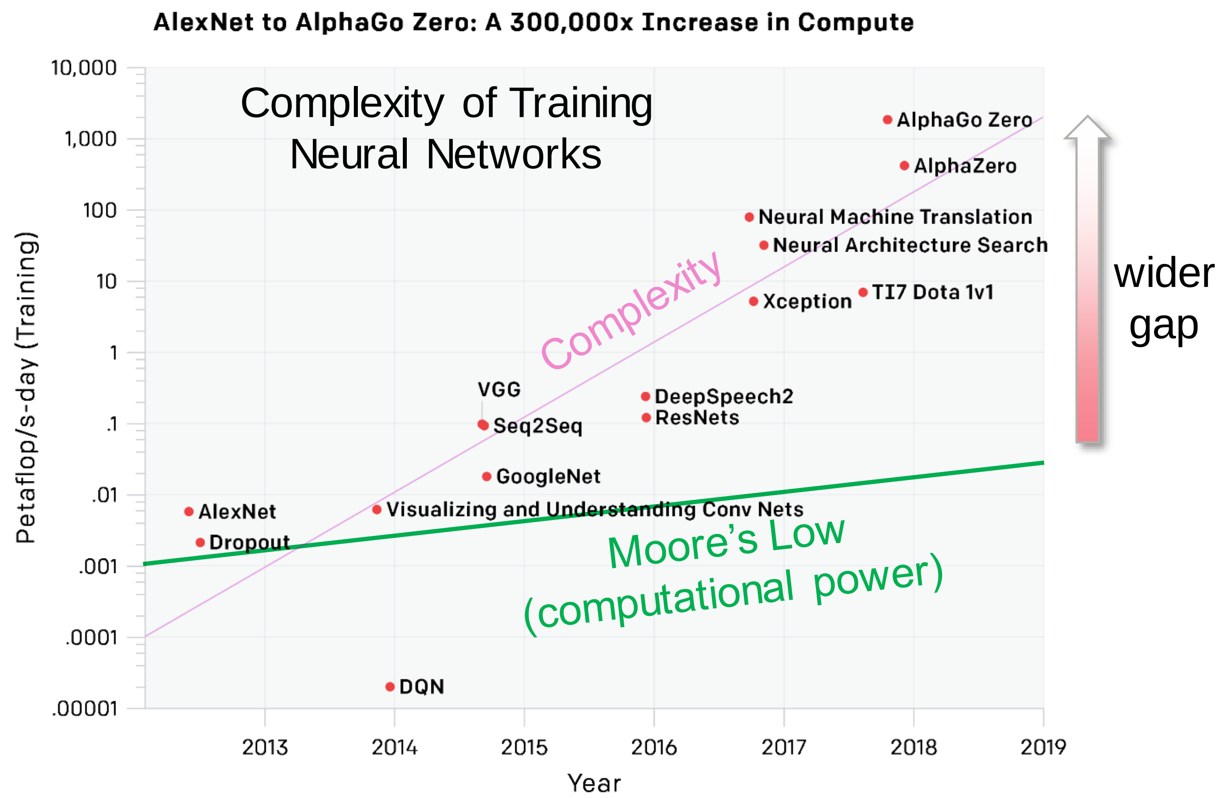
8th ADAC Workshop

Yet Another Accelerated SGD: ResNet-50 Training in 70.4 sec.

October 30, 2019

Fujitsu Laboratories. LTD.
senior researcher
Masafumi Yamazaki

Background



Source : Dario Amodei and Danny Hernandez, https://openai.com/blog/ai-and-compute/

- The scale of Deep Neural Network (DNN) is growing

The large-scale parallelization is one of the efficient way to accelerate training speed.

Contents

FUJITSU

1. Distributed Deep Learning

- History of DNN training speed challenges
- Our contributions

2. Key points for distributed training

- DNN training using Data parallel Synchronous-SGD
- Optimal mini-batch size
- Allreduce algorithms

3. Optimization Points

- Backward/Allreduce parallelization
- Optimization of initialization and unnecessary processing
- Optimization for MLPerf v0.6 (The Rule was changed from v0.5)

4. Evaluation and results

History of training speed of DNN

2012

ILSVRC2012

- **Barklay U.** AlexNet

  training in 5-6 days using 2 GPUs

2014

ILSVRC2014

- **google** GoogleNet

  training in a week using few GPUs
  (estimation in their paper)

2015

ILSVRC2015

- **MICROSOFT** ResNet

2016

2017

- **Facebook**

  ResNet-50 Training in 1 hour using 256 GPUs

- **PFN**

  Training in 15 Minutes using 1,024 GPUs

2018

- **Tencent**

  Training in 6.6 minute using 2,048 GPUs

- **Google**

  Training in 1.8 minute using 1,024 TPUs

2019

- **Sony**

  Training in 2.0 minute using 3,456 GPUs

…

Our contribution ... speed up

- In 2015, our group in Fujitsu Laboratories began working on large-scale distributed training

Year	Hardware	#GPUs	DNN / Dataset	Time	Remarks
Feb, 2016	Tatara (Kyushu Univ.)	64	AlexNet / ImageNet	-	
June, 2016	TSUBAME 2.5	256	AlexNet / ImageNet	-	*1
Aug., 2018	ABCI	~4096	ResNet-50 / ImageNet	(6.6 minute)	The Accuracy didn't reach 75%
April, 2019	ABCI	2048	ResNet-50 / ImageNet	74.7 seconds	arXiv:1903.12650
June, 2019	ABCI	2048	ResNet-50 / ImageNet	70.4 seconds	MLPerf v0.6

*1 SWoPP2016「MPIを用いたDL処理高速化の提案」
- evaluated the Allreduce algorithm
- proposed running computation and communication processes in parallel
- reported how accuracy worsened with large mini batch sizes.

History of training speed of DNN

2012

ILSVRC2012

- **Barklay U.** AlexNet
  training in 5-6 days using 2 GPUs

2014

ILSVRC2014

- **google** GoogleNet
  training in a week using few GPUs
  (estimation in their paper)

2015

ILSVRC2015

- **MICROSOFT** ResNet

2016

2017

- **Facebook**
  ResNet-50 Training in 1 hour using 256 GPUs
- **PFN**
  Training in 15 Minutes using 1,024 GPUs

