시간 단축을 위한
분산 학습의 원리와 코딩 최적화

2020. 06. 05.

Data Mining & Quality Analytics Lab
신 욱 수

고려대학교
KOREA UNIVERSITY

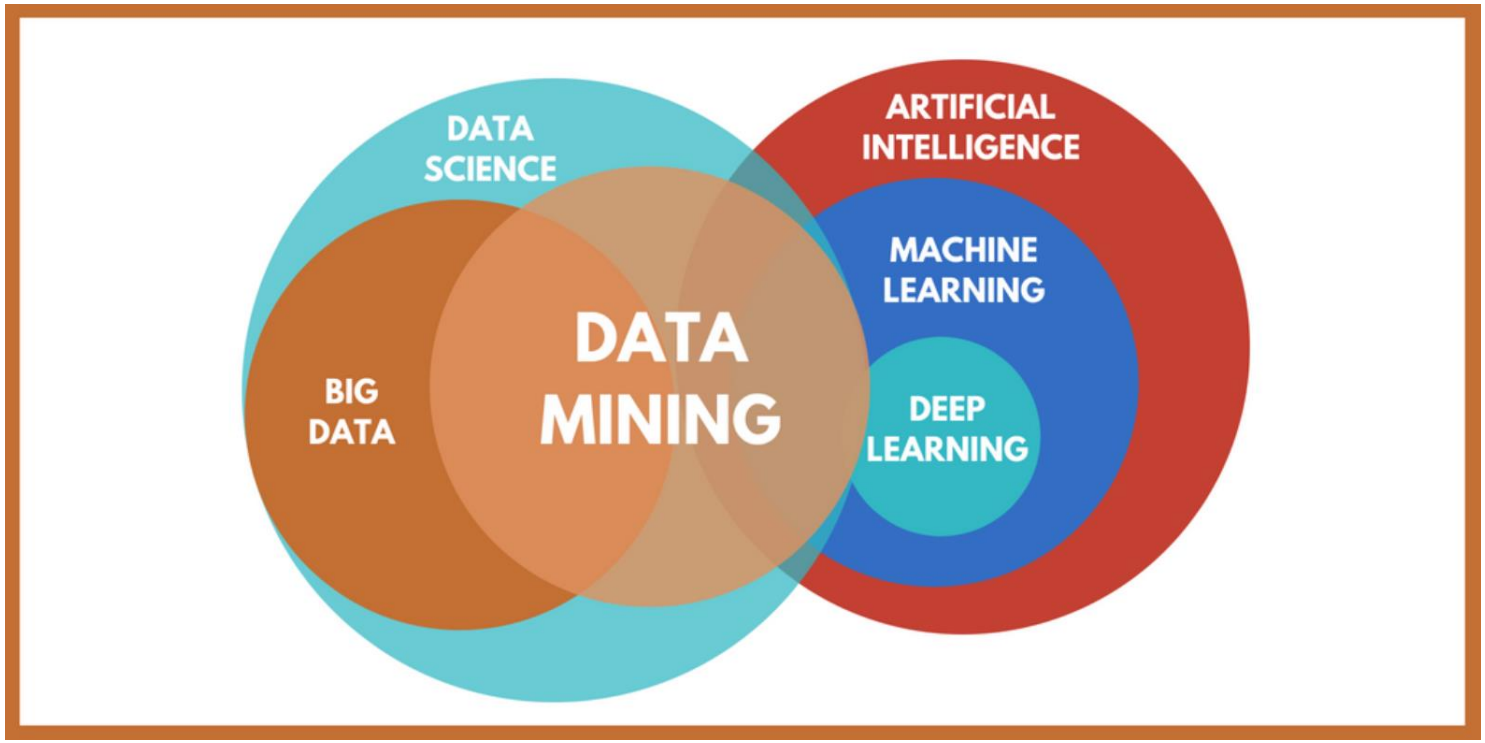
Outline

1. **Introduction**
   - **우리가 수행하는 연구의 특성 : GPU를 활용**
   - **어떤 문제에 부딪히는가? (memory, 학습속도)**
2. **Distributed Training**
   - 작업 분할 방식 : Data/Model Parallelism
   - 동기화 방식 : Sync/Async Parameters
   - 데이터 취합 방식 : All-Reduce , Ring-All-Reduce
   - H/W Architecture
3. **속도 개선을 위해 무엇을 보는가? (균형, 활용률)**
4. **Summary**

Introduction

❖ 우리가 하는 연구는?

- DMQA : Data Mining & Quality Analytics

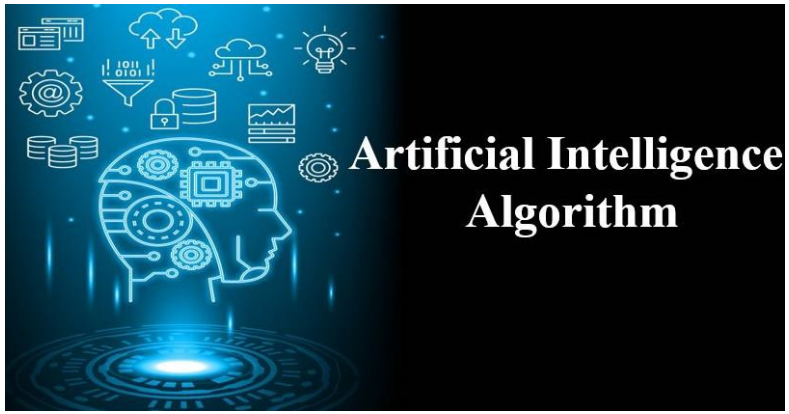


Source : https://www.dataevo.com.ar/post/diagrama-de-venn

Introduction

❖ 우리가 연구에 필요한 재료는 무엇일까요?

- Data & Computer & Algorithm (ML, DL)



Introduction

❖ 알고리즘은 점점 복잡해져 갑니다.

- 처리할 파라미터가 늘어납니다.

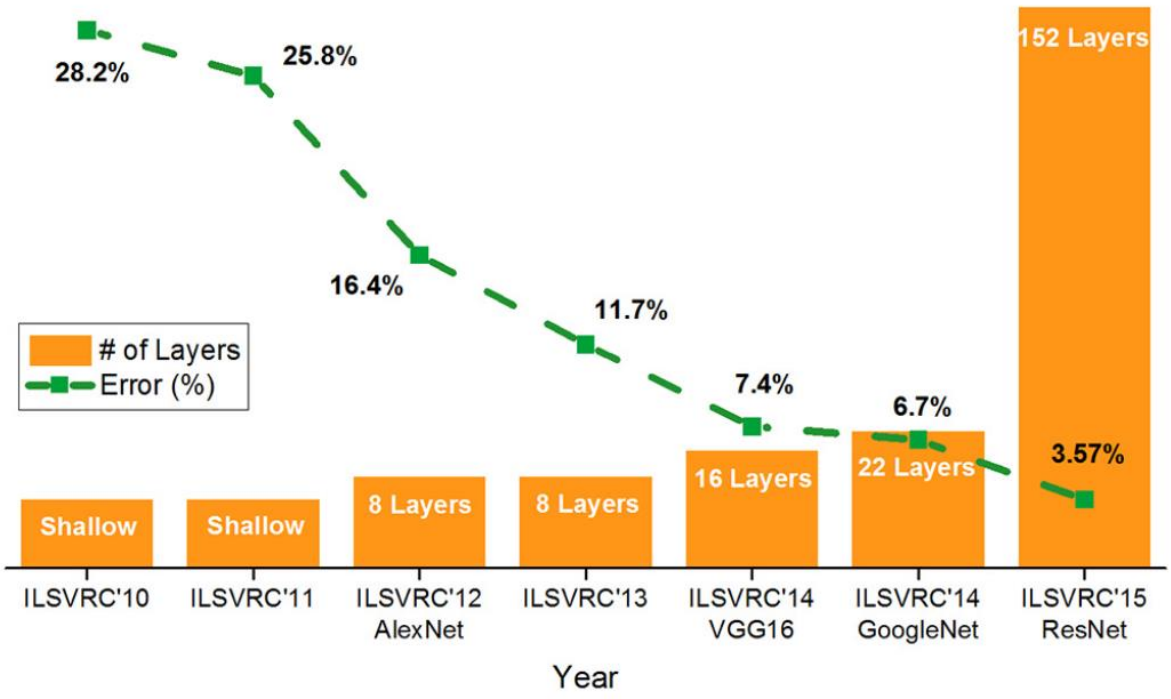


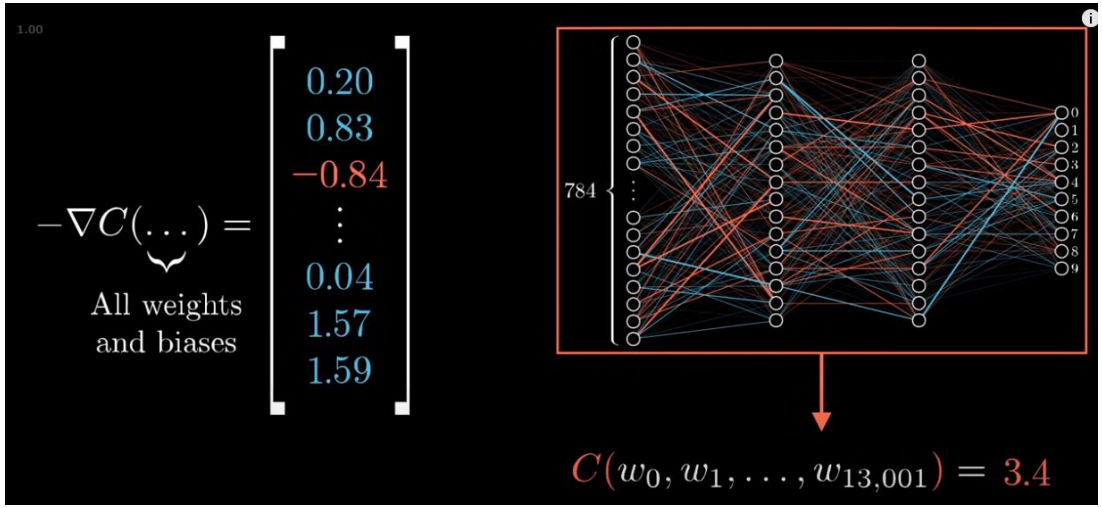
Figure 1: The performance and size of the DNNs in ILSVRC'10-15 [17, 22, 41, 43].

Source : Wang, Ji & Zhang, Jianguo & Bao, Weidong & Zhu, Xiaomin & Cao, Bokai & Yu, Philip. (2018). Not Just Privacy: Improving Performance of Private Deep Learning in Mobile Cloud.

