Performance Improvement of Deep Learning Training on Large-scale Manycore Cluster

Toshihiro Hanawa[1,2], Kohei Tamura[2*]
1 Information Technology Center, 2 Department of Electrical Engineering and Information Systems
The University of Tokyo (* Now in NTT DATA Corp.)

Overview

To shorten a large-scale training for deep learning, the distributed deep learning are widely applied to the massive clusters using accelerators such as GPUs. In contrast, manycore processor such as Intel Xeon Phi is also suitable for computing deep learning operation and it is easy to expand to large-scale cluster. In this study, to utilize deep learning training on large-scale many core cluster, we conduct performance evaluation of large-scale deep learning framework ChainerMN on Oakforest-PACS system operated by JCAHPC, and optimize the Allreduce communication latency. As a result, the improved communication of ChainerMN is 2.1x faster than the original one on Oakforest-PACS system.

ChainerMN

- **About**

ChainerMN is a scalable distributed deep learning framework developed by Preferred Networks [1]. It is an add-on package to Chainer and written in Python. Recently, ChainerMN have been merged into Chainer v5.

✓ **Scalable**
It makes full use of the latest technologies such as cuPy for GPU, MKL-DNN for CPU, and mpi4py for multi-node execution.

✓ **Flexibility**
Even dynamic neural networks can be trained in parallel.

✓ **Easy**
Minimal changes to existing code by Chainer are required.

- **Data Parallel and Model Parallel**

✓ **Data parallel**
- ➢ Divide minibatches
- ➢ Copy a model
- ➢ Average gradients

✓ **Model parallel**
- ➢ Divide a model
- ➢ Use a portion of the model
- ➢ Calculate for one minibatch

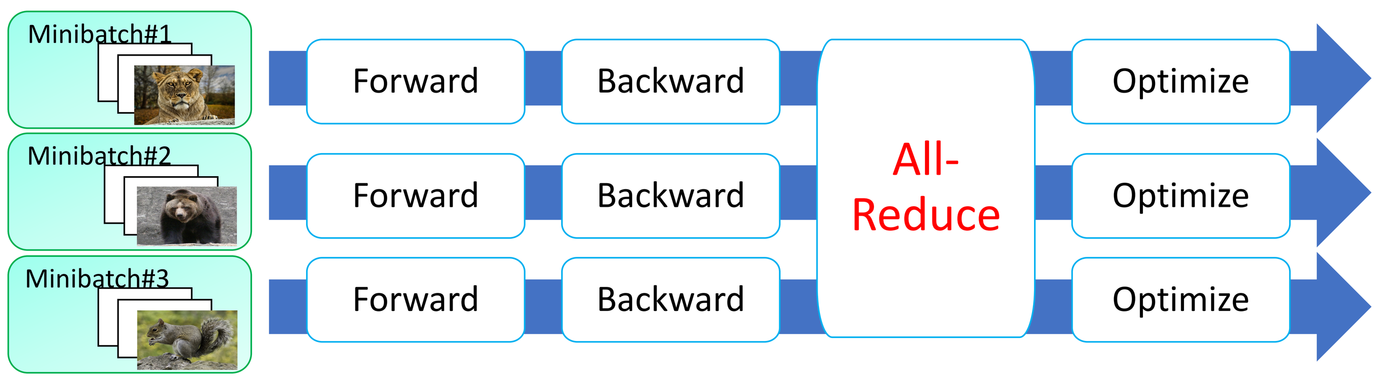


Figure 1: Process of Synchronous Data-parallel Deep Learning
The four steps of synchronous data-parallel deep learning, which is the standard method of parallelism, is illustrated. In All-Reduce step, workers communicate with each other to find the average of gradients. Each worker optimizes the model by the average of gradients.

Oakforest-PACS

TOP 500 #6, HPCG #3, Green 500 #6 @Nov. 2016
IO 500 #1 @Jun. 2018

- **Overview**

Oakforest-PACS is a supercomputer which is made up 8,208 nodes using Intel® Xeon Phi ™ 7250 processors(Code name: Knights Landing=KNL) [2, 3].

