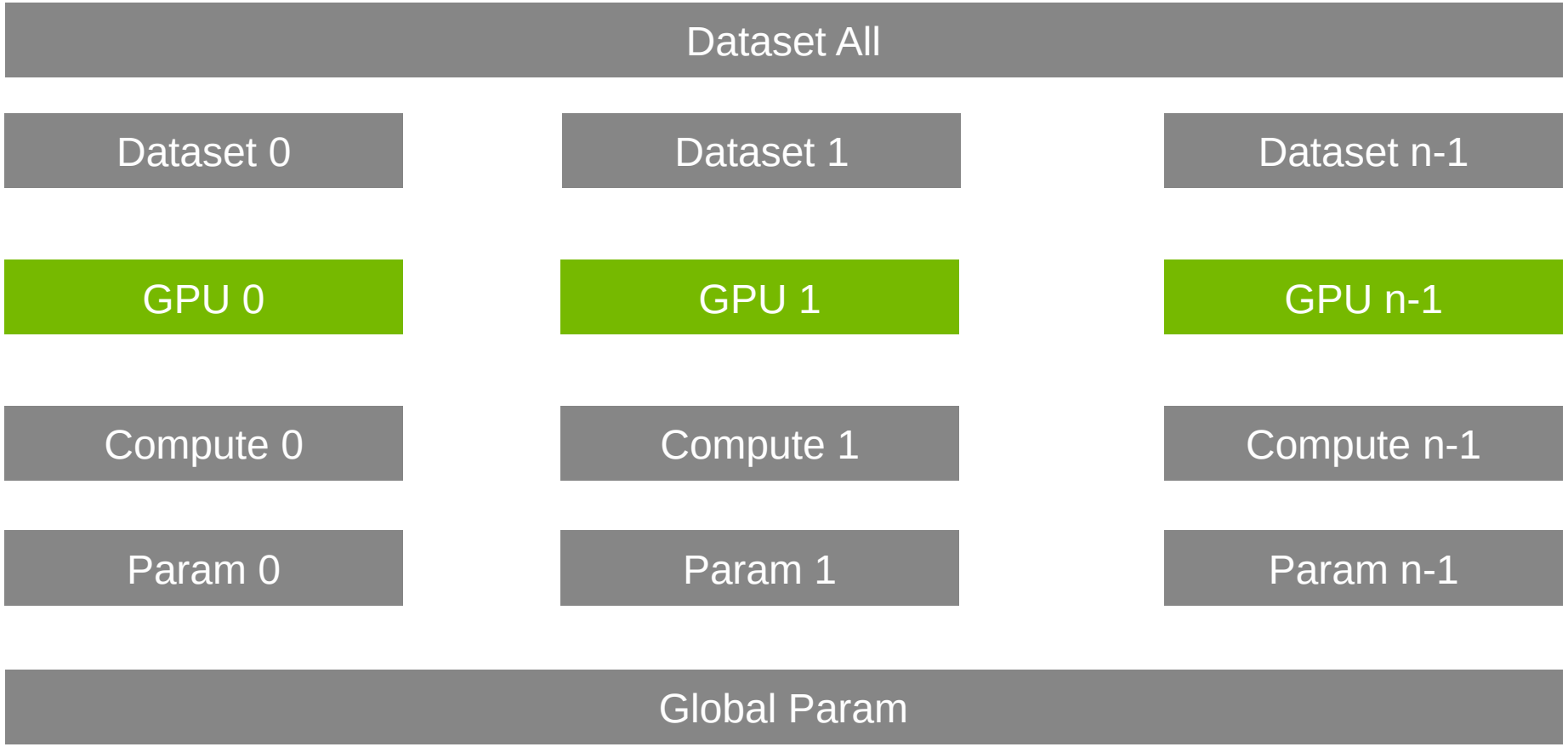
구체적인 접근2
train.py 수정

DL Data Parallel

데이터 분할

대부분
독립 계산

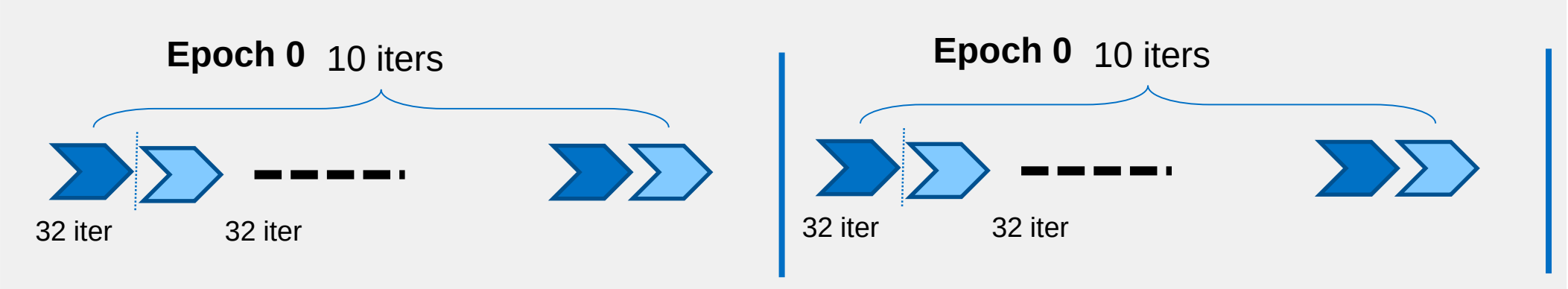
통신 전략
reduce
broadcast



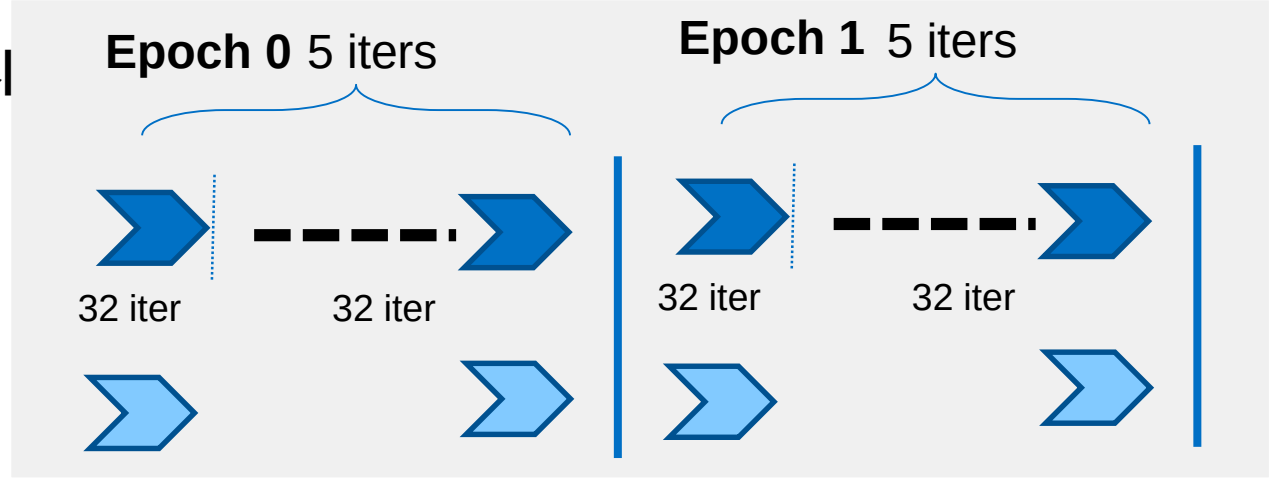
MultiGPU 병렬화

Dataset : 320
batch : 32
GPU : 2

GPU 1개 사용시



GPU 2개 사용시



접근 방법

PYTORCH DataParallel (DP)

Data Parallelism is when we split the mini-batch of samples into multiple smaller mini-batches and run the computation for each of the smaller mini-batches in parallel.

PYTORCH Distributed Data Parallel (DDP)

DistributedDataParallel (DDP) implements data parallelism at the module level. It uses communication collectives in the torch.distributed package to synchronize gradients, parameters, and buffers. Parallelism is available both within a process and across processes. Within a process, DDP replicates the input module to devices specified in device_ids, scatters inputs along the batch dimension accordingly, and gathers outputs to the output_device, which is similar to DataParallel.

Horovod-pytorch

Horovod core principles are based on the MPI concepts *size*, *rank*, *local rank*, *allreduce*, *allgather*, and *broadcast*.

APEX (NVIDIA custom DDP)

apex.parallel.DistributedDataParallel is a module wrapper, similar to DDP(torch.nn.parallel.DistributedDataParallel). It enables convenient multiprocess distributed training, optimized for NVIDIA's NCCL communication library.

torch.nn.DataParallel

쉬운 접근법
많은 community code (pytorch 구현)에서 DataParallel 조차 사용하지 않음

We couldn't find any code matching 'DataParallel' in mrzhu-cool/pix2pix-pytorch

We couldn't find any code matching 'DataParallel' in r9y9/tacotron_pytorch

You could se

We couldn't find any code matching 'DataParallel' in r9y9/wavenet_vocoder

You could search all of GitHub or try an advanced search.

Torch.nn.DataParallel

PYTORCH DataParallel (DP)
Data Parallelism is when we split the mini-batch of samples into multiple smaller mini-batches and run the computation for each of the smaller mini-batches in parallel.