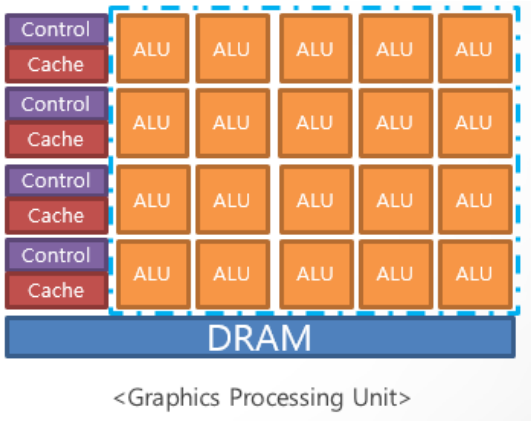
Introduction

❖ 수많은 파라미터를 잘 처리해야합니다.

- MNIST 예제 13002개
- AlexNet(2012) 60M개
- ResNet-152(2015) 11B개



→ GPU가 필요함



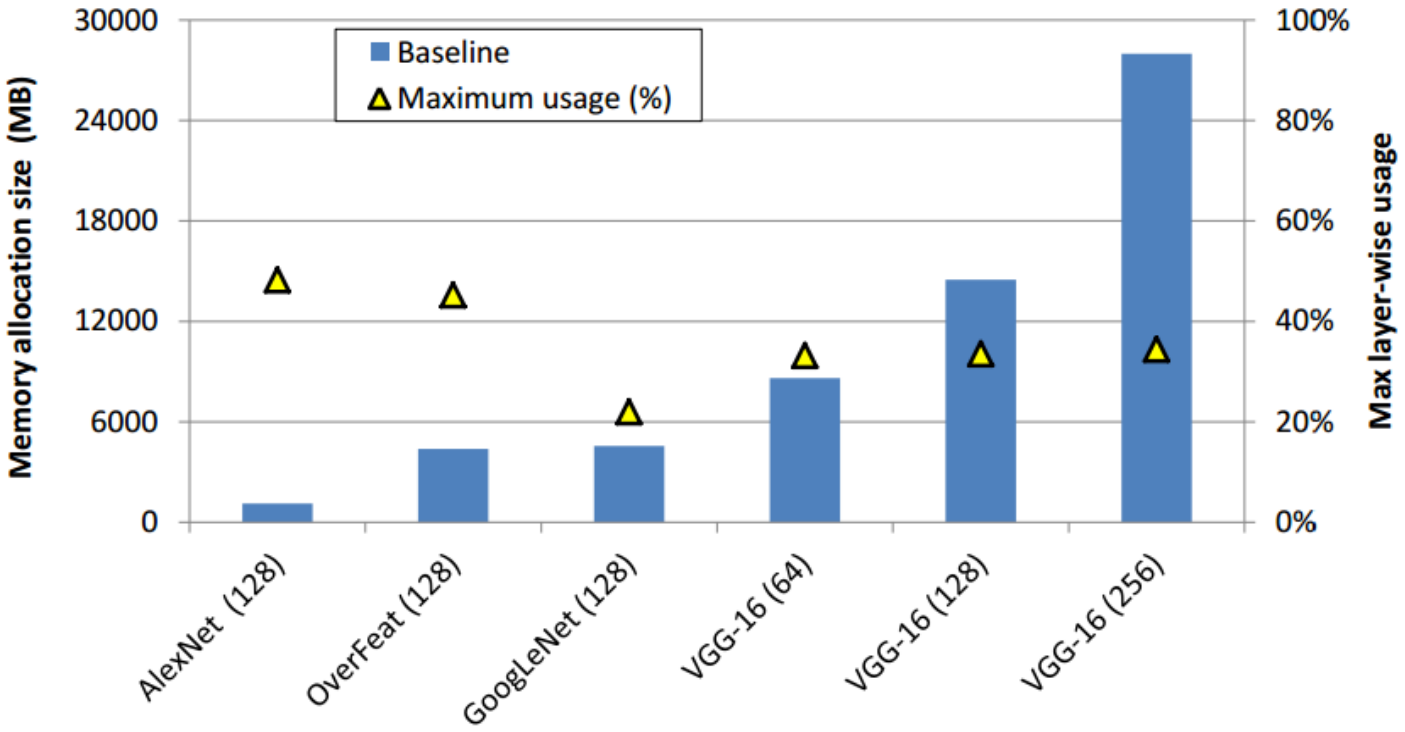
<Graphics Processing Unit>

Source : https://www.youtube.com/watch?v=Ilg3gGewQ5U&t=1m29s

Introduction

❖ 알고리즘은 점점 복잡해져 갑니다.

- 필요한 메모리가 늘어납니다.



Source : GPU memory usage when using the baseline, network-wide allocation policy (left axis). (Minsoo Rhu et al. 2016)

Introduction

❖ 해결책과 이슈

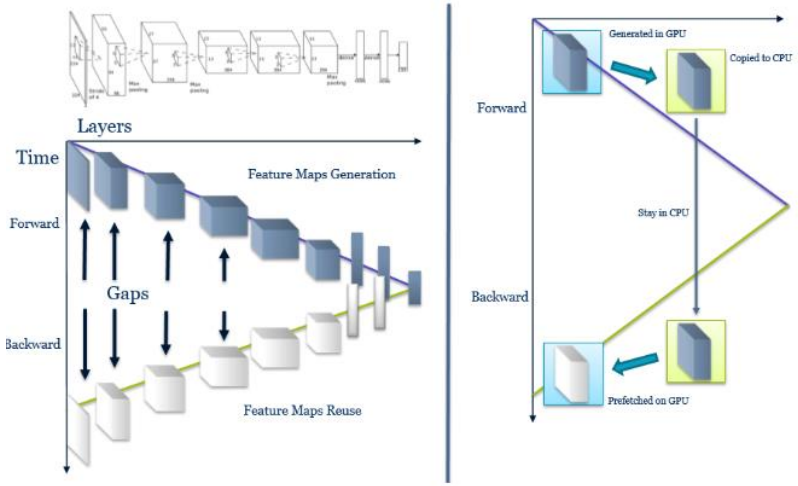
- 배치 사이즈 줄이기 → 학습 속도가 (거의) 비례하여 줄어듬

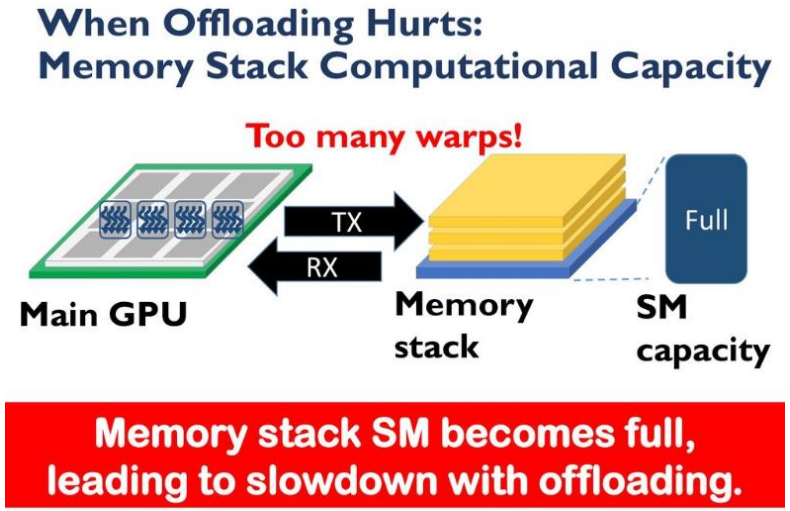
- 모델 사이즈 줄이기 → 모델 사이즈가 큰 것은 이유가 있음.
  자체로 큰 연구주제가 되는 부담.

Introduction

❖ 해결책과 이슈

- 배치 사이즈 줄이기 → 학습 속도가 (거의) 비례하여 줄어듬
- 모델 사이즈 줄이기 → 모델 사이즈가 큰 것은 이유가 있음.
  자체로 큰 연구주제가 되는 부담.
- Memory-offloading → 학습 속도가 줄어듬





Source : https://syncedreview.com/2017/04/29/how-to-train-a-very-large-and-deep-model-on-one-gpu/
https://slideplayer.com/slide/13081242/

Introduction

❖ 해결책과 이슈

- 배치 사이즈 줄이기 → 학습 속도가 (거의) 비례하여 줄어듬
- 모델 사이즈 줄이기 → 모델 사이즈가 큰 것은 이유가 있음.
  자체로 큰 연구주제가 되는 부담.
- Memory-offloading → 학습 속도가 줄어듬

- Multi-GPU 활용 → $$ 과 개발 난이도 상승
  → 개념을 이해하고 난이도(시행착오)를 줄이는 것이 오늘의 주제
  → 특히 “pytorch”에서 “데이터 분할 " 방식의 “multi-gpu” 활용
  → 그리고 숨겨진 무기 하나!

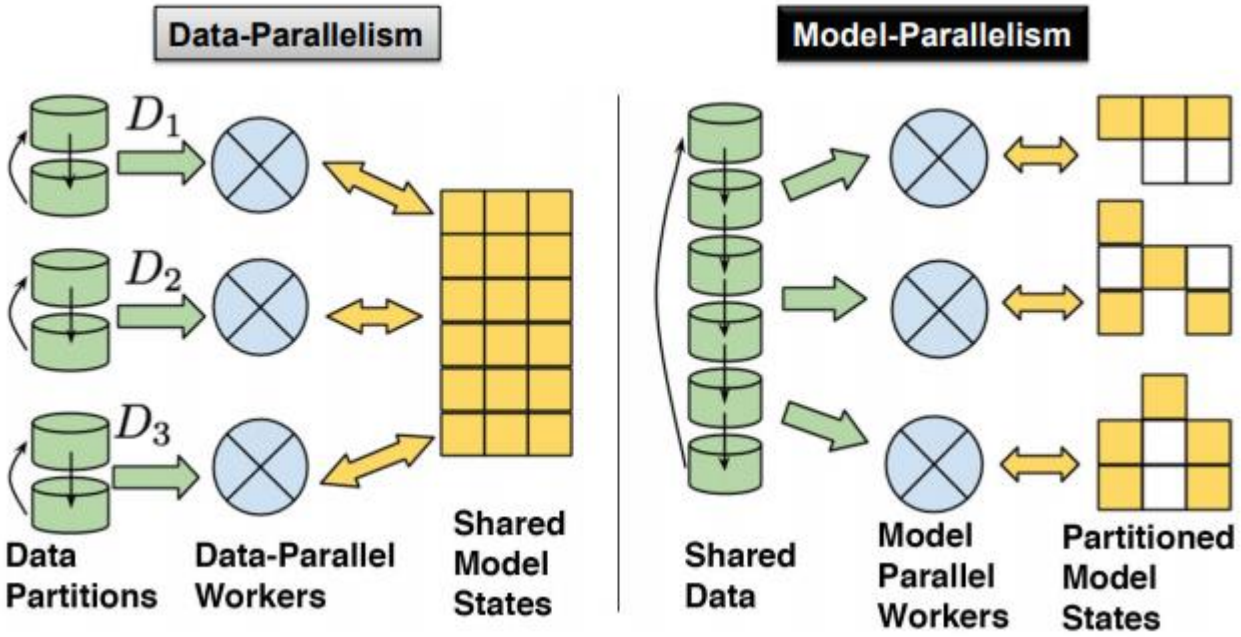
Distributed Training

- ❖ 작업 분할 방식
  - Model Parallelism
  - Data Parallelism
- ❖ 동기화 방식
  - Synchronous replication
  - Asynchronous replication
- ❖ 데이터 취합 방식
  - Parameter Server
  - Ring-AllReduce