68 cores/node, 3 TFLOPS x 8,208= 25 PF

Table 1: Specification of KNL

Item		Spec
Operation Frequency		1.40 GHz
Theoretical Computation Performance		3046.4 GFLOPS
Number of Core	Physical	68
	Logical	272
Memory Capacity	MCDRAM	16 GB
	DDR4	96 GB
Memory Bandwidth	MCDRAM	490 GB/s
	DDR4	84.5 GB/s

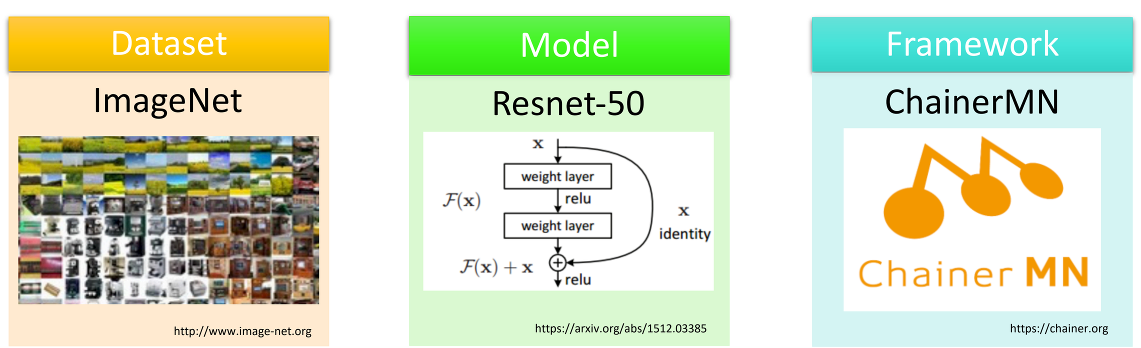
Table 2 : Specification of Oakforest-PACS

Item		Spec
Number of Node		8208
Interconnect		Intel Omnipath Architecture (100Gbps) Full-bisection BW Fat-tree
Parallel File System	Type	Lustre File System
	Storage Capacity	26 PB
	Data Transfer Rate	500 GB/sec
High-Speed File Cache	Type	DDN Infinite Memory Engine (IME)
	Capacity	940 TB
	Data Transfer Rate	1,560 GB/sec

Experiment

- **Dataset**

Imagenet is a large visual database which has over 1,400,000 pictures. Each picture is hand-annotated to indicate what objects are pictured.



Software	Version
Intel Python	3.6.3
Intel MPI	2018.1.163
MPI4py	3.0.0
Chainer	5.0.0
iDeep4py	2.0.0

- **Implementation**

From Chainer v4, iDeep(Intel Deep Learning Package) is added as a backend. iDeep enables us to parallelize threads with OpenMP and generates AVX-512 instructions for KNL automatically by JIT compiler technology.

Results

- **Loader Process and Core Affinity**

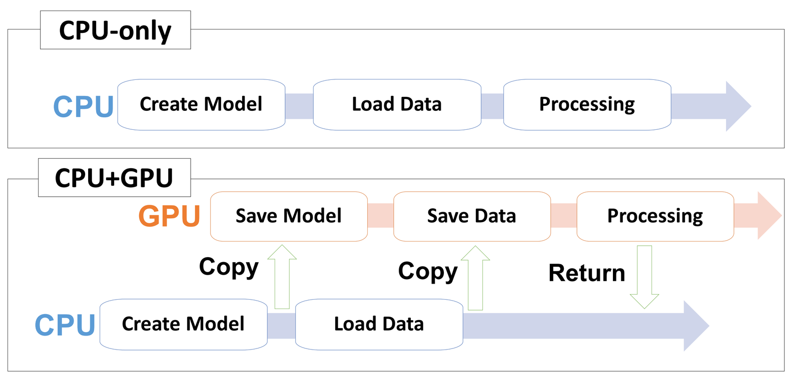


Figure 2: Execution Model in CPU-only and CPU+GPU
CPU+GPU: GPU computes asynchronously and CPU is dedicated for data load.
CPU-only: CPU have to manage both of load and compute.
=> Separate loader process from compute process

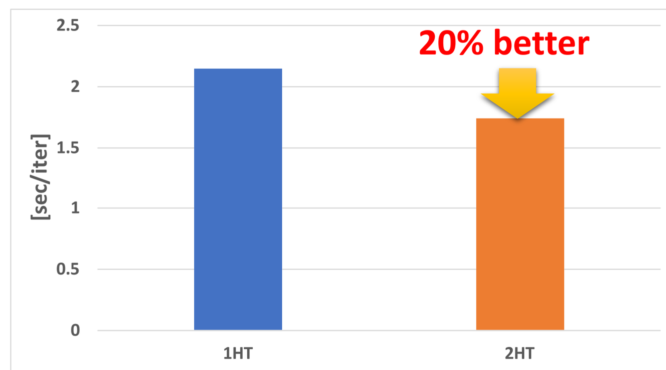


Figure 3: Affinity of Loader Process
1HT: Loader process on the same core
2HT: Loader process on the same physical core, but different logical core (HyperThreading)