Easiest Way
Model만 DataParallel로 감싸주면 pytorch가 알아서 작업

```
import torch
from torch import nn
from torch.utils.data import DataLoader

if args.cuda:
    model.cuda()
if args.parallel:
    model = nn.DataParallel(model)
```

nn.DataParallel 구현 코드들

1 code result in codertimo/BERT-pytorch or view all results on GitHub

bert_pytorch/trainer/pretrain.py

Showing the top match Last indexed on Oct 23, 2018

```
51              self.model = nn.DataParallel(self.model,
52
53          # Setting the train and test data loader
```

7 code results in huggingface/pytorch-pretrained-BERT or view all results on GitHub

examples/extract_features.py Python

Showing the top match Last indexed on Apr 2

```
247     elif n_gpu > 1:
248         model = torch.nn.DataParallel(model)
249
250     all_input_ids = torch.tensor([f.input_ids for f in features], dtype=torch.long)
```

4 code results in facebookresearch/SIN

sing/sequence/utils.py Python

Showing the top match Last indexed on Oct 20, 2018

```
38      embeddings_dataset = [None] * len(dataset)
39
40      if cuda:
41          encoder.cuda()
42      if parallel:
43          encoder = nn.DataParallel(encoder)
```

Torch.nn.DataParallel

단점 :

0번 GPU 메모리 사용량 증가 및 0번 GPU 부하발생
(타 GPU 메모리를 비효율적으로 사용)

통신량이 늘어날 수록 성능저하 발생 4GPU 이상

@yhgon I am planning on using DDP for my future work but at the time DP was the most obvious choice at the time.
I have not tried to make the training scale to such a large level of GPUs. In my my experiments 4 GPUs was giving the ideal speedup. Using DDP might help help to scale further.
I now realize that there are some design mistakes in the way I used DataParallel. In particular, the generation of the waveform is parallelized and then gathered on GPU 0 and the loss is computed only on GPU 0. This explain explain the increased memory usage for GPU0 as well as the some suboptimal suboptimal usage of the GPUs. Changing that would require a bit of work but I but I don't currently have a lot of time to do so.

