Fugaku-LLM: A Large Language Model Trained on the Supercomputer Fugaku

February 19, 2025

Koichi Shirahata

Fujitsu Limited

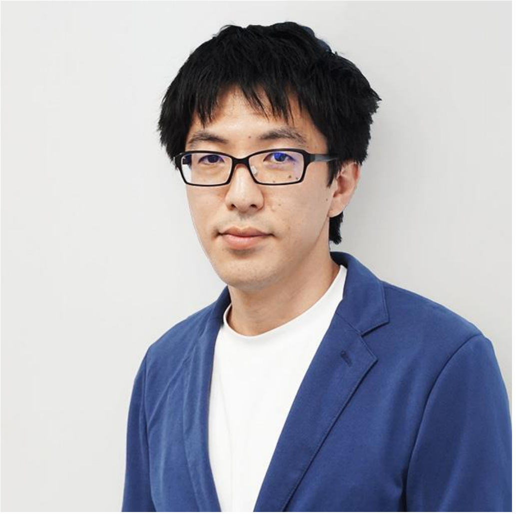
Self-introduction

FUJITSU

Koichi SHIRAHATA

Senior Project Director

Artificial Intelligence Laboratory, Fujitsu Research, Fujitsu Limited

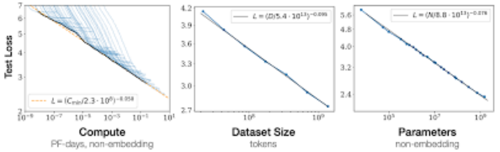


- March 2015: Received Ph.D. at School of Information Science and Engineering, Tokyo Institute of Technology
- April 2015: Joined FUJITSU LABORATORIES LTD.
- November 2020, 2021: Achieved the world's highest performance in MLPerf™ HPC, a machine learning performance benchmark, using the Fugaku supercomputer and ABCI
- May 2024: Development of a distributed parallel training method for large-scale language models in the policy response framework of the supercomputer "Fugaku"

History of generative AI over the past few years

2019/06/22 Microsoft Invests $1 billion in 110 billion OpenAI

2020/01/23 OpenAI Releases Scaling Law Paper on Language Generation Models



2021/01/05 OpenAI Announces Image + Language Model CLIP and Image Generation Model DALL-E

2022/05/23 Google Launches Imagen Image Generation Model

2022/07/22 BigScience (HuggingFace, CNRS, GENCI) Launches Multilingual Bloom

2022/08/04 Tsinghua University Announces GLM-130B

2022/08/22 Stability AI Launches Stable Diffusion Image Generation Model

2022/11/30 OpenAI Announces ChatGPT

2023/02/02 exceeded 100 million users in about 2 months after its release.

2023/02/03 Liberal Democratic Party Project Team on AI Evolution and Implementation (1st meeting)

2023/03/14 GPT-4 Now Available in ChatGPT Plus

2023/03/16 Microsoft 365 Copilot brings AI Assistant to Word, Outlook, Teams and more

2023/05/24 Initiation of the Government-Initiated Project of Supercomputer Fugaku

2023/07/18 Meta Releases LLaMA2

2023/10/03 National Institute of Informatics (NII) and ELYZA adopted the AIST generative AI development support program

2023/12/19 Tokyo Tech AIST releases Swallow-7B, 13B, 70B

2024/02/15 Google Announces Gemini Pro 1.5

2024/03/04 Anthropic Announces Claude3

2024/04/18 Meta Releases LLaMA3

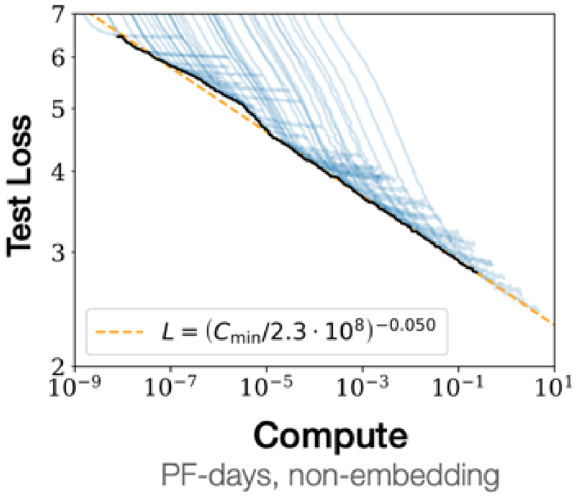
2024/05/10 Fugaku-LLM Published

Predictable cost-effectiveness (scaling law)