2018

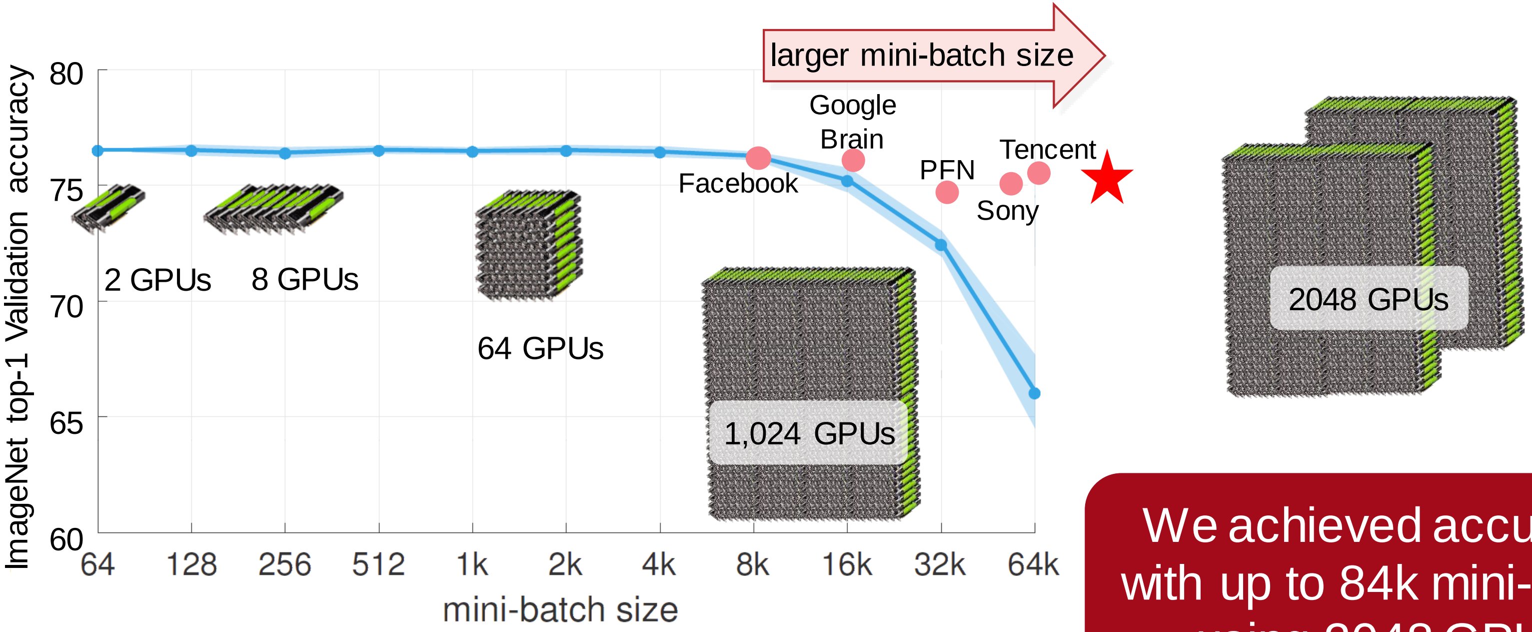
- **Tencent**
  Training in 6.6 minute using 2,048 GPUs
- **Google**
  Training in 1.8 minute using 1,024 TPUs

2019

- **Sony**
  Training in 2.0 minute using 3,456 GPUs
- **MLPerf v0.6 (Our work)**
  Training in 1.2 minute using 2,048 GPUs
- **Huawei**
  Training in 1.0 minute using 1,024 A910 processors

Our contribution ... mini-batch size

- Facebook was able to increase the mini-batch size up to 8k using ideas such as warm-ups. However, increasing the batch size further would worsen accuracy



We achieved accuracy with up to 84k mini-batch using 2048 GPUs

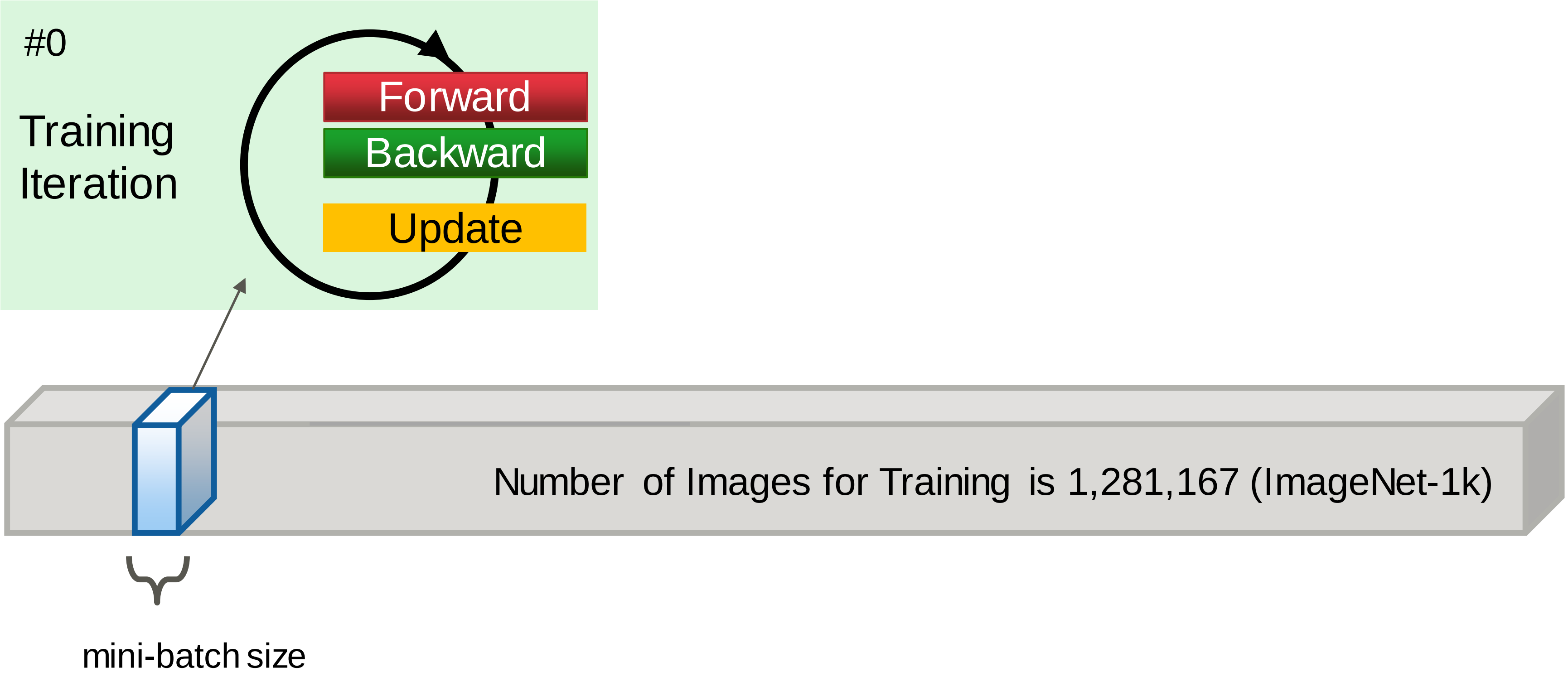
Source: Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, P. Goyal (Facebook) et al, 2017