torch.nn.parallel.DistributedDataParallel

PYTORCH Distributed Data Parallel (DDP)

DistributedDataParallel (DDP) implements data parallelism at the module level. It uses communication collectives in the torch.distributed package to synchronize gradients, parameters, and buffers. Parallelism is available both within a process and across processes. Within a process, DDP replicates the input module to devices specified in device_ids, scatters inputs along the batch dimension accordingly, and gathers outputs to the output_device, which is similar to DataParallel.

torch.nn.parallel.DistributedDataParallel

동일 프로세스이면
DP와 DDP가 동일하게 작동

Pytorch imagenet example

```
import torch.distributed as dist
from  torch.nn.parallel import DistributedDataParallel as DDP
from  torch.nn import DataParallel as DP

if args.distributed:
        if args.gpu is not None:
            torch.cuda.set_device(args.gpu)
            model.cuda(args.gpu)
            args.batch_size = int(args.batch_size / ngpus_per_node)
            args.workers = int(args.workers / ngpus_per_node)
            model = DDP(model, device_ids=[args.gpu])
        else:
            model.cuda()
            model = DDP(model)
 else:
            model.cuda()
            model = DP(model)
```

horovod

Horovod-pytorch

Horovod core principles are based on the MPI concepts *size*, *rank*, *local rank*, *allreduce*, *allgather*, and *broadcast*.

https://eng.uber.com/horovod/

https://github.com/horovod/horovod

Horovod 사용법

To use Horovod

1. Run hvd.init().

2. Pin a server GPU to be used by this process using config.gpu_options.visible_device_list.

3. Scale the learning rate by number of workers.

4. Wrap optimizer in hvd.DistributedOptimizer.

5. Add hvd.BroadcastGlobalVariablesHook(0) to broadcast initial variable states from rank 0 to all other processes.

6. Modify your code to save checkpoints only on worker 0 to prevent other workers from corrupting them.

```python
import tensorflow as tf
import horovod.tensorflow as hvd


# Initialize Horovod
hvd.init()

# Pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())

# Build model...
loss = ...
opt = tf.train.AdagradOptimizer(0.01 * hvd.size())

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)

# Add hook to broadcast variables from rank 0 to all other processes during
# initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# Make training operation
train_op = opt.minimize(loss)

# Save checkpoints only on worker 0 to prevent other workers from corrupting them.
checkpoint_dir = '/tmp/train_logs' if hvd.rank() == 0 else None

# The MonitoredTrainingSession takes care of session initialization,
# restoring from a checkpoint, saving to a checkpoint, and closing when done
# or an error occurs.
with tf.train.MonitoredTrainingSession(checkpoint_dir=checkpoint_dir,
                                       config=config,
                                       hooks=hooks) as mon_sess:
  while not mon_sess.should_stop():
    # Perform synchronous training.
    mon_sess.run(train_op)
```

Horovod multiGPU 병렬화 전략

Horovod on Tensorflow Keras 활용시 (약 7 줄 수정)

Keras와 Pytorch Horovod 사용법 동일

```
import horovod.keras as hvd

# Horovod: initialize Horovod.
hvd.init()

# Horovod: pin GPU to be used to process local rank (one GPU per process)
config.gpu_options.visible_device_list = str(hvd.local_rank())

# Horovod: add Horovod Distributed Optimizer.
opt = hvd.DistributedOptimizer(opt)

callbacks = [
   hvd.callbacks.BroadcastGlobalVariablesCallback(0),
   hvd.callbacks.MetricAverageCallback(),
   hvd.callbacks.LearningRateWarmupCallback(warmup_epochs=5, verbose=1),
   keras.callbacks.ReduceLROnPlateau(patience=10, verbose=1), ]

verbose = 1 if hvd.rank() == 0 else 0
```