Distributed Training

❖ 작업 분할 방식 : Parallelism

- Model Parallelism : 모델을 여러 GPU에 분산하여 처리하는 방법
- Data Parallelism : 데이터를 여러 GPU에 분산하여 처리하는 방법



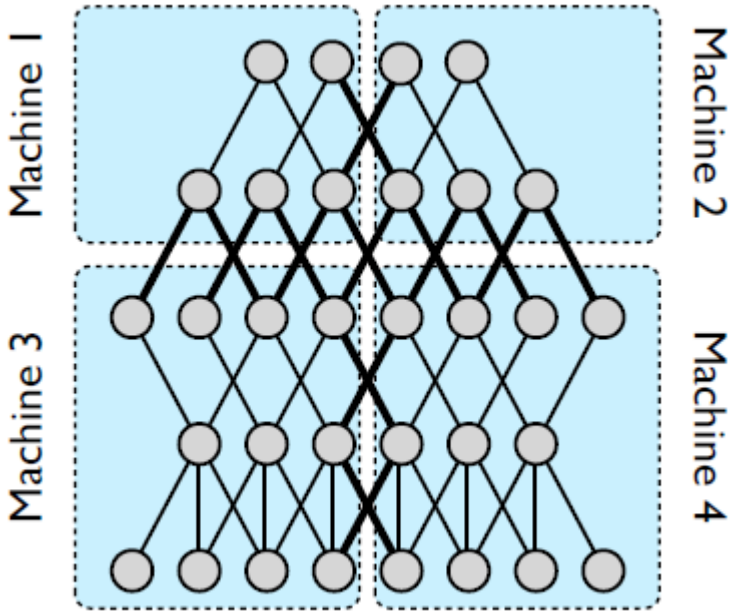
Source : Xing, Eric P., et al. "Petuum: A new platform for distributed machine learning on big data." IEEE Transactions on Big Data 1.2 (2015): 49-67.

Distributed Training

❖ 작업 분할 방식 : Model Parallelism

- 모델을 여러 GPU에 분산하여 처리하는 방법

- 모델이 큰 경우 사용 가능
- 모델을 "잘" 분할 해야함

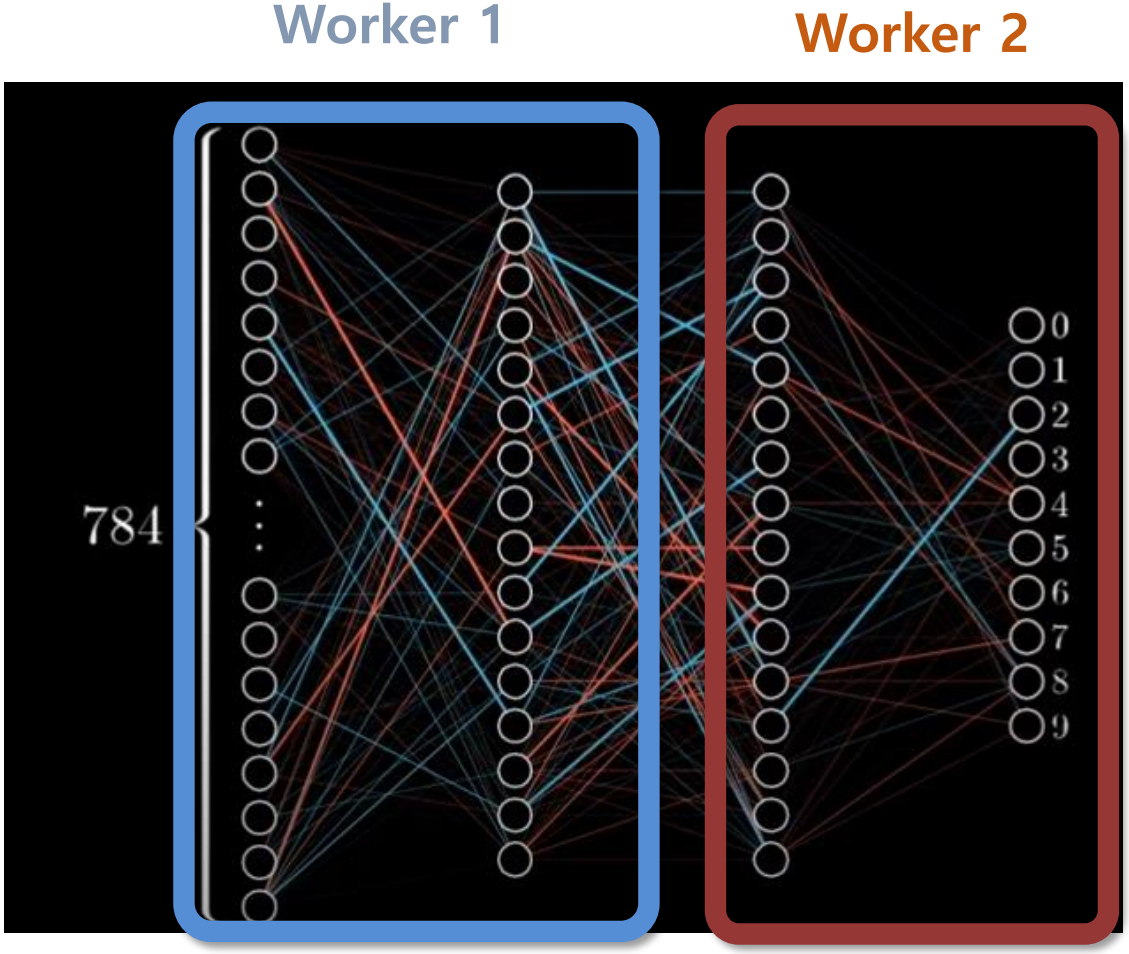


Source : Dean, Jeffrey, et al. "Large scale distributed deep networks." Advances in neural information processing systems. 2012. https://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf

Distributed Training

❖ 작업 분할 방식 :

Model Parallelism



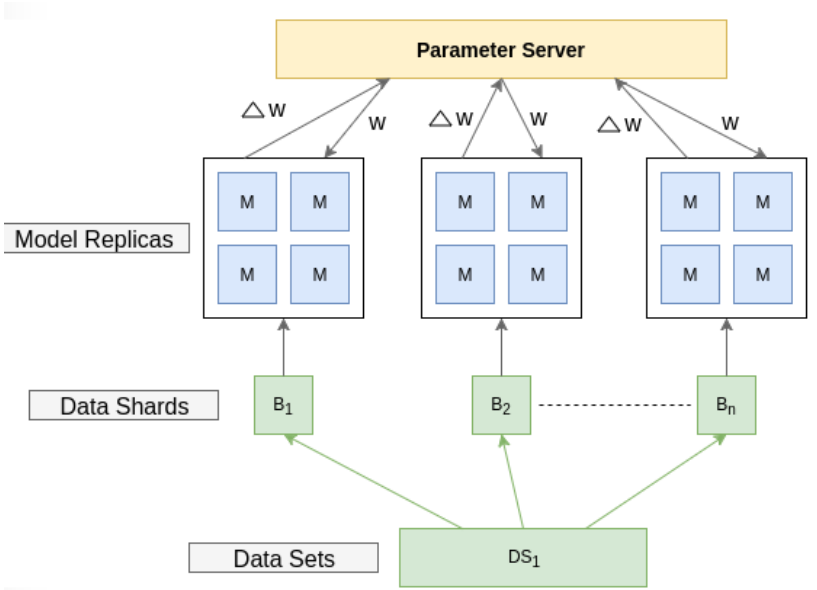
Source : http://www.youtube.com/watch?v=Ilg3gGewQ5U&t=10m45s

Distributed Training

❖ 작업 분할 방식 : Data Parallelism

- 데이터을 여러 GPU에 분산하여 처리하는 방법
- 각 데이터마다 weight를 계산
- weight를 통합 후 다음 iteration 처리

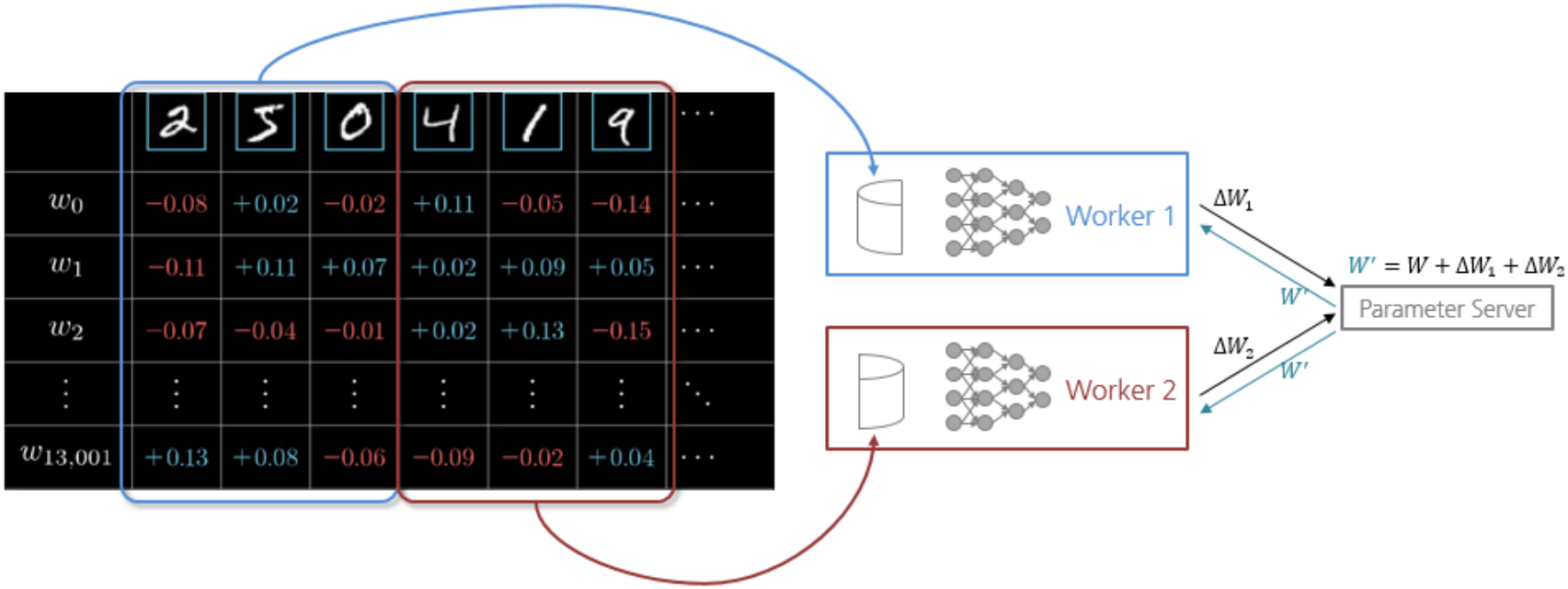
- 데이터는 항상 분리할 수 있으니
  모든 모델과 Framework에서 사용 가능
- 모든 weight 통합을 위한
  overhead 존재함.