Key points for distributed training

- Data Parallel Method Based on Synchronous-SGD using Allreduce
- Optimal mini-batch size
- Allreduce algorithms

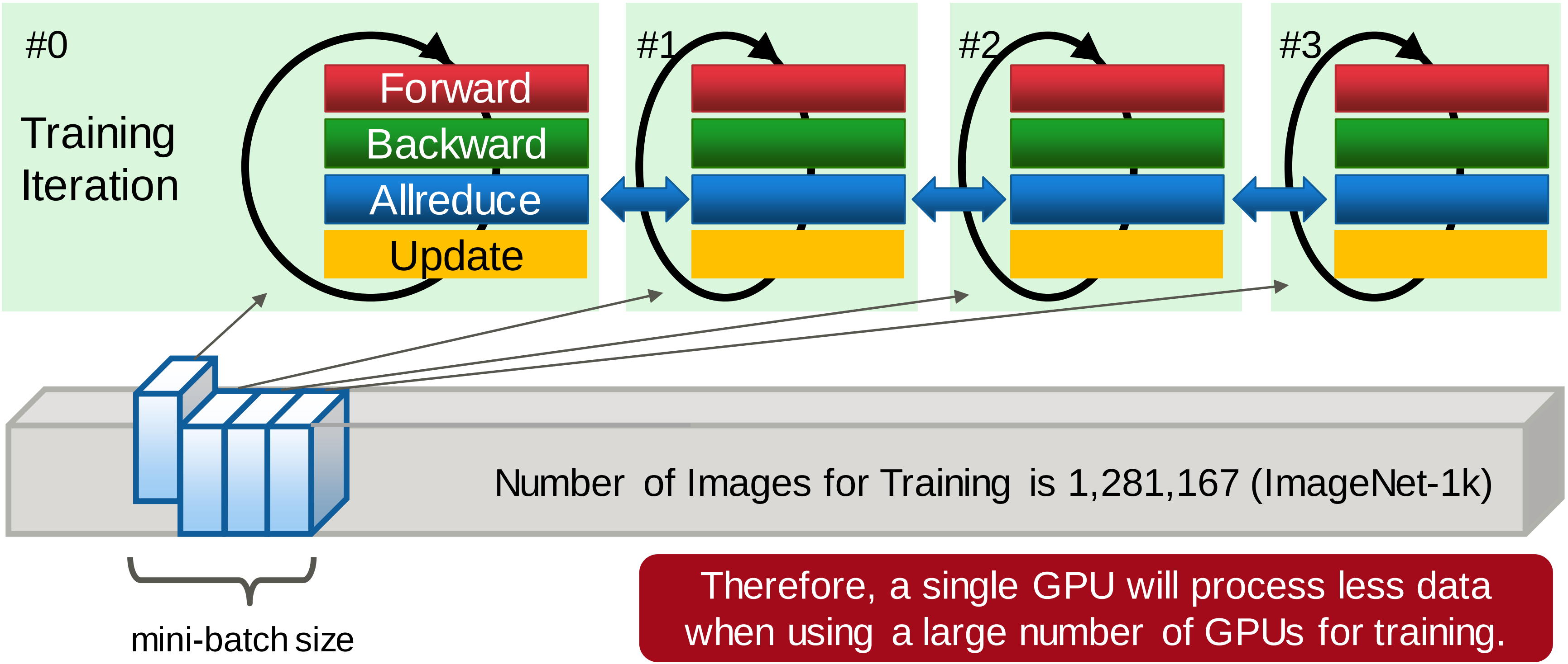
Data Parallel Method

- We have continued to accelerate training using a data parallel method based on synchronous-SGD using Allreduce



Data Parallel Method

- Since 2016, we have continued to accelerate training using a data parallel method based on synchronous-SGD using Allreduce

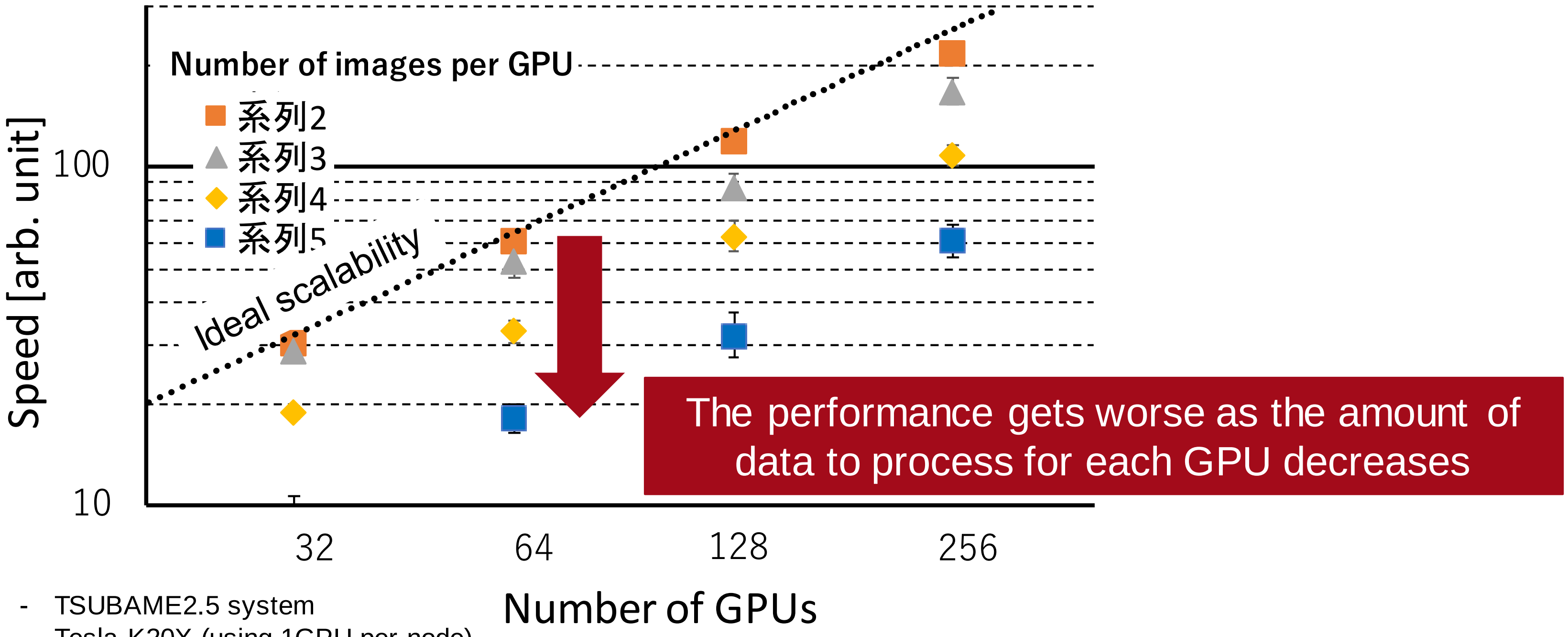


Therefore, a single GPU will process less data when using a large number of GPUs for training.