- **Performance Analysis of ImageNet Training on OFP**

64 thread / process, excluding the busy core for OS services

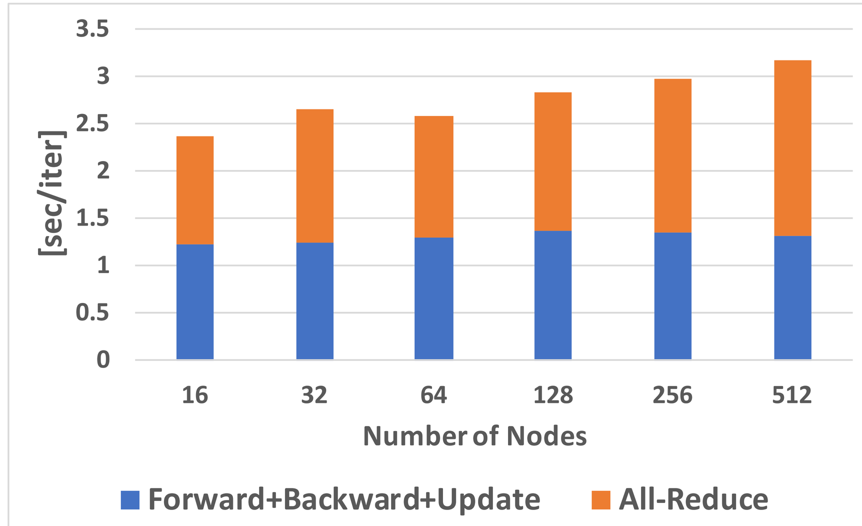
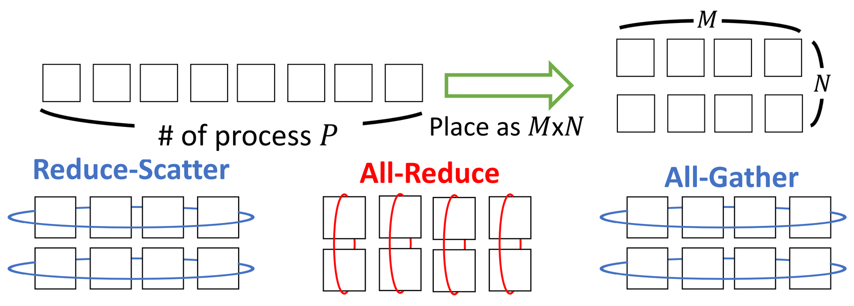


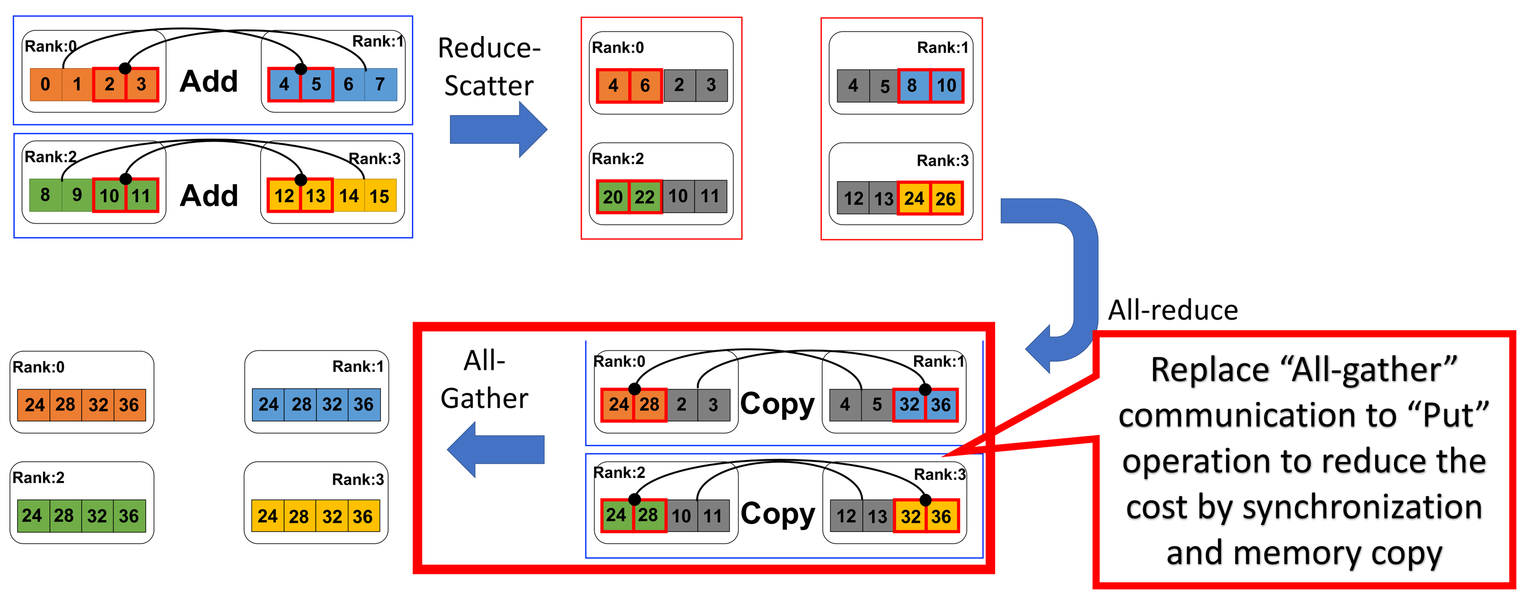
Figure 4: Break down of Execution Time per Iteration
- Weak scaling problem, well scaling for the computation.
- The communication time by All-reduce is dominant in each iteration.
  => Improvement for All-reduce is required.