Source : https://mc.ai/comparative-analysis-raven-protocol-v-s-conventional-methods/

Distributed Training

❖ 작업 분할 방식 : Data Parallelism



1. 학습 코드를 복수개 수행하면서 매 수행마다
   - 데이터 읽기
   - 모델에 데이터 넣고 수행하기
   - Gradient 계산하기
2. 복수개 수행 결과의 gradient 평균 구하기
3. 모델 업데이트하기
4. Goto 1

Distributed Training

❖ 작업 분할 방식 : Parallelism Example

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

3.2 Training on Multiple GPUs

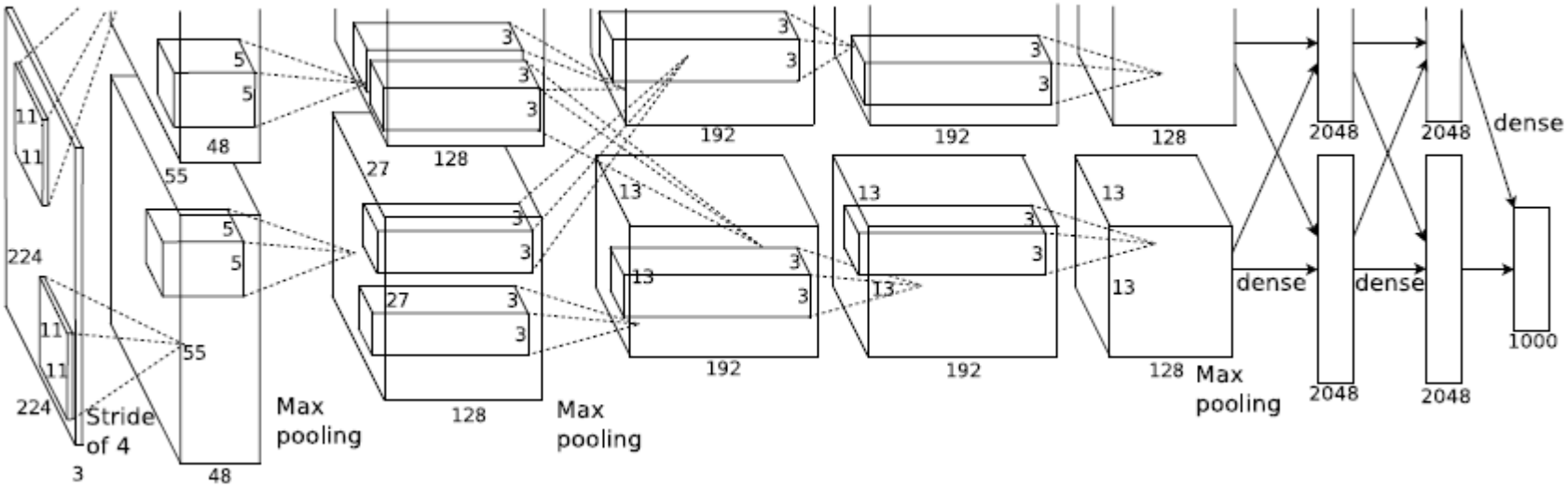
A single GTX 580 GPU has only 3GB of memory, which limits the maximum size of the networks that can be trained on it. It turns out that 1.2 million training examples are enough to train networks which are too big to fit on one GPU. Therefore we spread the net across two GPUs. Current GPUs are particularly well-suited to cross-GPU parallelization, as they are able to read from and write to one another's memory directly, without going through host machine memory. The parallelization scheme that we employ essentially puts half of the kernels (or neurons) on each GPU, with one

GTX 580 메모리가 3GB이기에 1.2M개의 Data를 학습할 수 없어서 2개의 GPU로 나눠서 학습했고 각 GPU는 서로 다른 GPU 메모리에 직접 write가 가능했다.

Source : Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

Distributed Training

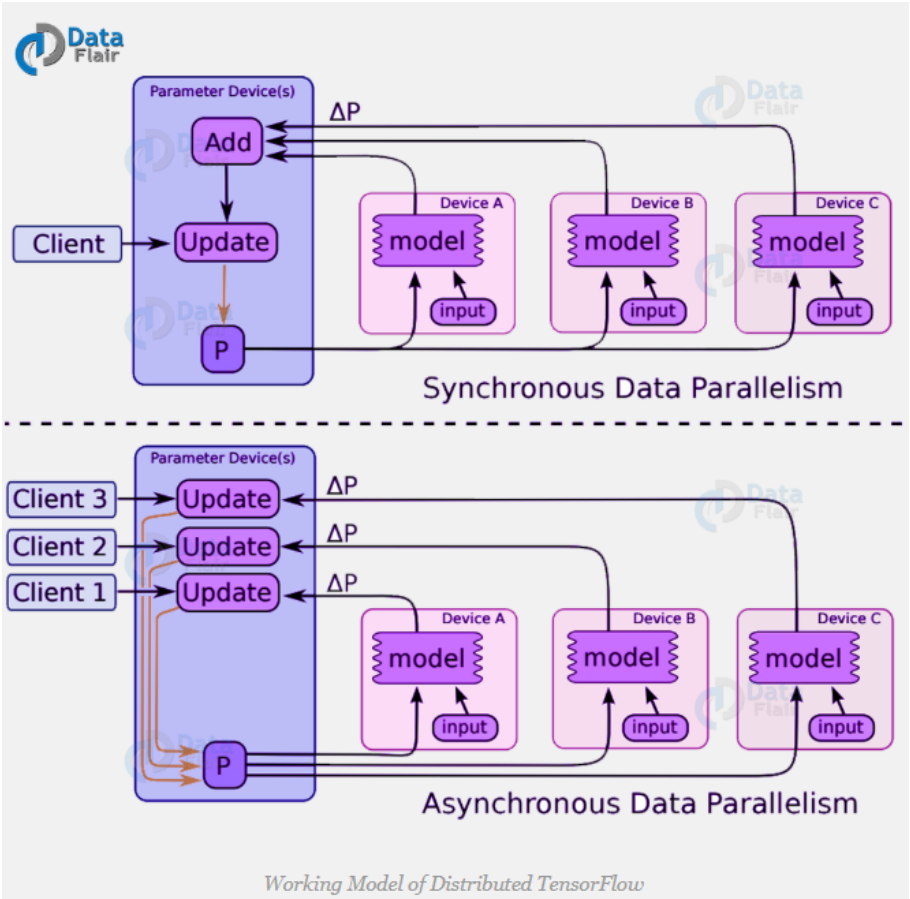
❖ 작업 분할 방식 : Parallelism Example



Source :Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

Distributed Training

❖ 동기화방식 : Synchronous vs Asynchronous replication



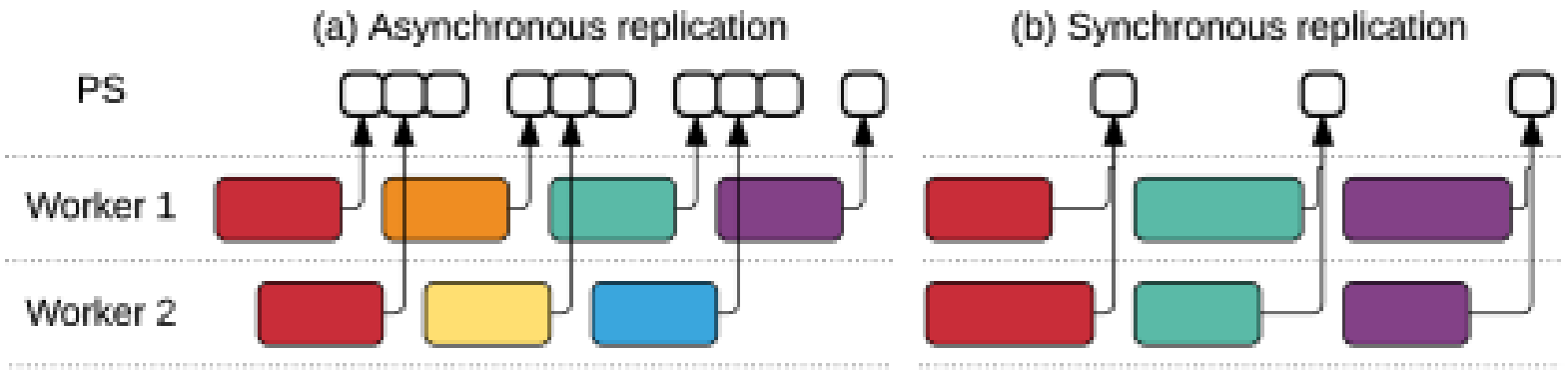
분산처리가 모두 끝났을 때
Parameter 업데이트
=> 수렴이 빠르다

각 분산 처리 job마다
Parameter 업데이트
=> 수렴이 느릴 수도 있다.