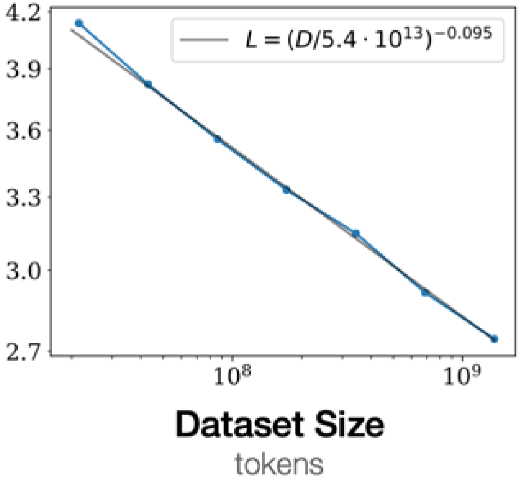
FUJITSU

- Amount of calculation ∝ number of parameters x amount of data
- Amount ∝ number of GPUs x computation time
- If we improve the quality of the data and the model, it will be cheaper.
- At least this lower cost effectiveness is guaranteed without any effort

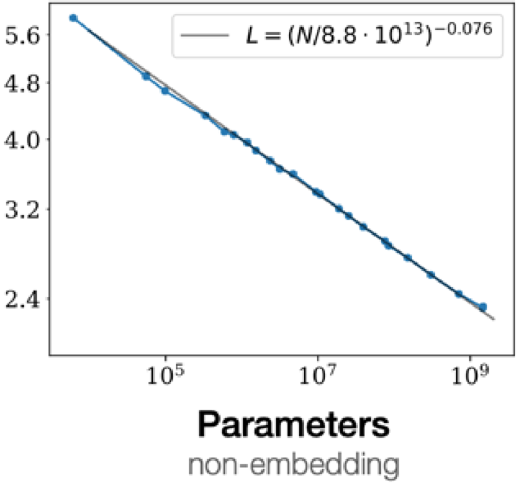
Improvement of accuracy as computation increases



Improvement of accuracy as parameters increases



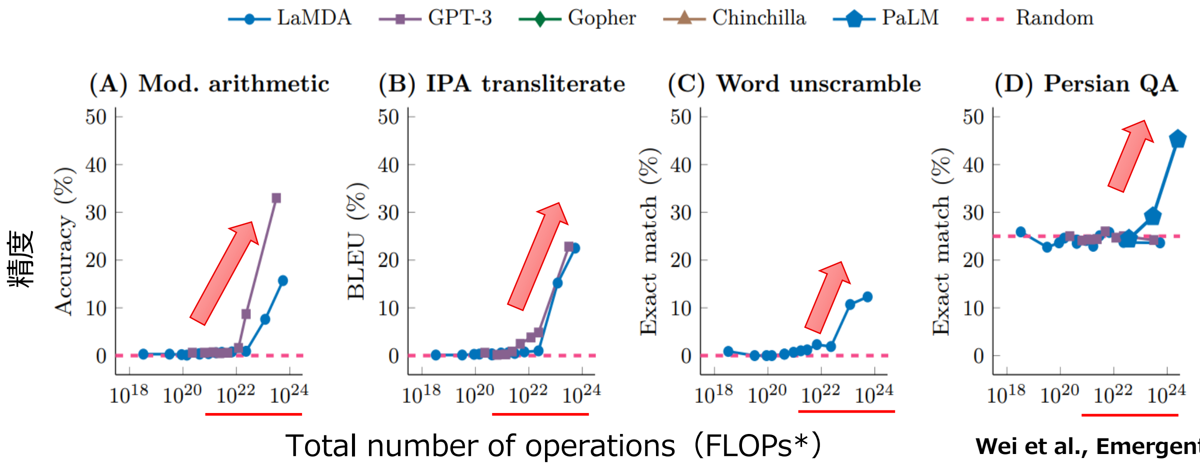
Improvement of accuracy as data size increases