Optimal mini-batch size

FUJITSU

- The performance gets worse as the amount of data to process for each GPU decreases

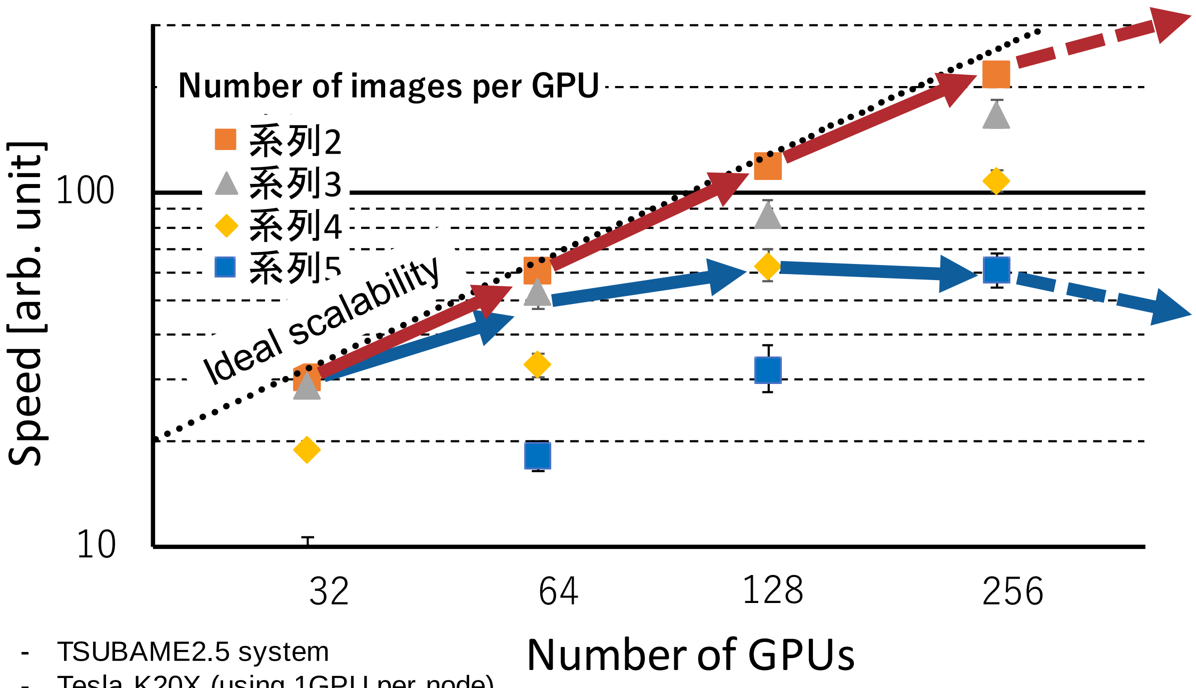


- TSUBAME2.5 system
- Tesla K20X (using 1GPU per node)

Optimal mini-batch size

FUJITSU

- We selected the optimal mini batch size for enough accuracy



- TSUBAME2.5 system
- Tesla K20X (using 1GPU per node)

Weak scale

Increases the amount of images per iteration in proportion to the number of accelerators
Pros; good scalability
Cons; accuracy down in a large mini batch

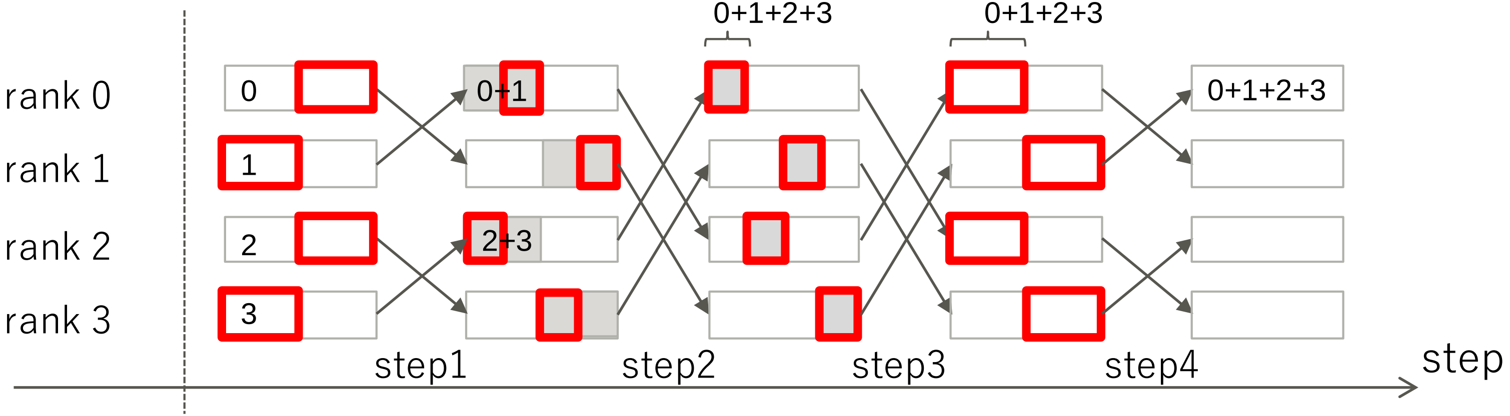
Strong scale

Execute the same process by dividing it with an accelerator
Pros; get the same result
Cons; Bad scalability
- **Reduced parallelism of GPU**
- **Communication overhead**

Comparison of Allreduce algorithms

algorithm	Data size to transfer	#steps	Remarks
Recursive Halving/Doubling			
Ring			
Double Tree			

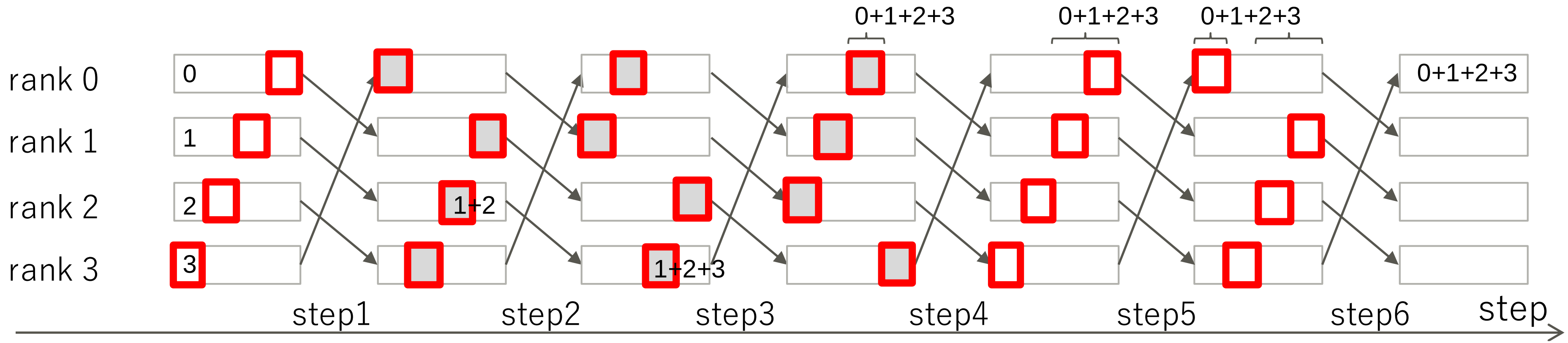
Recursive Halving/Doubling Algorithm



algorithm	Data size to transfer	#steps	Remarks
Recursive Halving/Doubling	$\frac{N}{2} \sim \frac{N}{m}$	$2 \times log_2 m$	We Implemented

※ *N*, *m* are the Data size and the number of ranks, respectively.