Source : https://data-flair.training/blogs/distributed-tensorflow/

Distributed Training

❖ 동기화방식 : Synchronous vs Asynchronous replication



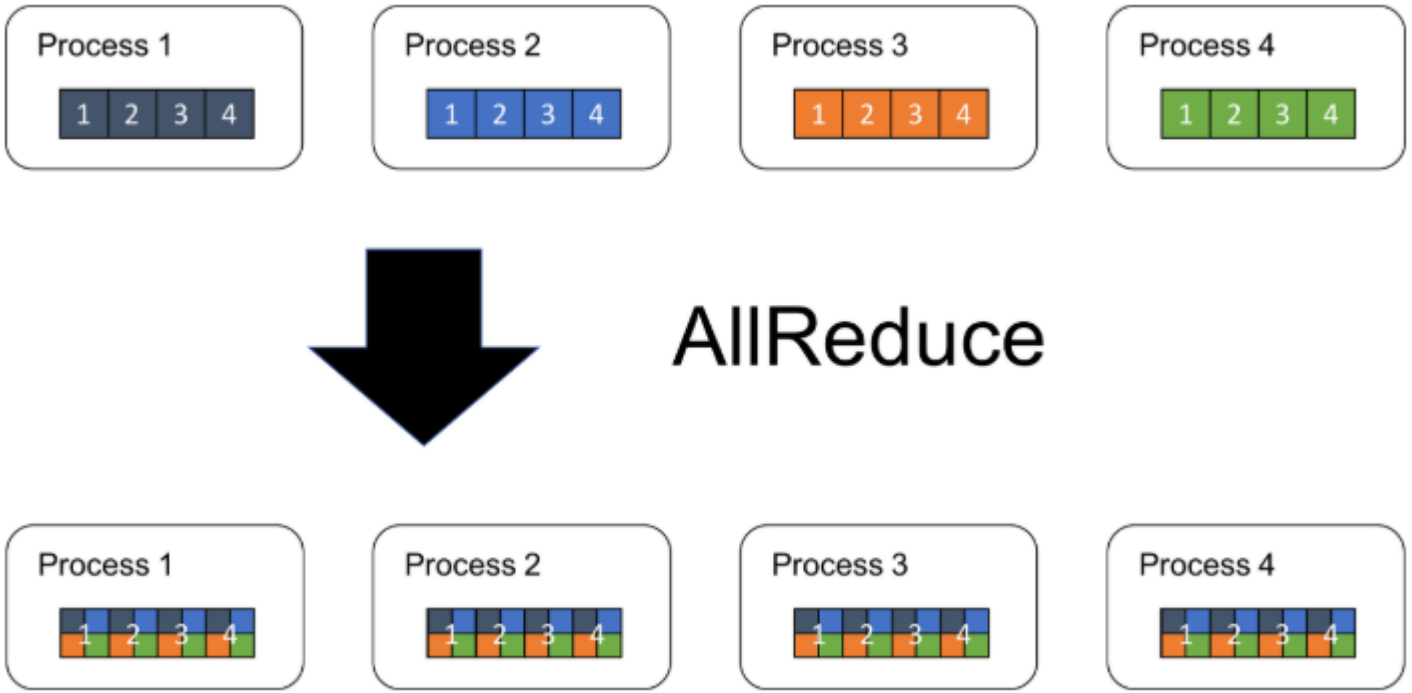
Sync 방식은 모든 worker(GPU)의 job이 끝날 때까지 대기.

-> 분산하는 worker(GPU)가 많을 수록 Async 방식이 효율적

Source : https://www.arxiv-vanity.com/papers/1605.08695/

Distributed Training

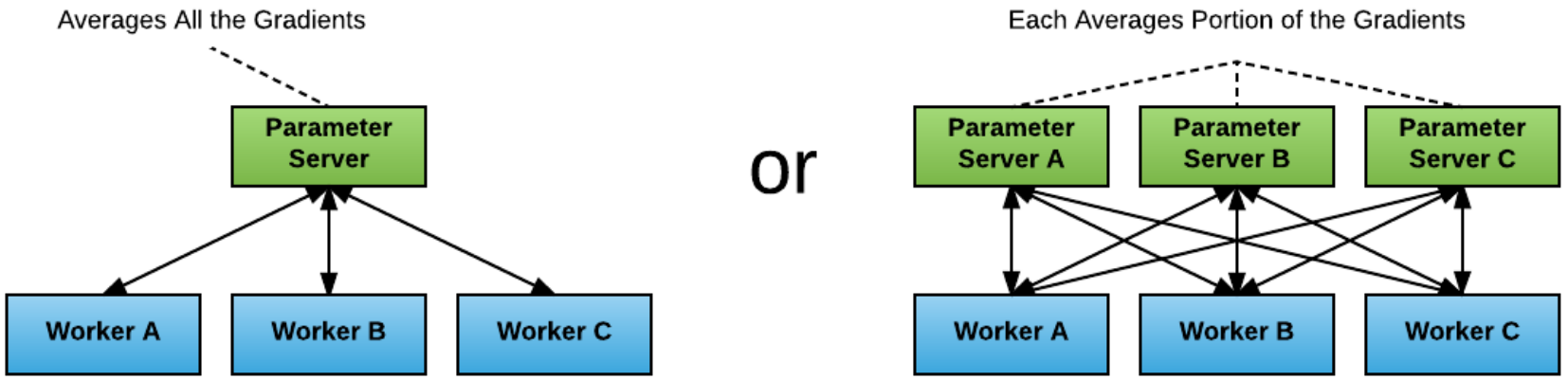
❖ 취합방식 : All-Reduce (a.k.a. Parameter Server)



Source : https://brunch.co.kr/@chris-song/96

Distributed Training

❖ 취합방식 : All-Reduce (a.k.a. Parameter Server)



- PS가 모든 gradient를 취합하여 재분배
- 장점 : 단순하게 사용 가능
- 단점 : PS에 메모리 사용량 및 네트워크 부하 집중

→ PS 개수에 따라 통합의 overhead와 network bandwidth 및 메모리 사용량 달라짐
→ 권장 초기 값 : No. PSs == No. Workers

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

Distributed Training

❖ 취합방식 : Ring-AllReduce (1st stage, 기본 컨셉 제공)

Bandwidth Optimal All-reduce Algorithms for Clusters of Workstations

Pitch Patarasuk Xin Yuan*
Department of Computer Science, Florida State University
Tallahassee, FL 32306
{patarasu, xyuan}@cs.fsu.edu

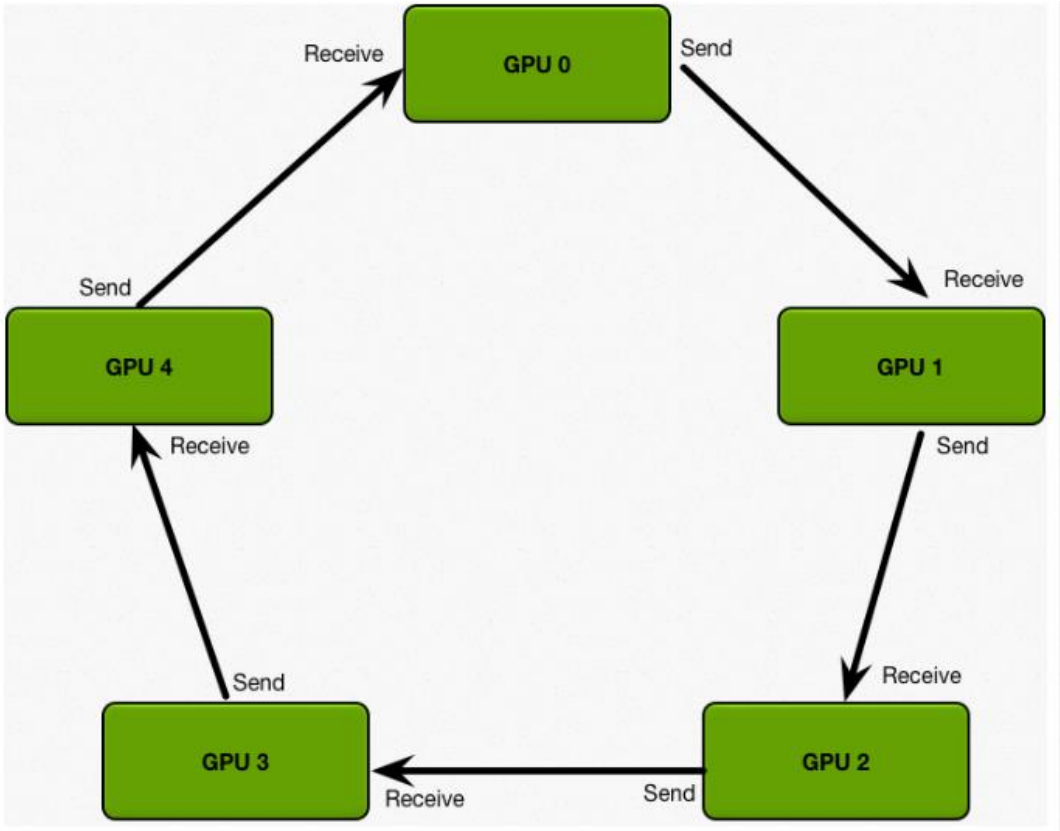
Abstract

We consider an efficient realization of the all-reduce operation with large data sizes in cluster environments, under the assumption that the reduce operator is associative and commutative. We derive a tight lower bound of the amount of data that must be communicated in order to complete this operation and propose a ring-based algorithm that only requires tree connectivity to achieve bandwidth optimality. Unlike the widely used butterfly like all-reduce algorithm that incurs network contention in

Source : Patarasuk, Pitch, and Xin Yuan. "Bandwidth optimal all-reduce algorithms for clusters of workstations." Journal of Parallel and Distributed Computing 69.2 (2009): 117-124.

Distributed Training

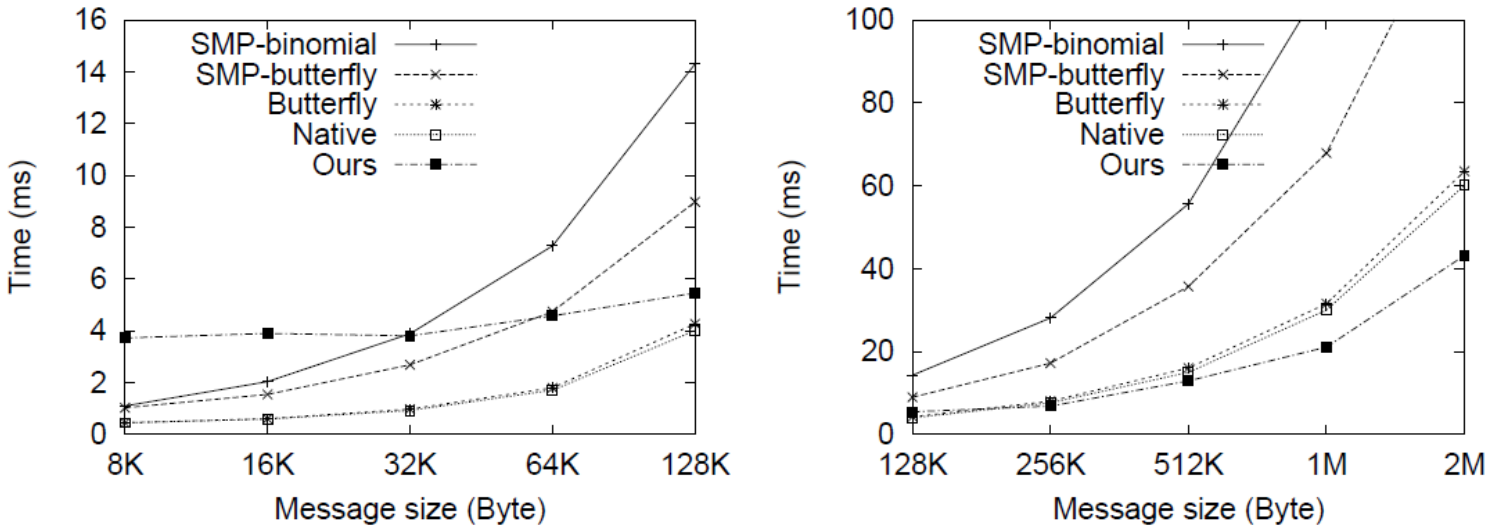
❖ 취합방식 : Ring-AllReduce (1st stage, 기본 컨셉 제공)