https://github.com/horovod/horovod/blob/master/examples/keras_mnist_advanced.py

Horovod pytorch

데이터 분할

통신

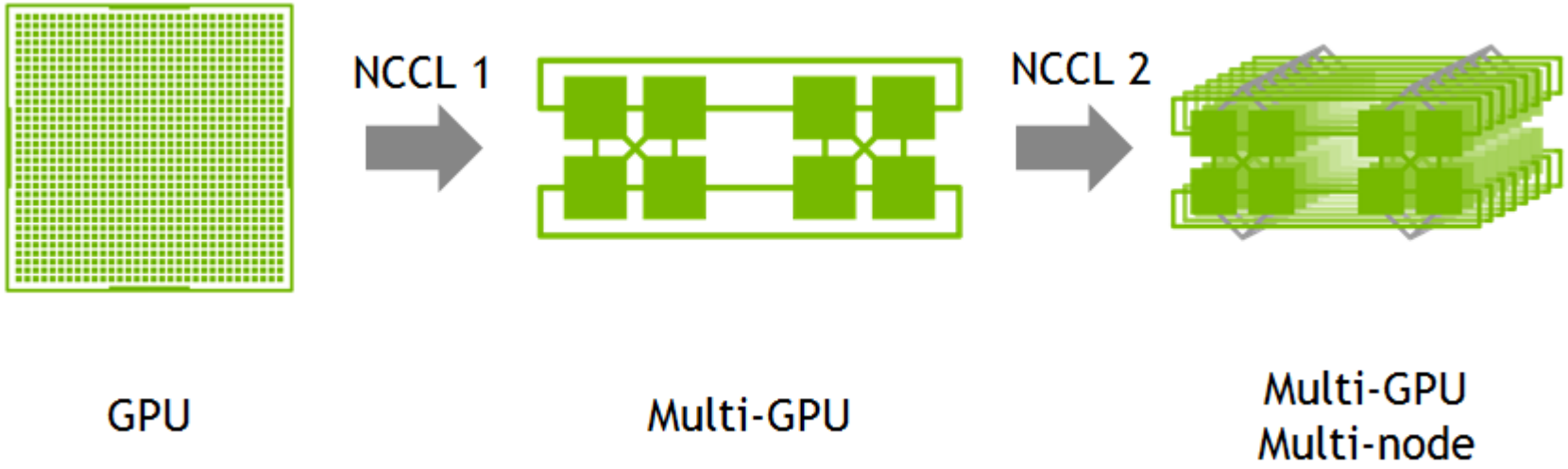
독립 계산

```
import torch.nn as nn
import torch.utils.data.distributed as DDP
import horovod.torch as hvd

hvd.init()
torch.cuda.set_device(hvd.local_rank())
train_sampler = DDP.DistributedSampler( train_dataset, num_replicas=hvd.size(), rank=hvd.rank())
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=args.batch_size, sampler=train_sampler,
**kwargs)

model = Net()

optimizer = optim.SGD(model.parameters(), lr=args.lr * hvd.size(),  momentum=args.momentum)
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
hvd.broadcast_optimizer_state(optimizer, root_rank=0)
compression = hvd.Compression.fp16 if args.fp16_allreduce else hvd.Compression.none
optimizer = hvd.DistributedOptimizer(optimizer, named_parameters=model.named_parameters(),
compression=compression)

 output = model(data)
 loss = F.nll_loss(output, target)
 loss.backward()
 optimizer.step()
```

APEX 활용

APEX (NVIDIA custom DDP)

apex.parallel.DistributedDataParallel is a module wrapper, similar to DDP(torch.nn.parallel.DistributedDataParallel). It enables convenient multiprocess distributed training, optimized for NVIDIA's NCCL communication library.



NVIDIA APEX DDP

```
import torch.distributed as dist
from torch.nn.modules import Module
from torch.autograd import Variable

class DistributedDataParallel(Module):
 def __init__(self, module, ...):
        super(DistributedDataParallel, self).__init__()

        if hasattr(dist, "get_backend"):
            self._backend = dist.get_backend()
            if hasattr(dist, "DistBackend"):
                self.backend_enum_holder = dist.DistBacken
            else:
                self.backend_enum_holder = dist.Backend
        else:
            self._backend = dist._backend
            self.backend_enum_holder = dist.dist_backend

        self.warn_on_half = True if self._backend ==
self.backend_enum_holder.GLOO else False

        self.world_size = float(dist.get_world_size())

        self.retain_allreduce_buffers = retain_allreduce_bu
        self.allreduce_always_fp32 = allreduce_always_fp3
        self.gradient_average = gradient_average
        self.gradient_predivide_factor = gradient_predivide

        self.custom_allreduce_triggers = False
```

```
self.delay_allreduce = delay_allreduce
        self.message_size = message_size

        self.reduction_stream = torch.cuda.Stream()
        self.reduction_event = torch.cuda.Event(enable_timing=False, blocking=False)

        self.module = module

        self._disable_allreduce = False

        if self._backend == self.backend_enum_holder.NCCL:
            for param in self.module.parameters():
                assert param.is_cuda, "NCCL  on GPU."

        self.active_params = []

        self.param_type_to_tmp_i = {"torch.cuda.HalfTensor" : 0,
                                    "torch.cuda.FloatTensor" : 1,
                                    "torch.cuda.DoubleTensor" : 2}

        if multi_tensor_applier.available:
            # TODO:  I really need to centralize the C++ backed imports
            import amp_C
            self.multi_tensor_scale = amp_C.multi_tensor_scale
            self._overflow_buf = torch.cuda.IntTensor([0])

        self.create_hooks()
        flat_dist_call([param.data for param in self.module.parameters()],
dist.broadcast, (0,))
```

https://github.com/NVIDIA/apex/blob/master/apex/parallel/distributed.py

APEX 사용법

1. APEX source build 사용

```
$ git clone https://github.com/NVIDIA/apex
$ cd apex
$ pip install -v --no-cache-dir --global-option="--cpp_ext" --global-option="--cuda_ext" ./
```

2. NGC 사용 (apex included)

- Ubuntu 16.04 including Python 3.6 environment
- NVIDIA CUDA 10.1.168 including cuBLAS 10.2.0.168
- NVIDIA cuDNN 7.6.0
- NVIDIA NCCL 2.4.7 (optimized for NVLink™)
- APEX
- OpenMPI 3.1.3
- TensorRT 5.1.5
- DALI 0.10.0 Beta

3. Custom script

APEX의 distributed.py 혹은 multiproc.py만 가져다 사용

Template NV-Tacotron2 pytorch

filelists	changing structure for better organization	last year
text	text/: adding Keith Itos text pre-processing	last year
Dockerfile	Dockerfile: adding dockerfile	last year
LICENSE	Update license such that it appears on repo fron tpage	
README.md	README.md: describing how to load mel from disk	
audio_processing.py	adding python files	
data_utils.py	data_utils.py: adding support for loading mel from disk	
distributed.py	adding python files	
fp16_optimizer.py	adding python files	
hparams.py	hparams.py: adding load_mel_from_disk params	
inference.ipynb	ipynb typo	
layers.py	adding python files	
logger.py	adding python files	
loss_function.py	adding python files	
loss_scaler.py	loss_scaler.py: patching loss scaler for compatibility with current p...	last year
model.py	model.py: moving for better readibility	
multiproc.py	adding python files	
plotting_utils.py	adding python files	
requirements.txt	requirements.txt: updating tensorflow requirements	
stft.py	adding python files	
tensorboard.png	tensorboard.png: adding tensorboard image	
train.py	train.py: fixing typo	
utils.py	mask utils update for 0.4 cuda	