Ring algorithm



algorithm	Data size to transfer	#steps	Remarks
Ring	$\frac{N}{m}$	$2 \times (m-1)$	NCCL 2.3
2D – Ring	$\frac{N}{m_x}, \frac{N}{m_y}$	$2 \times (m_x + m_y - 2)$	

※ *N*, *m*, m_x, m_y are the Data size, the number of ranks, the x-ring size and y ring size, respectively

Double Tree algorithm

FUJITSU

```
/* Build a double binary tree. Take the previous tree for the first tree.
 * For the second tree, we use a mirror tree (if nranks is odd)
 *
 *                 8---------0---------5
 *          ______/ ¥______     ______/ ¥______
 *         4               12   1              9
 *       /   ¥            /      ¥           /   ¥
 *     2       6       10          3       7      10
 *    / ¥     / ¥     /  ¥        / ¥     / ¥    /  ¥
 *   1   3   5   7   9   11      2   4   6   8  11  12
 *
 * or shift it by one rank (if nranks is even)
 *
 *                 8---------0--------------9
 *          ______/ ¥                ______/ ¥
 *         4         ¥              5         ¥
 *       /   ¥        ¥           /   ¥        ¥
 *     2       6       10       3       7       11
 *    / ¥     / ¥     /  ¥     / ¥     / ¥     /  ¥
 *   1   3   5   7   9   11   2   4   6   8   10   1
 */
ncclResult_t ncclGetDtree(int nranks, int rank, int* s0, int* d0_0, int* d0_
```

source: NCCL v2.4 comments

algorithm	Data size to transfer	#steps	Remarks
Double Tree	$\frac{N}{2}$	$2 \times log_2 m$	NCCL 2.4

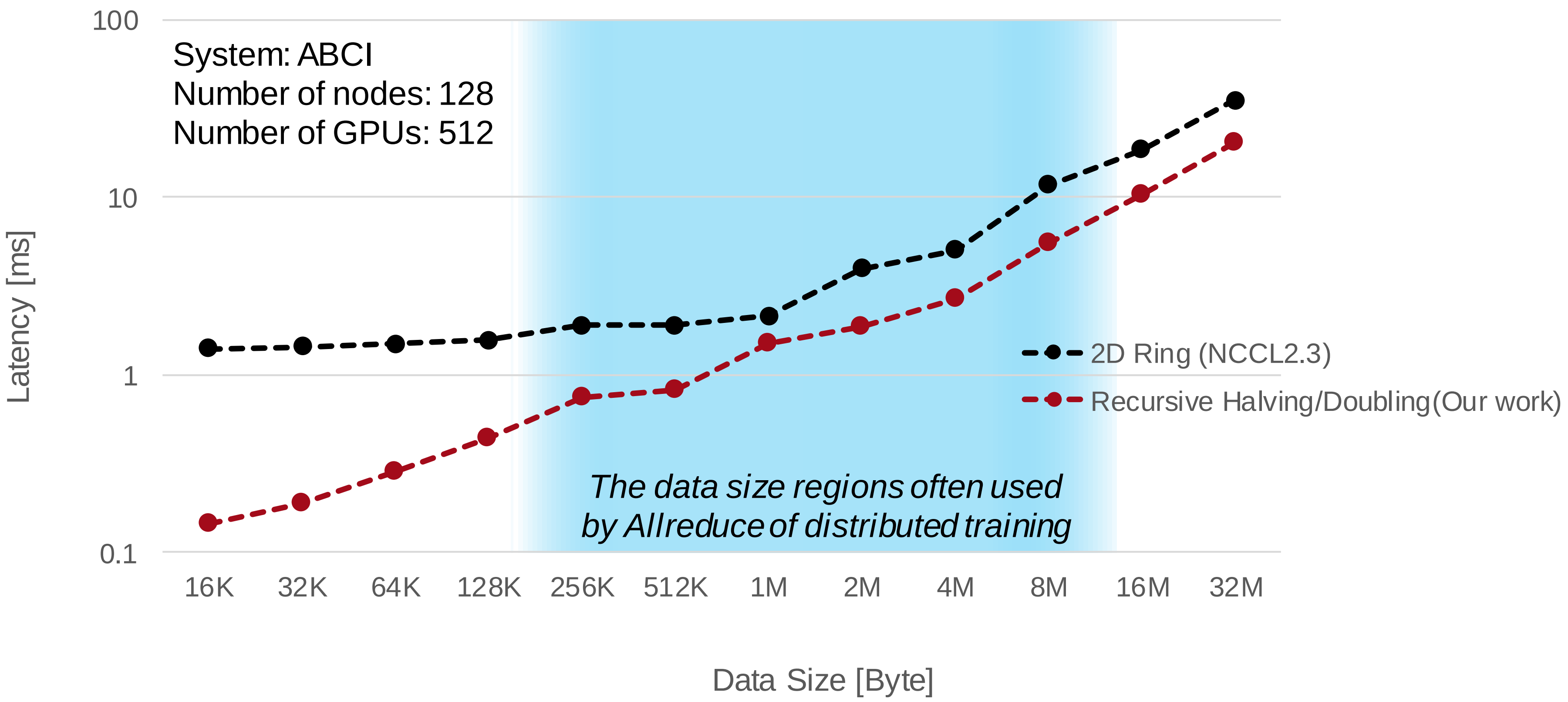
※ *N*, *m* are the Data size and the number of ranks, respectively.