Source: [Scaling Laws for Neural Language Models]

Compute resource requirements (scaling law)

- Emergence is observed when the underlying model is trained with a computational complexity of 10^{23} FLOPs*.
  - Suddenly learn to do something that one had never been able to do before
  - If we increase the amount of data and computation, they will naturally acquire various capabilities in the future.
  - Emergence is observed not only in GPT but also in other models on a similar scale.



Significantly improved accuracy at 10^{23} FLOPs*

Wei et al., Emergent Abilities of Large Language Models, 2022

* Here, FLOPs refers to the total number of operations (not speed) of pre-training.

Why do we train GPT in Fugaku?

FUJITSU

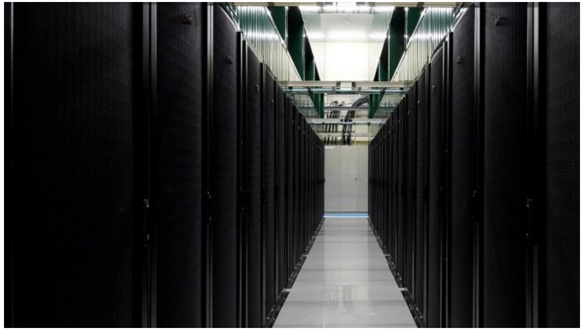
- GPUs are said to be suitable for deep learning, but the emergent figure of 10^{23} FLOPs cannot be achieved even if the V-Large class of ABCI's grand challenge system (as of 2023), one of the largest GPU supercomputers in Japan, is used.
  - 60×10^{12} FLOP/s/GPU x 4,352 GPU x 24 hours x 3,600 s = 2.2×10^{22} FLOPs
- If the effective performance of half of the theoretical peak performance of Fugaku's FP32 can be achieved, pre-training on a scale where emergency is observed (10^{23} Flop/s) using 10 million node time can be realized immediately.
  - 3.38 x 1012 FLOP/s/node x 10 million node hours x 3,600 s = 1.22×10^{23} FLOPs

10 million node hours
=10K nodes x 41.6days

The supercomputer Fugaku

CPU system (A64FX): 158,976 CPUs
Theoretical performance 1.07 ExaFlop/s (single precision)

AIST ABCI (as of 2023)



GPU systems: 4,352 GPUs
Theoretical performance 226 PFlop/s (single precision)

Fugaku supercomputer

富岳

Fugaku

Four consecutive quadruple crown

Ranked #1 in HPCG and Graph500 for 9 consecutive terms

#1 in Machine Learning Processing Benchmark MLPerf HPC in 2021

#1 in the world
- HPCG
- Graph500

- TOP500 ('20.06-'21.11)
- HPL-AI ('20.06-'21.11)