SOL

- **APEX의 예제 template사용(old)**
  multiproc.py
- **TF hparam 사용**
  hparams.py
- **Keith Ito의 Text 유틸 재활용**
  text/cleaner.py
- **Librosa 라이브러리 사용**
  stft.py

Training

1. `python train.py --output_directory=outdir --log_directory=logdir`

Multi-GPU (distributed) and Automatic Mixed Precision Training

1. `python -m multiproc train.py --output_directory=outdir --log_directory=logdir --hparams=distributed_run=True,fp16_run=True`

NVIDIA Tacotron2

```python
import os
import time
import argparse
import math
from numpy import finfo

import torch
from distributed import apply_gradient_allreduce
import torch.distributed as dist
from torch.utils.data.distributed import DistributedSampler
from torch.utils.data import DataLoader

from model import Tacotron2
from data_utils import TextMelLoader, TextMelCollate
from loss_function import Tacotron2Loss
from logger import Tacotron2Logger
from hparams import create_hparams

```

```python
init_distributed(hparams, n_gpus, rank, group_name)

model = load_model(hparams)
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate,
                             weight_decay=hparams.weight_decay)
if hparams.distributed_run:
    model = apply_gradient_allreduce(model)

criterion = Tacotron2Loss()
x, y = model.parse_batch(batch)
y_pred = model(x)

if hparams.distributed_run:
    reduced_loss = reduce_tensor(loss.data, n_gpus).item()
loss.backward()
```

Template WaveGlow pytorch

tacotron2 @ fc0cf6a	waveglow	8 months ago
.gitmodules	waveglow	
LICENSE	LICENSE: adding license file	
README.md	Merge branch 'master' into master	
config.json	config.json: adding optional tensorboard	
convert_model.py	convert_model.py: script for converting pre fused res_skip to fuse	
denoiser.py	denoiser.py: adding code to remove model bias	
distributed.py	waveglow	
glow.py	glow.py: inverse only exists on fp32+ data	
glow_old.py	Resolve PyTorch deprecation warnings	7 months ago
inference.py	inference.py: using amp for mixed precision inference	3 months ago
mel2samp.py	mel2samp.py: adding fmin and fmax params	8 months ago
requirements.txt	torch in requirements	5 months ago
train.py	train.py: offseting ten	29 days ago
waveglow_logo.png	waveglow	

SOL
- NV-Wavenet pytorch와 구조 동일
- APEX의 예제 template사용(최신)
  distributed.py
- 기존 utility 재활용
  tacotron2/STFT.py
- 데이터툴 재활용
  load checkpoint from nv-wavenet

Train your own model

```
python train.py -c config.json
```

For multi-GPU training replace `train.py` with `distributed.py`. Only tested with single node and NCCL.

Good Start Point !!!

NVIDIA WaveGlow

```
import torch



from torch.utils.data import DataLoader



criterion = WaveGlowLoss(sigma)
model = WaveGlow(**waveglow_config).cuda()

optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)



trainset = Mel2Samp(**data_config)
train_sampler =  None

train_loader = DataLoader(trainset, num_workers=1, shuffle=False,
                          sampler=train_sampler,
                          batch_size=batch_size,
                          pin_memory=False,
                          drop_last=True)

model.train()
outputs = model((mel, audio))
loss = criterion(outputs)
reduced_loss = loss.item()
```

```
import torch

from distributed import init_distributed, apply_gradient_allreduce, reduce_tensor
from torch.utils.data.distributed import DistributedSampler
from torch.utils.data import DataLoader

init_distributed(rank, num_gpus, group_name, **dist_config)

criterion = WaveGlowLoss(sigma)
model = WaveGlow(**waveglow_config).cuda()

optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

model = apply_gradient_allreduce(model)

trainset = Mel2Samp(**data_config)
train_sampler = DistributedSampler(trainset) if num_gpus > 1 else None

train_loader = DataLoader(trainset, num_workers=1, shuffle=False,
                          sampler=train_sampler,
                          batch_size=batch_size,
                          pin_memory=False,
                          drop_last=True)

model.train()
outputs = model((mel, audio))
loss = criterion(outputs)
reduced_loss = reduce_tensor(loss.data, num_gpus).item()
```

multiGPU

multiGPU

multiGPU

```python
def train(num_gpus, rank, group_name, output_directory, epochs, learning_rate,
          sigma, iters_per_checkpoint, batch_size, seed, fp16_run,
          checkpoint_path, with_tensorboard):
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    #=====START: ADDED FOR DISTRIBUTED======
    if num_gpus > 1:
        init_distributed(rank, num_gpus, group_name, **dist_config)
    #=====END:   ADDED FOR DISTRIBUTED======

    criterion = WaveGlowLoss(sigma)
    model = WaveGlow(**waveglow_config).cuda()

    #=====START: ADDED FOR DISTRIBUTED======
    if num_gpus > 1:
        model = apply_gradient_allreduce(model)
    #=====END:   ADDED FOR DISTRIBUTED======

    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

    if fp16_run:
        from apex import amp
        model, optimizer = amp.initialize(model, optimizer, opt_level='O1')

    # Load checkpoint if one exists
    iteration = 0
    if checkpoint_path != "":
        model, optimizer, iteration = load_checkpoint(checkpoint_path, model,
                                                      optimizer)
        iteration += 1  # next iteration is iteration + 1

    trainset = Mel2Samp(**data_config)
    # =====START: ADDED FOR DISTRIBUTED======
    train_sampler = DistributedSampler(trainset) if num_gpus > 1 else None
    # =====END:   ADDED FOR DISTRIBUTED======
    train_loader = DataLoader(trainset, num_workers=1, shuffle=Fa
                              sampler=train_sampler,
                              batch_size=batch_size,
```