Comparison of Allreduce algorithms

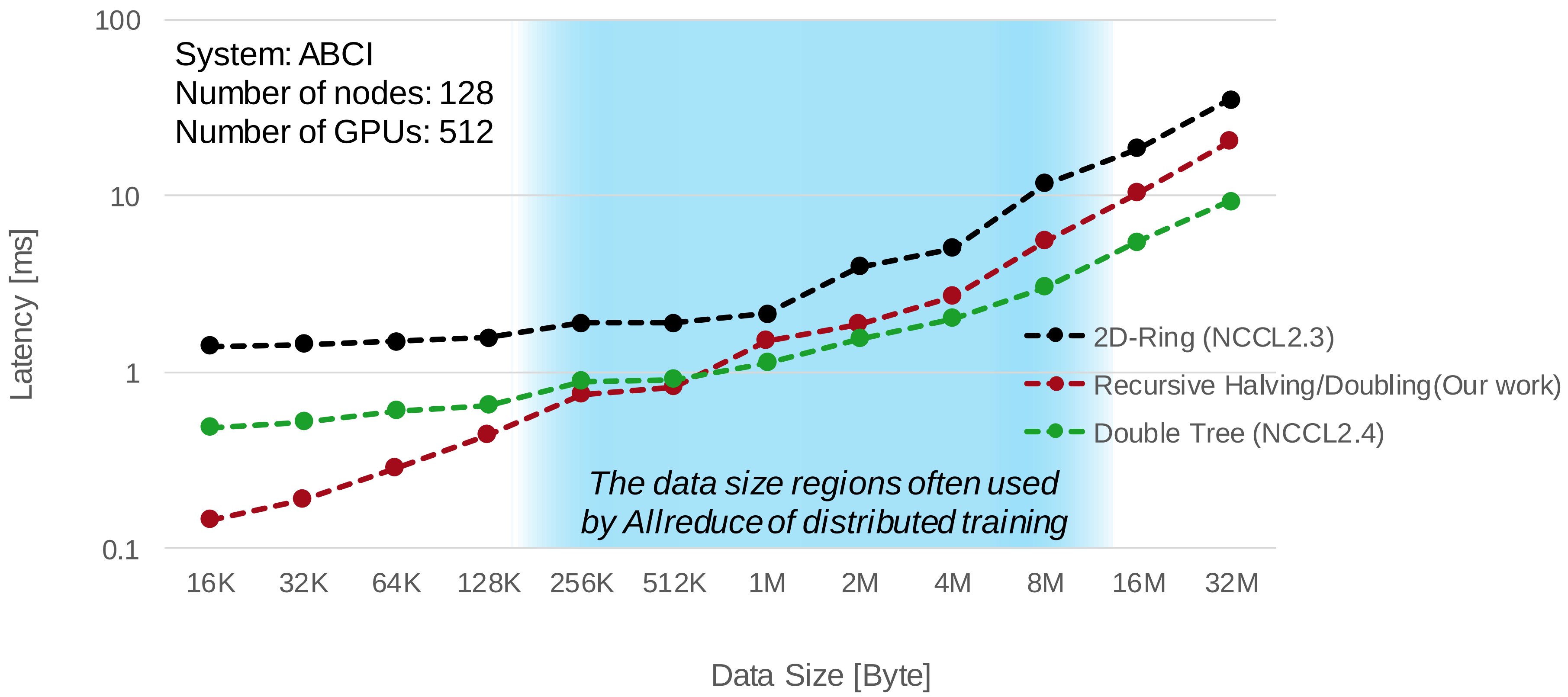
algorithm	Data size to transfer	#steps	Remarks
Recursive Halving/Doubling	$\frac{N}{2} \sim \frac{N}{m}$	$2 \times log_2 m$	We Implemented
Ring	$\frac{N}{m}$	$2 \times (m-1)$	NCCL 2.3
2D – Ring		$2 \times (m_x + m_y - 2)$	
Double Tree	$\frac{N}{2}$	$2 \times log_2 m$	NCCL 2.4

※ N, m, m_x, m_y are the Data size, the number of ranks, the x-ring size and y ring size, respectively

Evaluate the Allreduce algorithms



Evaluate the Allreduce algorithms



Optimization Points

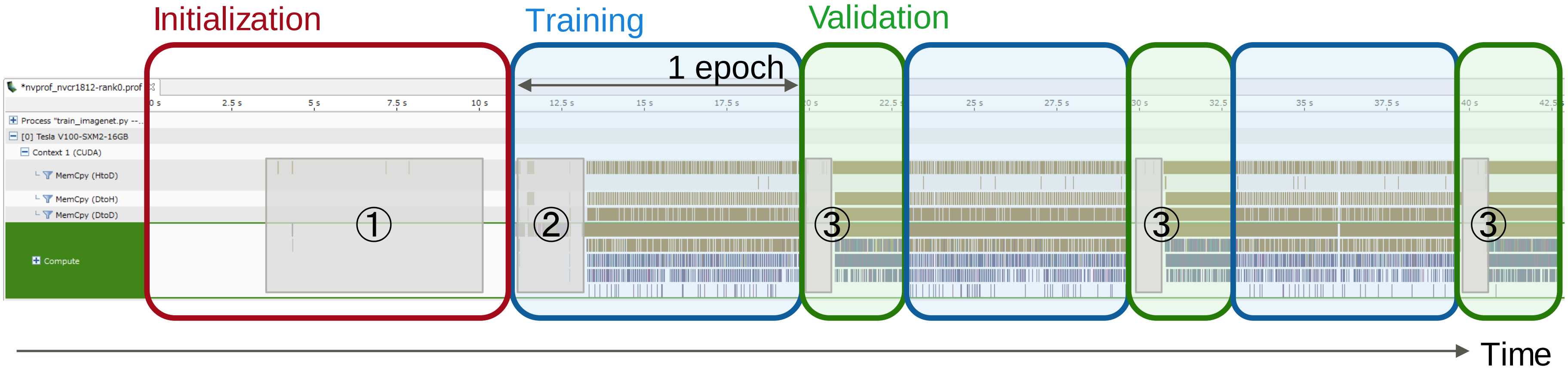
- Accelerate training speed
- Optimization of initialization and unnecessary processing
- Changes for MLPerf v0.6

Accelerate Training Speed

- Overlapping Allreduce communication with backward computation

The processing speed increased to over 1.5 M images/sec.
However we could not reach our goal for the overall training time.

Optimization of initialization and unnecessary processes



①generate common initial-weights using common random-seed in each GPU instead of broadcasting initial weights from one GPU

②Overlapping NCCL initialization with framework initialization

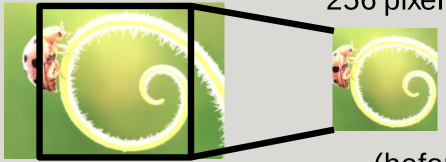
③Eliminating unnecessary processes after each epoch（0.1 ～ 0.2 sec. / epoch）

Reduce overall training time by 45 seconds (120s → 75s)

Changes when submitting MLPerf v0.6

- In MLPerf v0.6 rules, the validation accuracy was increased from 74.9% to 75.9%.