Source : https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

Distributed Training

❖ 취합방식 : Ring-AllReduce (1st stage, 기본 컨셉 제공)



(a) Medium sized data

(b) Large sized data

Figure 3: Results on the NCSA Teragrid IA-64 cluster (128 processors, Myrinet)

Parameter가 많아서 주고 받는 데이터가 클 때 의미가 있음! (네트워크 부하 감소)

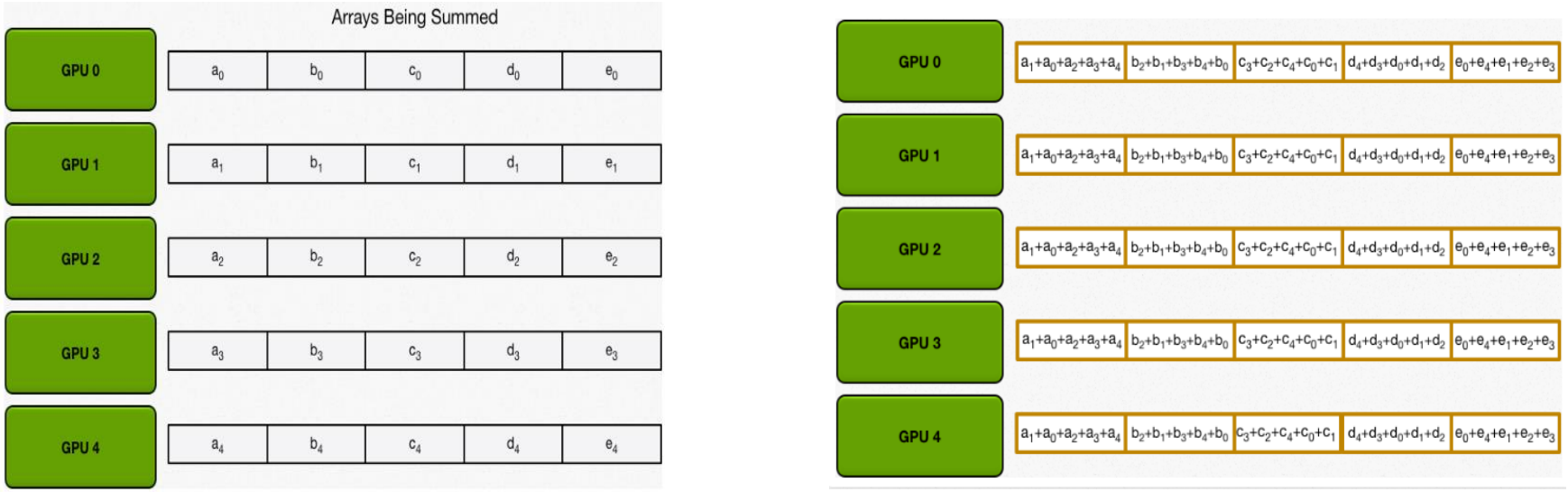
Source : Patarasuk, Pitch, and Xin Yuan. "Bandwidth optimal all-reduce algorithms for clusters of workstations." Journal of Parallel and Distributed Computing 69.2 (2009): 117-124.

Distributed Training

❖ 취합방식 : Ring-AllReduce (2st stage, 알고리즘 완성, Baidu Tech Blog)

Bringing HPC Techniques to Deep Learning

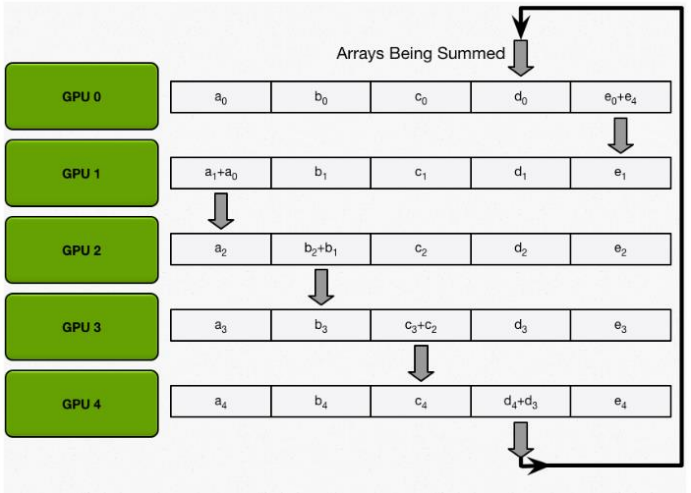
Start → Goal



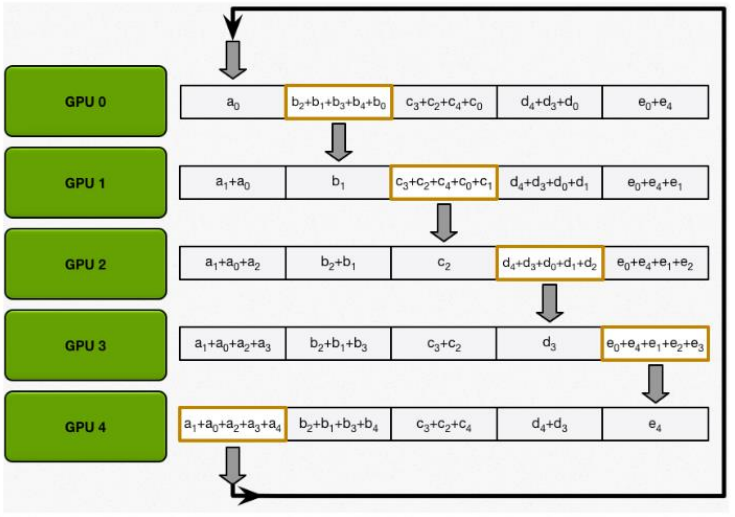
Source : https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/.

Distributed Training

- 취합방식 : Ring-AllReduce (2st stage, 알고리즘 완성, Baidu Tech Blog)
  - 1. Scatter-reduce
    - GPU를 순환하며 파라미터의 일부를 보냄



  - 2. AllGather
    - 완성된 파라미터를 공유



Source : https://andrew.gibiansky.com/blog/machine-learning/baidu-allredu

Distributed Training

❖ 취합방식 : Ring-AllReduce (3rd stage, Open Source, Uber, Horovod)

Horovod: fast and easy distributed deep learning in TensorFlow

Alexander Sergeev
Uber Technologies, Inc.
asergeev@uber.com

Mike Del Balso
Uber Technologies, Inc.
mdb@uber.com

Source : Sergeev, A., & Del Balso, M. (2018). *Horovod: fast and easy distributed deep learning in TensorFlow*. Retrieved from http://arxiv.org/abs/1802.05799

Distributed Training

- 취합 방식에 따른 네트워크 부하
- GPU 개수 P개, Parameter개수 N개 인 경우 1개 GPU의 최대 데이터 전송량
  - All-Reduce
    - N * (P - 1)
  - Ring All-Reduce
    - 2 * (N / P) * (P – 1)