init_distributed(rank, num_gpus, group_name, **dist_config)

model = apply_gradient_allreduce(model)

train_sampler = DistributedSampler(trainset) if num_gpus > 1 else None

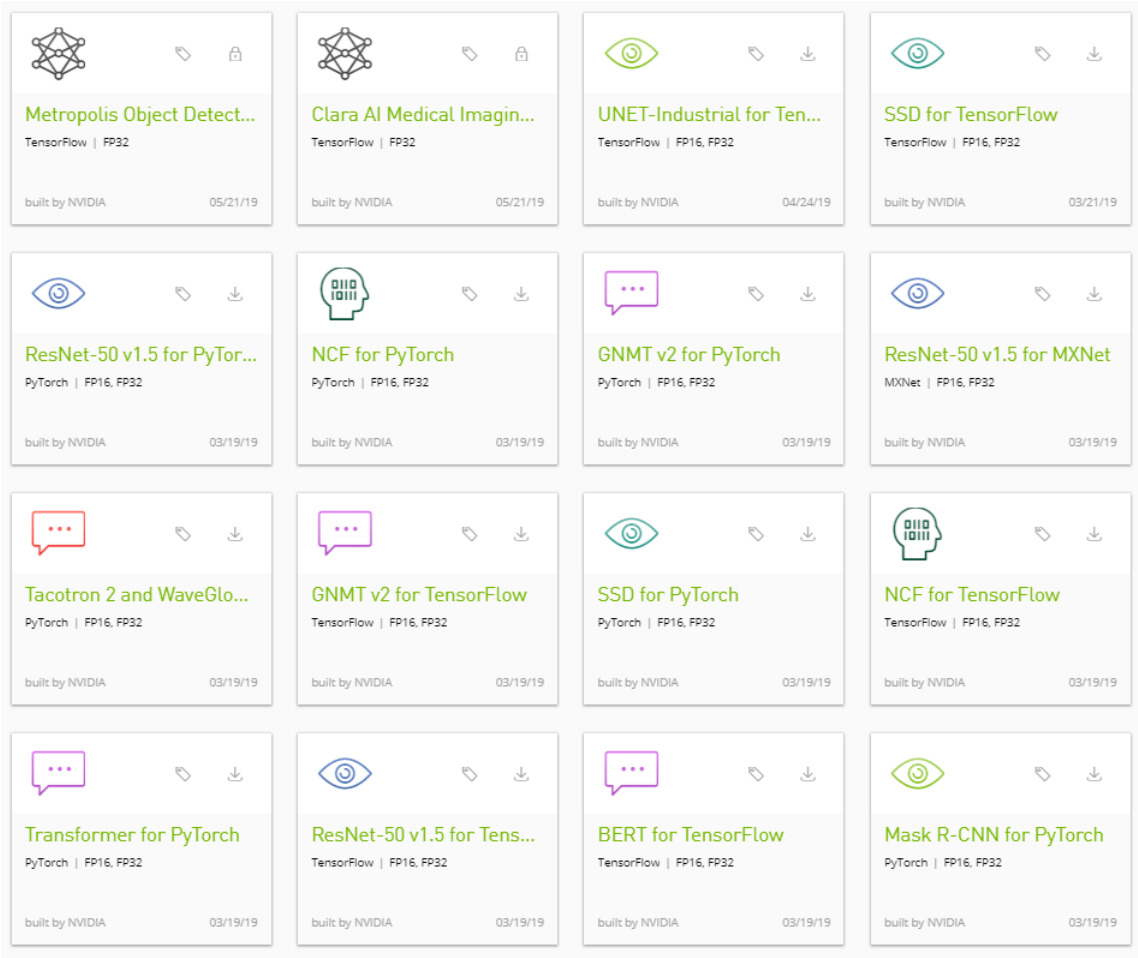
NVIDIA Metatron-LM

Use custom DDP from APEX

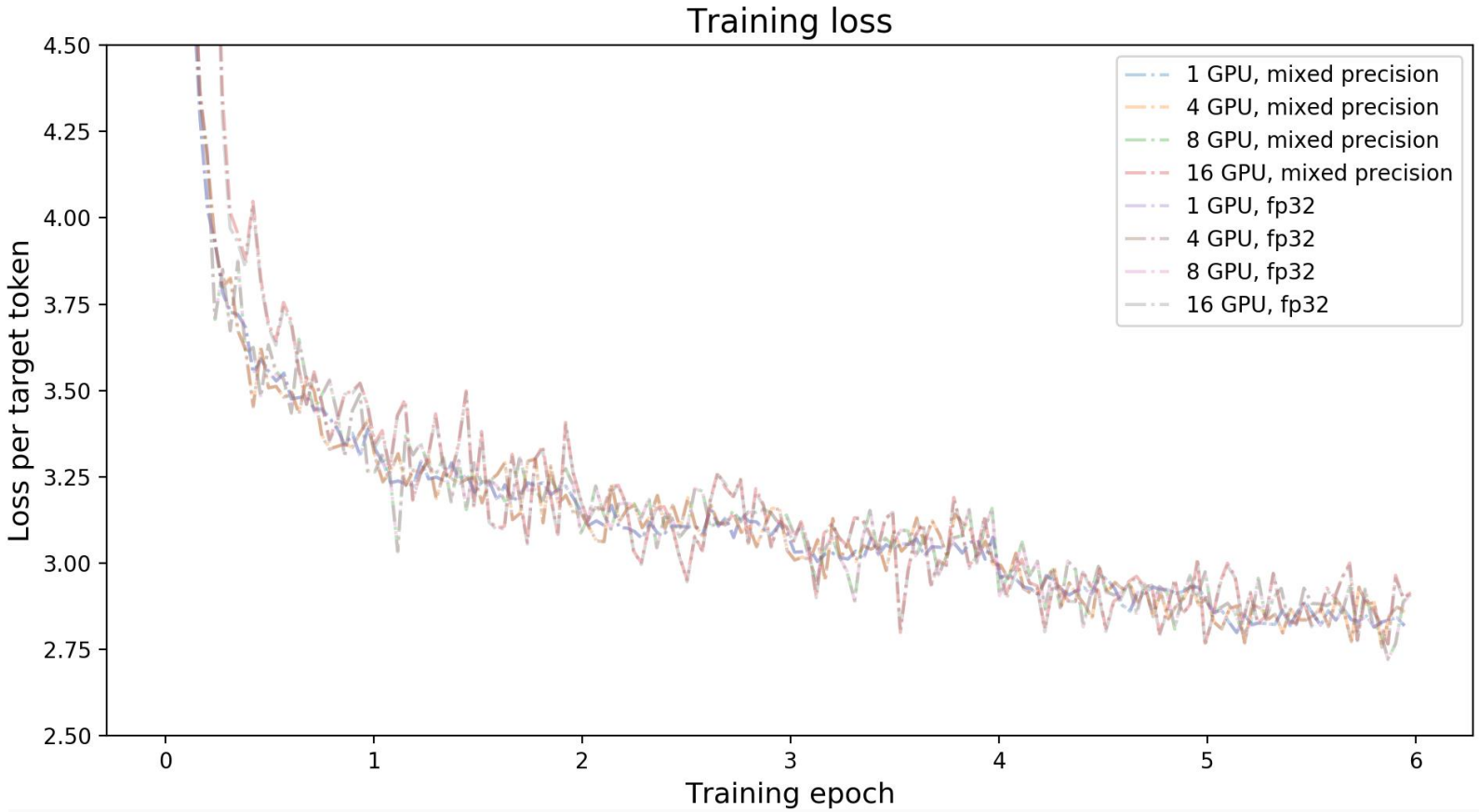
```
from model import DistributedDataParallel as DDP

model = BertModel(tokenizer, args)
model.cuda(torch.cuda.current_device())
model = DDP(model)
optimizer = Adam(param_groups, lr=args.lr, weight_decay=args.weight_decay)
optimizer = get_optimizer(model, args)
criterion = torch.nn.CrossEntropyLoss(reduce=False, ignore_index=-1)
output, nsp = model(tokens, types, 1-padding_mask, )
optimizer.zero_grad()
loss.backward()
```