① Database for training



256 pixel

256 pixel

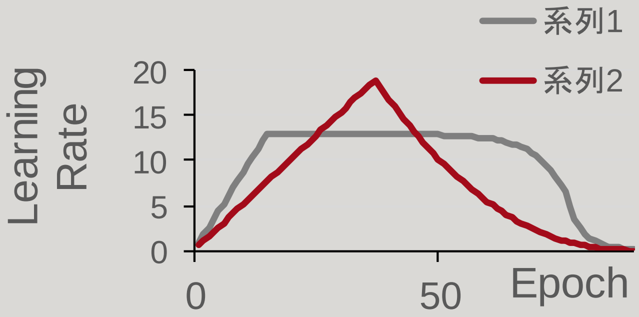
training image
(before Data augmentation)

Clopped image for April 1 result

Original image for MLPerf v0.6

② Learning Rate scheduling

We tuned LR scheduling, and changed it to follow MLPerf v0.6 rules



Evaluation and results

- Hardware
- Software
- Results
  - Validation Accuracy in training
  - Accuracy improvement
  - Scalability

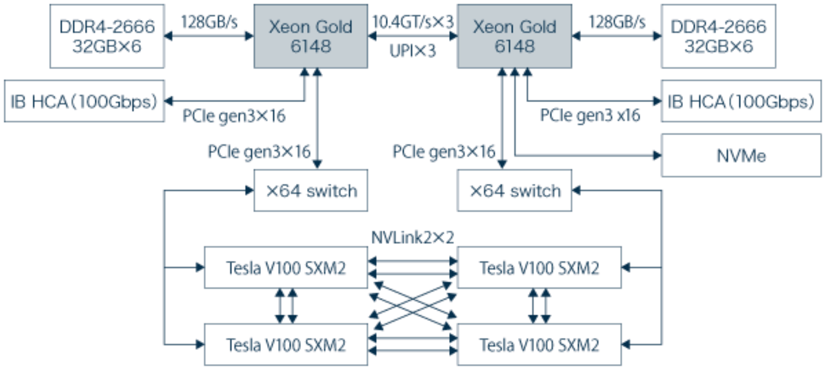
Evaluation environment

Hardware

- compute nodes
  - 4-GPUs / node
  - 2-HCA / node

- IB Interconnect
  - The intra-rack network has topology of full bi-section fat-tree
  - The inter-rack network has topology of fat-tree with 1/3 over subscription

ABCI compute node configuration



Evaluation environment

FUJITSU

Software

MXNet / Horovod

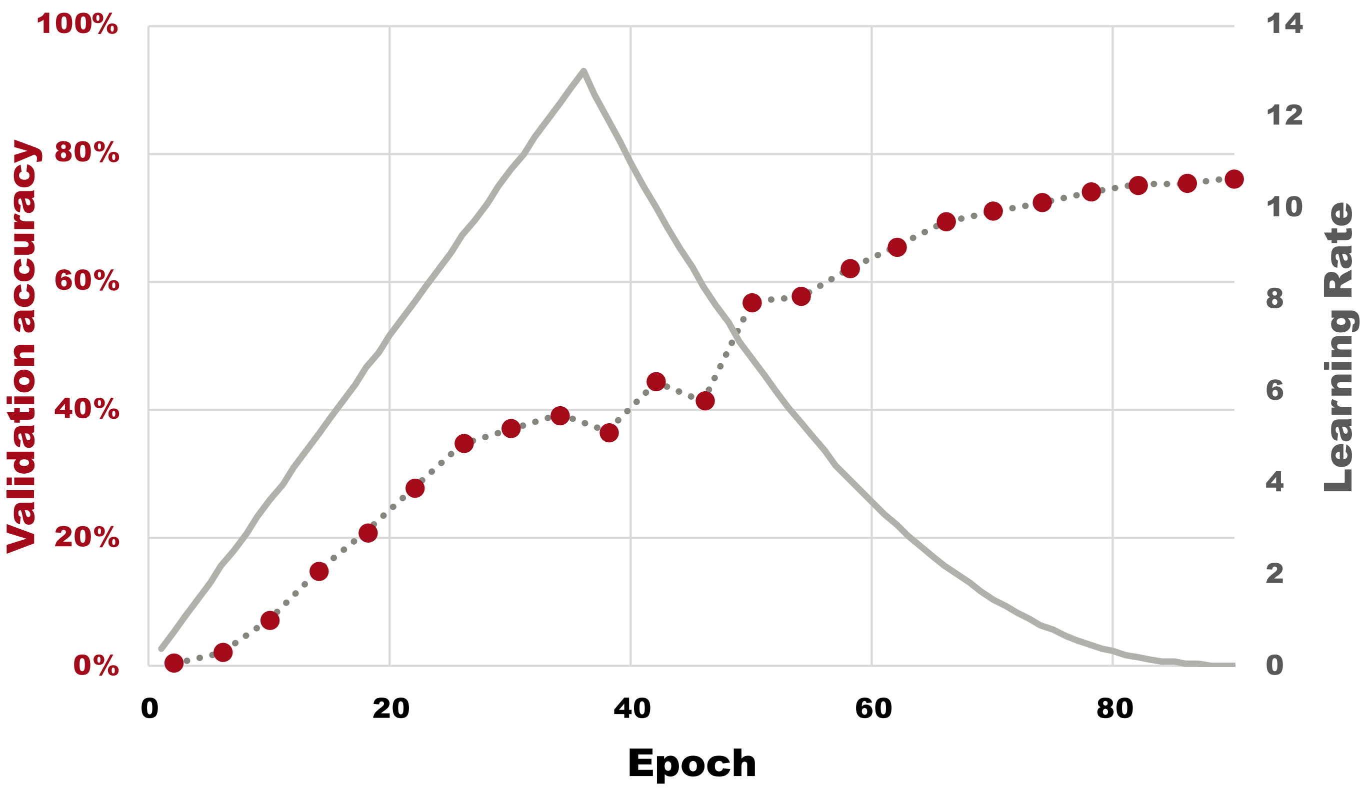
- Original source is NVIDIA tuned MXNet
- We customized and tuned