Source : https://brunch.co.kr/@chris-song/96

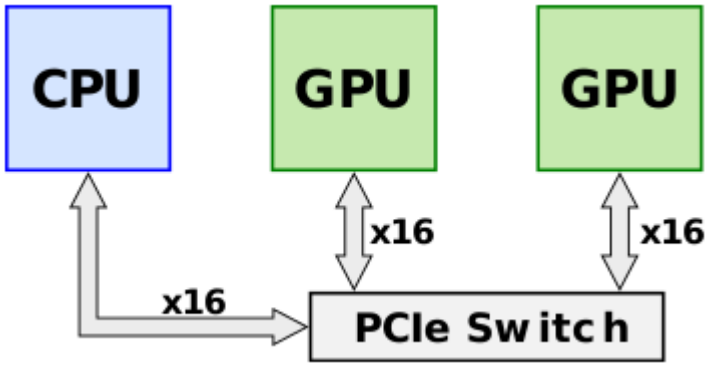
Distributed Training

- HW적 연결상태
  - PCIe
  - NV-SLI
  - Nvlink
  - Nvswitch
  - GPUDirect-RDMA

- NVSwtich > Nvlink > NV-SLI > PCIe

Distributed Training

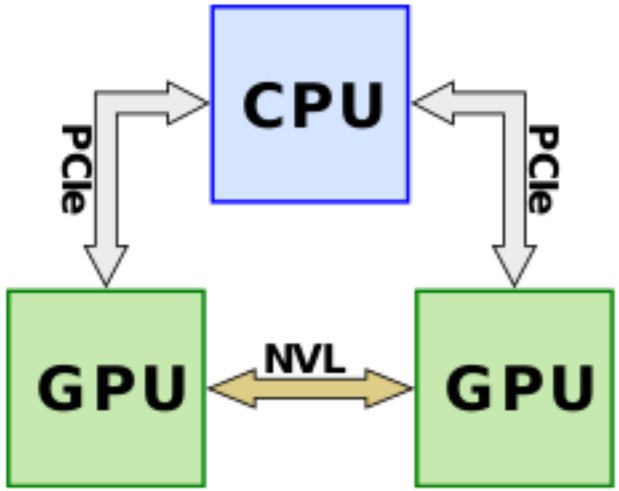
- HW적 연결상태
  - PCIe



Source : https://en.wikichip.org/wiki/nvidia/nvlink

Distributed Training

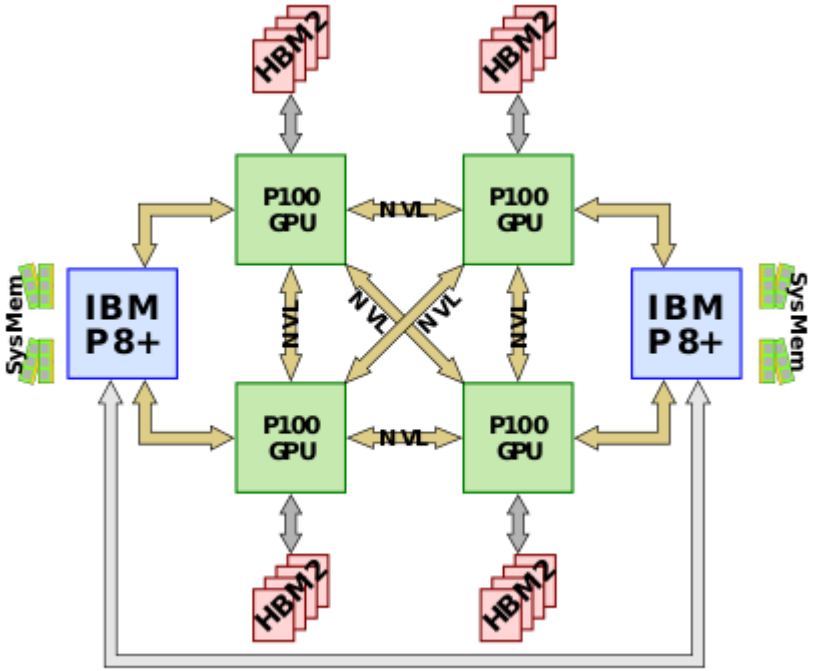
- HW적 연결상태
  - NV-SLI

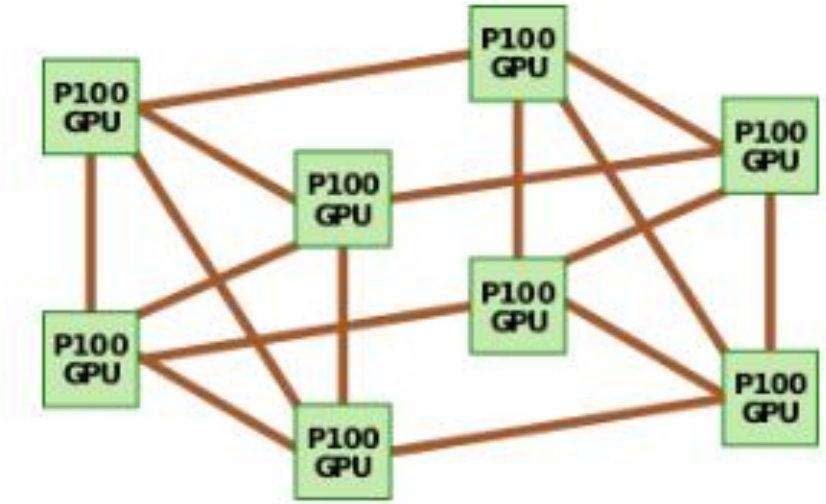


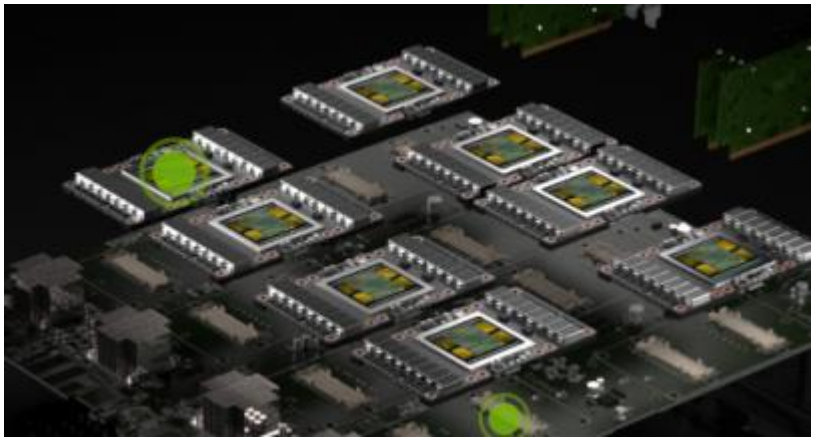
Source : https://en.wikichip.org/wiki/nvidia/nvlink

Distributed Training

- HW적 연결상태
  - NV-link



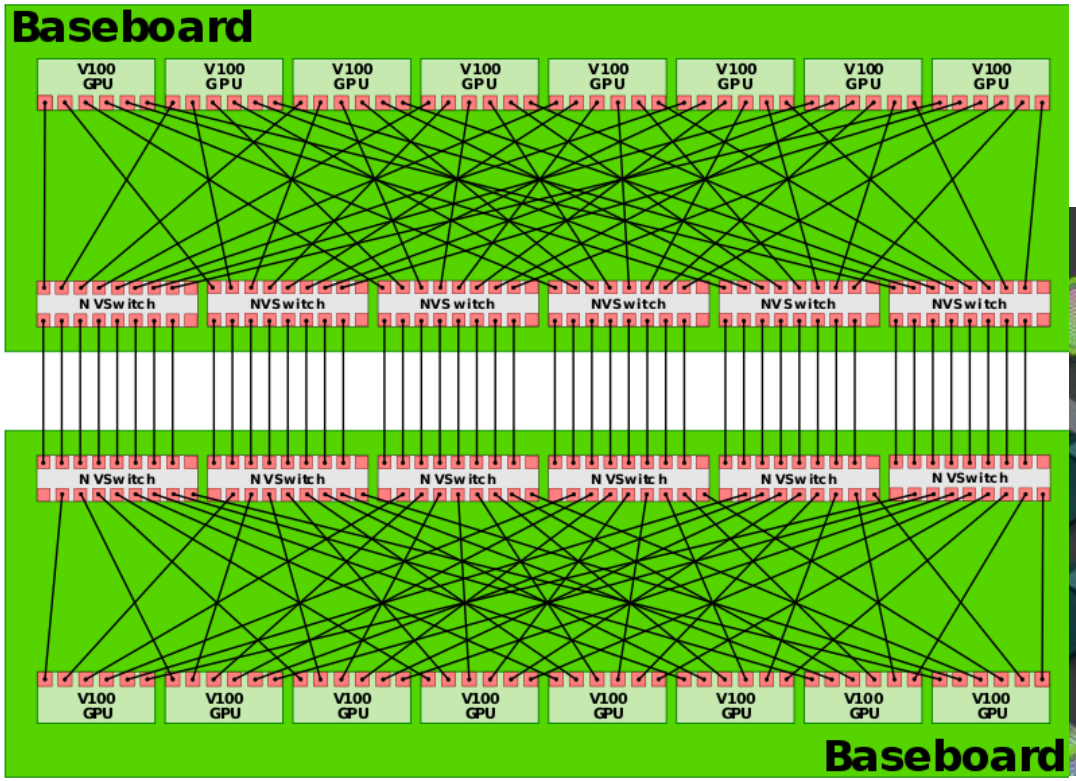


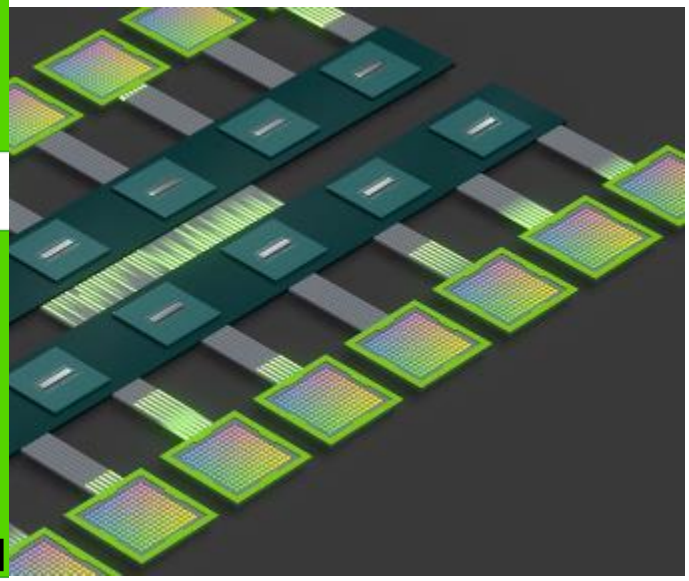


Source : https://en.wikichip.org/wiki/nvidia/nvlink

Distributed Training

- HW적 연결상태
  - NV-switch

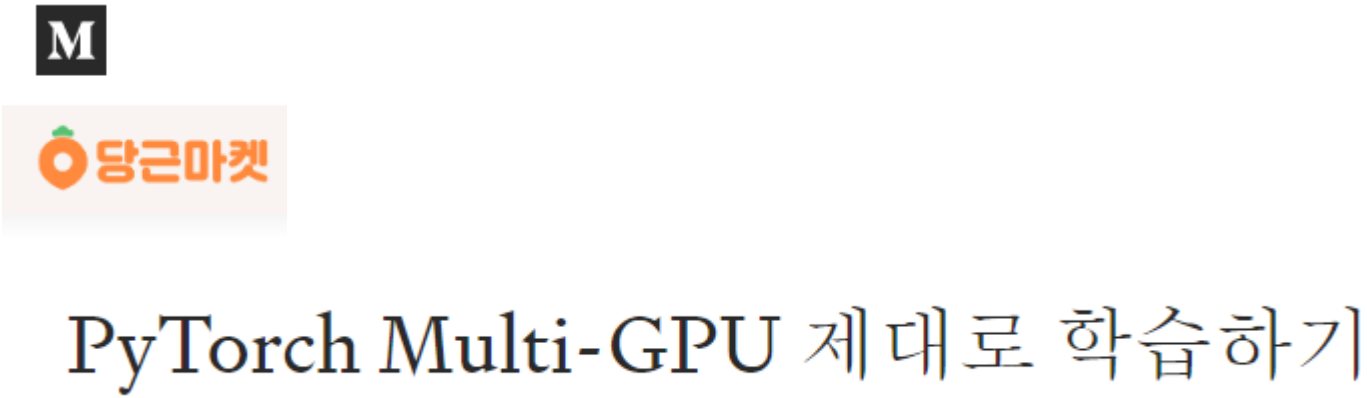




Source : https://en.wikichip.org/wiki/nvidia/nvlink

속도 개선을 위해 무엇을 보는가?

❖ Pytorch로 분산학습을 하면서 얻어가는 Tip..