Reference 코드들 NGC MODEL SCRIPTS

https://ngc.nvidia.com/catalog/model-scripts

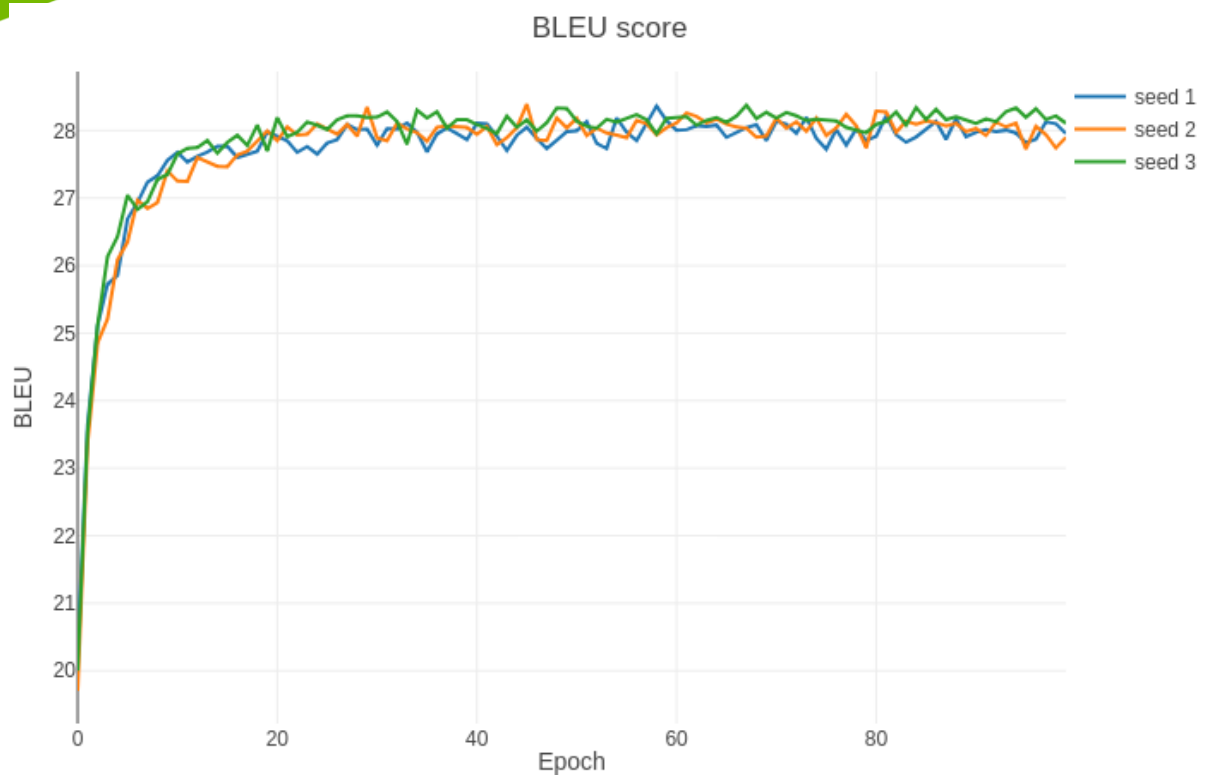


GNMT-pytorch



number of GPUs	batch size/GPU	mixed precision BLEU	fp32 BLEU	mixed precision training time	fp32 training time
1	128	24.59	24.71	265.0 minutes	825.1 minutes
4	128	24.69	24.33	87.4 minutes	216.3 minutes
8	128	24.50	24.47	49.6 minutes	113.5 minutes
16	128	24.22	24.16	26.3 minutes	58.6 minutes

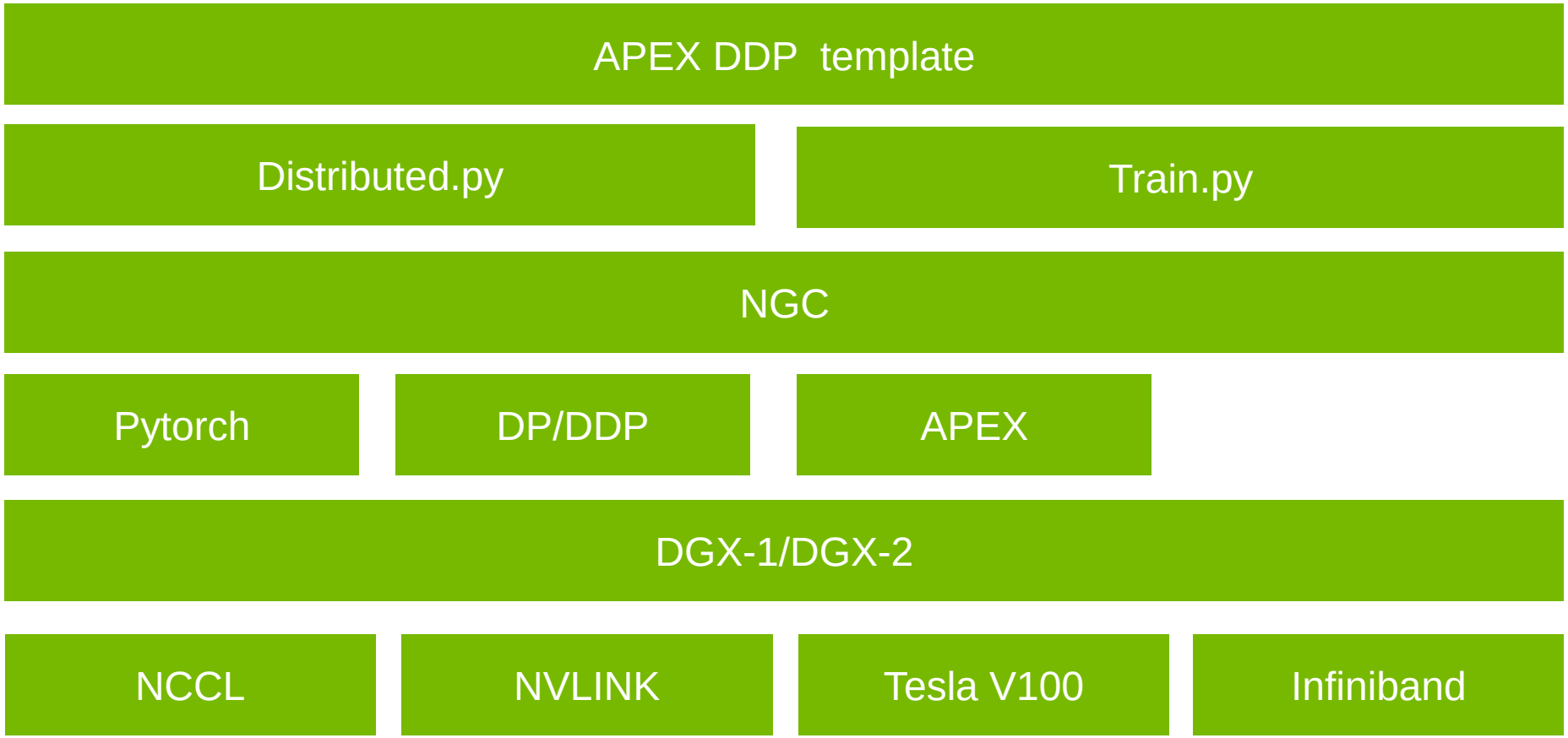
Transformer Pytorch



GPU count	Mixed precision	FP32	FP32/Mixed speedup	Mixed precision week scaling	FP32 week scaling
1	37650	8630	4.36	1.0	1.0
4	132700	30500	4.35	3.52	3.53
8	260000	61000	4.26	6.91	7.07

정리

Ref Model : WaveGlow, Tacotron2, NV-Wavenet-pyt, Megatron-LM, GNMT-pyt, Transformer-pyt



NVIDIA AI CONFERENCE

감사합니다.