Other libralies

- CUDA, cuDNN v7.5, NCCL v2.4
- OpenMPI
- GCC 7.3
- Python 3.6

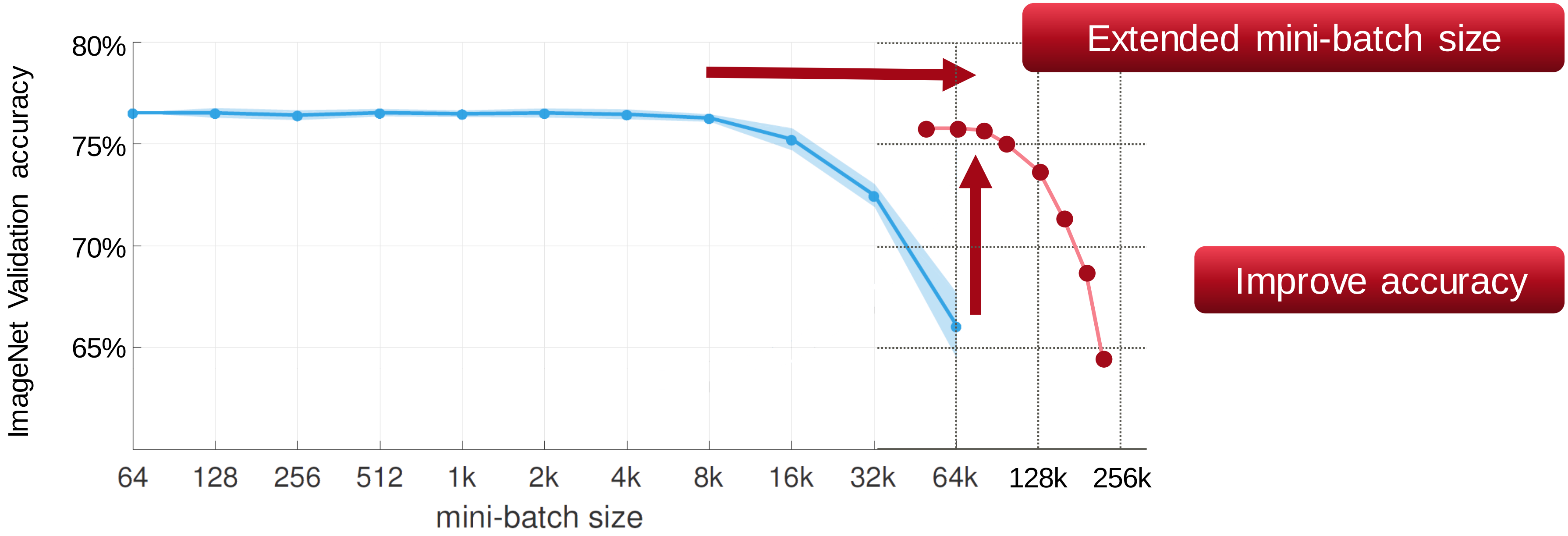
Result; Validation Accuracy in training

FUJITSU



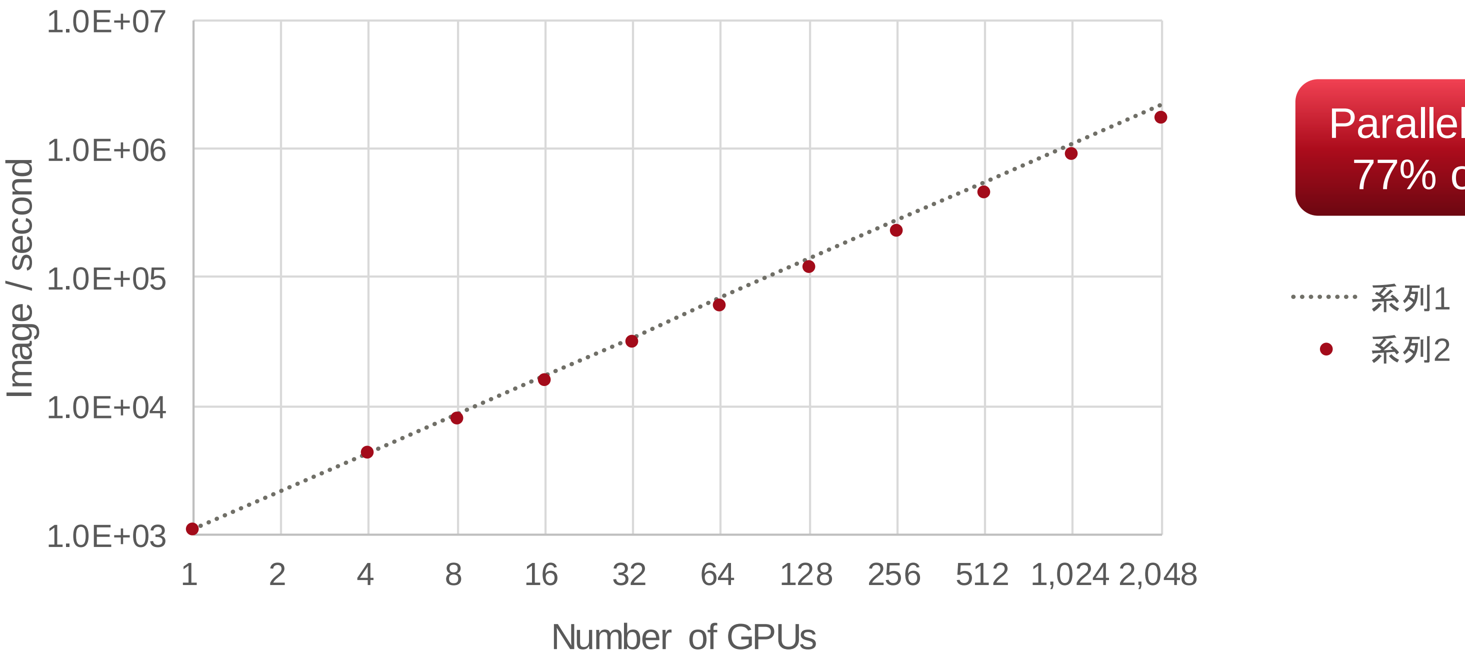
We achieved 76% accuracy with 84k mini-batch using 2k GPUs

Result; Accuracy improvement



Result; Scalability

- The number of computation images per GPU is the same (Weak scale)



Parallel efficiency is 77% on 2k GPUs

Conclusion

- We achieved 70 sec. training time (world record†) and 84K mini-batch size (world record††) of ResNet-50/ImageNet under MLPerf v0.6 rules†††
- Using ABCI 512 compute nodes (2,048 GPUs)

	Mini-batch size(max)	Processor	DL Software	Time	Validation accuracy
Facebook	8,192	Tesla P100 × 256	Caffe2	1 hour	76.3 %
Google	16,384	full TPU Pod	TensorFlow	30 min.	76.1 %
Preferred Networks	32,768	Tesla P100 × 1,024	Chainer	15 min.	74.9 %
Tencent	65,536	Tesla P40 × 2,048	TensorFlow	6.6 min.	75.8 %
Google	65,536	TPU v3 × 1,024	TensorFlow	1.8 min.	75.2 %
Sony	55,296	Tesla V100 × 3,456	NNL	2.0 min.	75.3 %
Fujitsu Labs.	86,016	Tesla V100 × 2,048	MXNet	1.17 min.	76.1 %
Huawei	?	Ascend A910 × 1,024	MindSpore	0.997 min.	>75.9%

† Our investigation in March, 2019 †† Our investigation under the conditions of SGD and fixed mini-batch size in March, 2019
††† Used closed Division rules, except for tuning six hyper parameters

shaping tomorrow with you