Source : https://bit.ly/304KWrJ

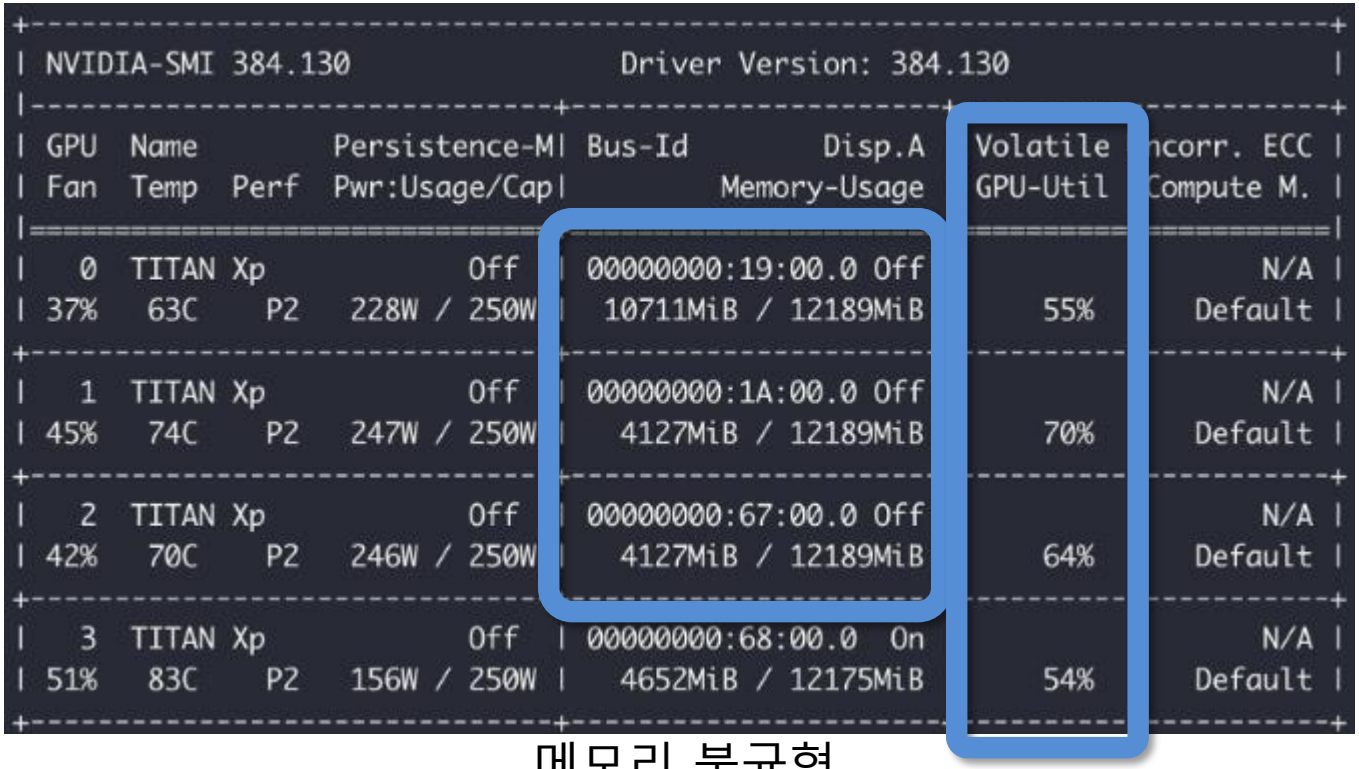
속도 개선을 위해 무엇을 보는가?

- 앞에서 많이 했으니 빠르게 지나갑니다.
  - Data Parallel => 1 parameter server in 1 node
    - 여러 개 GPU를 사용할 수 있지만 메모리 불균형
  - Custom Data Parallel => 1 parameter server in 1 node
    - 메모리 불균형을 아주 조금 해소
    - GPU utilization이 낮음
  - Distributed Data Parallel => multi parameter servers in 1 node
    - 사용하지 않는 Parameter로 인한 간헐적 오류
  - Nvidia Apex => multi parameter servers in 1 node
    - 일단 만족
  - 여러 대의 컴퓨터를 활용하는 방법
    => multi parameter servers in multi nodes with all-reduce method
  - 결론 : 마지막 페이지에 밝힙니다.

Source : https://bit.ly/304KWrJ
https://medium.com/huggingface/training-larger-batches-practical-tips-on-1-gpu-multi-gpu-distributed-setups-ec88c3e51255

속도 개선을 위해 무엇을 보는가?

❖ 무엇을 보고 개선이 필요한지 결정하는가?



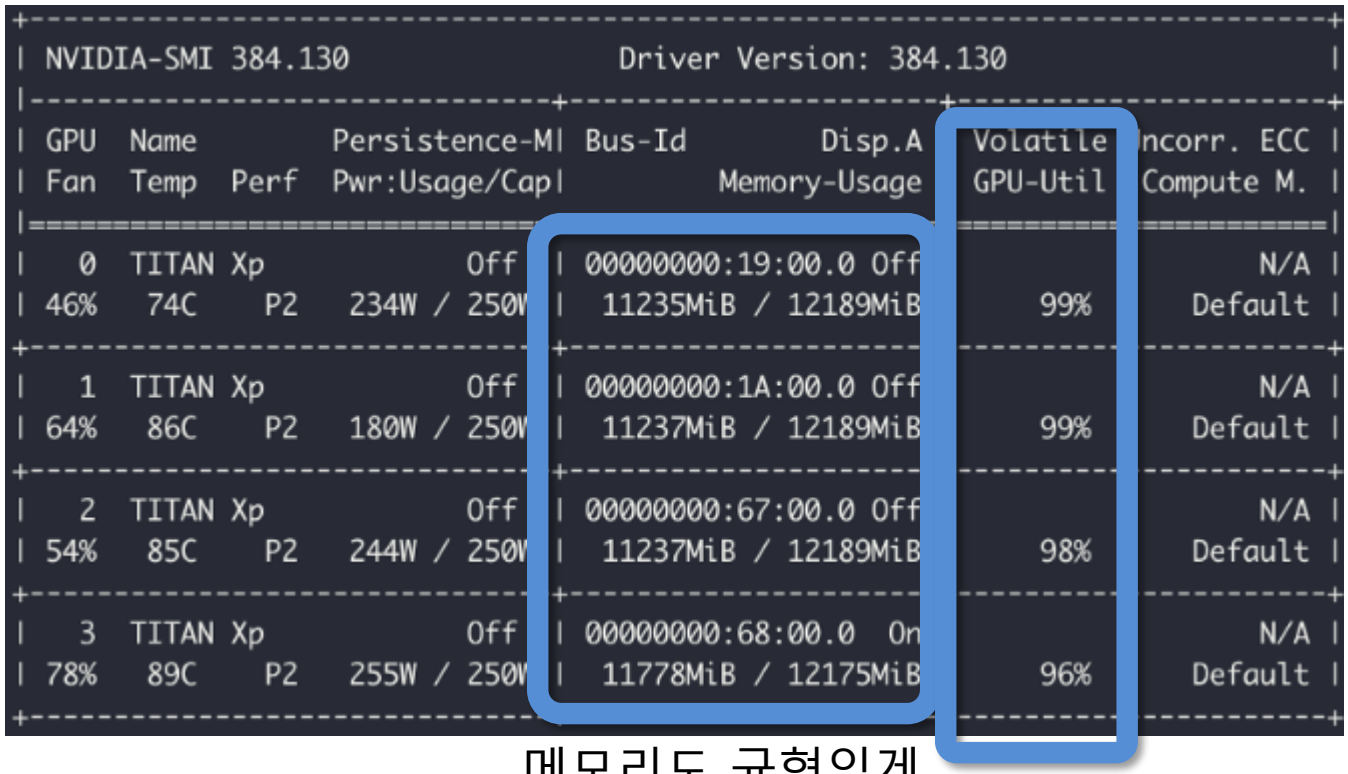
메모리 불균형

GPU 내부의 cuda core를
제대로 활용하지 못함

Source : https://bit.ly/304KWrJ
https://medium.com/huggingface/training-larger-batches-practical-tips-on-1-gpu-multi-gpu-distributed-setups-ec88c3e51255

속도 개선을 위해 무엇을 보는가?

❖ 최종의 아름다운 모습



메모리도 균형있게
모두 사용

GPU 내부의 cuda core도 100% 사용

Source : https://bit.ly/304KWrJ
https://medium.com/huggingface/training-larger-batches-practical-tips-on-1-gpu-multi-gpu-distributed-setups-ec88c3e51255

속도 개선을 위해 무엇을 보는가?

```
from torch.utils.data.distributed import DistributedSampler
from torch.utils.data import DataLoader

# Each process runs on 1 GPU device specified by the local_rank argument.
parser = argparse.ArgumentParser()
parser.add_argument("--local_rank", type=int)

# Initializes the distributed backend which will take care of sychronizing nodes/GPUs
torch.distributed.init_process_group(backend='nccl')

# Encapsulate the model on the GPU assigned to the current process
device = torch.device('cuda', arg.local_rank)
model = model.to(device)
distrib_model = torch.nn.parallel.DistributedDataParallel(model,
                                                          device_ids=[args.local_rank],
                                                          output_device=args.local_rank)

# Restricts data loading to a subset of the dataset exclusive to the current process
sampler = DistributedSampler(dataset)

dataloader = DataLoader(dataset, sampler=sampler)
for inputs, labels in dataloader:
    predictions = distrib_model(inputs.to(device))          # Forward pass
    loss = loss_function(predictions, labels.to(device))    # Compute loss function
    loss.backward()                                         # Backward pass
    optimizer.step()                                        # Optimizer step
```

Source : https://medium.com/huggingface/training-larger-batches-practical-tips-on-1-gpu-multi-gpu-distributed-setups-ec88c3e51255

속도 개선을 위해 무엇을 보는가?

```
python -m torch.distributed.launch --nproc_per_node=4 --nnodes=2 --
node_rank=0 --master_addr="192.168.1.1" --master_port=1234
OUR_TRAINING_SCRIPT.py (--arg1 --arg2 --arg3 and all other arguments
of our training script)
```

On the second machine we similarly start our script:

```
python -m torch.distributed.launch --nproc_per_node=4 --nnodes=2 --
node_rank=1 --master_addr="192.168.1.1" --master_port=1234
OUR_TRAINING_SCRIPT.py (--arg1 --arg2 --arg3 and all other arguments
of our training script)
```

속도 개선을 위해 무엇을 보는가?

❖ 예산과 공간에 제약이 없다면 누구나 할 수 있습니다.

NVIDIA Was The First To Train BERT-Large In Under An Hour. NVIDIA

(92 * DGX-2H , 1472 * V100)

- NVIDIA도 사용한 방법은 비슷합니다.

```
python -m torch.distributed.launch $DISTRIBUTED_ARGS ./pretrain_bert.py \
                $BERT_ARGS \
                $OUTPUT_ARGS \
                --save $CHECKPOINT_PATH \
                --load $CHECKPOINT_PATH \
                --data-path $DATA_PATH \
                --model-parallel-size $MP_SIZE \
                --DDP-impl torch
```

Summary

- If **느린 것을 모르겠으면**
  - 그냥 GPU 1장 쓰세요.
  - 좋은 GPU에서는 1장으로도 잘 돌아갑니다.
  - 오히려 분산하는데 부하가 생깁니다.
- Elseif **nvidia-smi로 살펴본 GPU Core Utilization이 낮으면**
  - 그냥 GPU 1장 쓰고 코드를 개선해보세요.
- Elseif **업데이트할 파라미터 개수가 적은 작은 모델이면**
  - GPU간 네트워킹에 부하가 별로 없을테니
  - 어떤 분산 방법을 쓰던지 상관없습니다.
  - 분산을 하기만 하세요. 그럼 빨라집니다.
- Else
  - **kubeflow 등의 편리한 분산 기능을 제공하는 platform을 적용하기 전까지는**
  - **고민하지 말고 Horovod를 쓰세요.**

감사합니다.