- **Optimization of Communication**

✓ Huge Overhead
✓ Synchronization
✓ Memory copy
✓ Average calculation after reduction



Introduction of 2D-Torus All-Reduce [4]



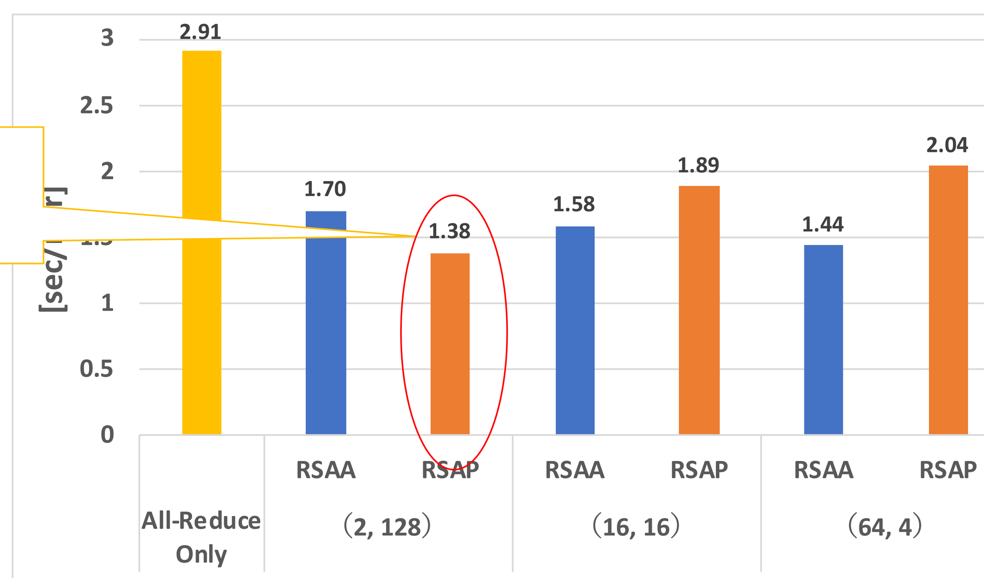


Figure 6: Elapsed Time of "All-Reduce" Process (Fig. 4) per Iteration
- All-Reduce Only：**Original pure-allreduce**
- RSAA：Reduce-Scatter, All-Reduce, All-Gather (original **2D Torus-AllReduce)**
- RSAP：Reduce-Scatter, All-Reduce, Put (**Proposed**)

References

[1] T. Akiba, K. Fukuda, and S. Suzuki, "ChainerMN: Scalable Distributed Deep Learning Framework," Proceedings of Workshop on ML Systems in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS), 2017.
[2] Joint Center for Advanced HPC, "Oakforest-PACS", http://jcahpc.jp/eng/ofp_intro.html .
[3] A. Sodani, "Knights Landing (KNL): 2nd Generation Intel® Xeon Phi™ Processor," IEEE Hot Chips 27 Symposium, 2015.
[4] H. Mikami, et al. "ImageNet/ResNet-50 Training in 224 Seconds," arXiv preprint arXiv:1811.05233, 2018.