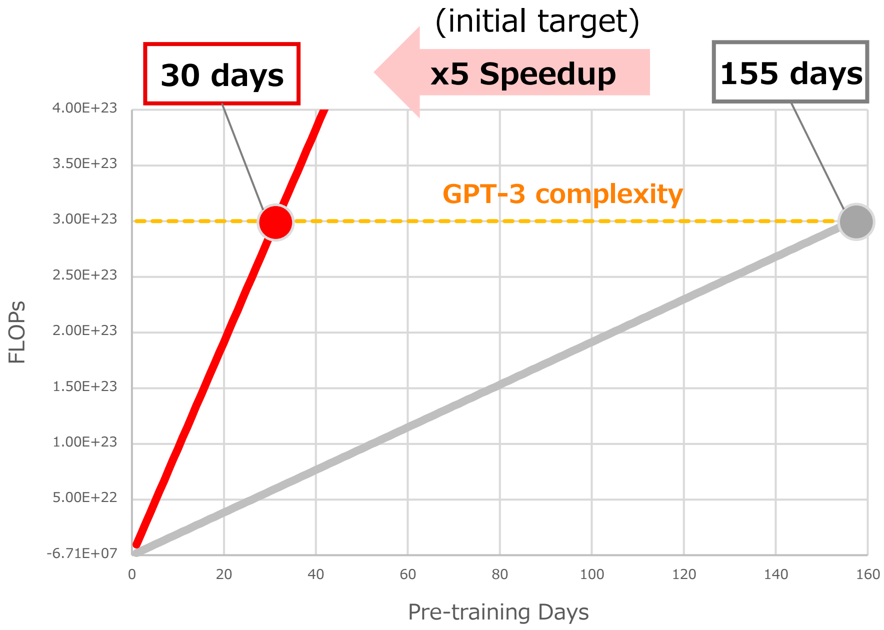
High Performance & High Efficiency
- A64FX (ARM)
- 5 PB memory
- 158,976 nodes
- 442 Petaflops *
  (Benchmark Performance)
- 30M W

The supercomputer Fugaku was jointly developed by RIKEN and Fujitsu.

photo credit: RIKEN

How do we train GPT on Fugaku?

- Pre-training the underlying model requires:
  - GPT-3: 3×10^{23} FLOPs
  - Until now, training large models such as large language models was not an intended application of Fugaku, and no optimization was made for it.
- Acceleration of GPT for Fugaku
  - Original performance: 22 PetaFLOP/s (10% efficiency)
  - Target performance: 110 PetaFLOP/s (50% efficiency)
- Acceleration strategy
  - Performance optimization of a large-scale deep training framework for Fugaku



Use approximately 1/5 (32,768 nodes) of Fugaku estimation of the case

How do we train GPT on Fugaku?

- Challenges in high-performance computing
  - Porting deep learning framework Megatron-DeepSpeed to Fugaku
  - Faster batch processing of small matrix products
  - Faster group communication over TofuD network
  - Development of a stable training method even with FP16
- Issues in natural language processing
  - Collecting and cleaning language data
  - legal review by an attorney
    - Confirming copyright, contract, and other restrictions on the release of research results (source code, model, and data), and establishing a system to legally release research results
  - Examination of methods for post-training

Roles of each organization

Tokyo Institute of Technology: Overall review, parallelization of large language models and faster communication (Combine three types of parallelization to optimize communication performance and speed up collective communication on Tofu Interconnect D)

Tohoku University: Collecting training data, selecting models to train

Fujitsu: Accelerated computation and communication (Accelerated collective communication on Tofu interconnect D, optimized pipeline parallel performance), pre-training and post-training

RIKEN: Distributed parallelization of large language models and faster communication (faster collective communication on Tofu Interconnect D)

Nagoya University: Application of Fugaku-LLM to 3D Shape Generation AI

Cyber Agents: Providing data for training

Kotoba Technologies: Porting deep learning framework to Fugaku

Performance optimization of Transformer on Fugaku

FUJITSU

- Performance analysis and optimization of each layer of the software stack to optimize Transformer performance on Fugaku
  - In particular, to **speed up dense matrix products** and to **optimize communication performance**

Transformer (GPT-x)	**Measuring Transformer performance, analyzing bottlenecks**
Parallelization (Megatron-DeepSpeed)	**Combining three types of parallelization for Fugaku communication performance optimization**
Deep Learning Framework (PyTorch)	**Uses Fujitsu's accelerated framework for Fugaku. Acceleration for LLM**
Math library	**Accleration for Transformer in dense matrix product libraries**

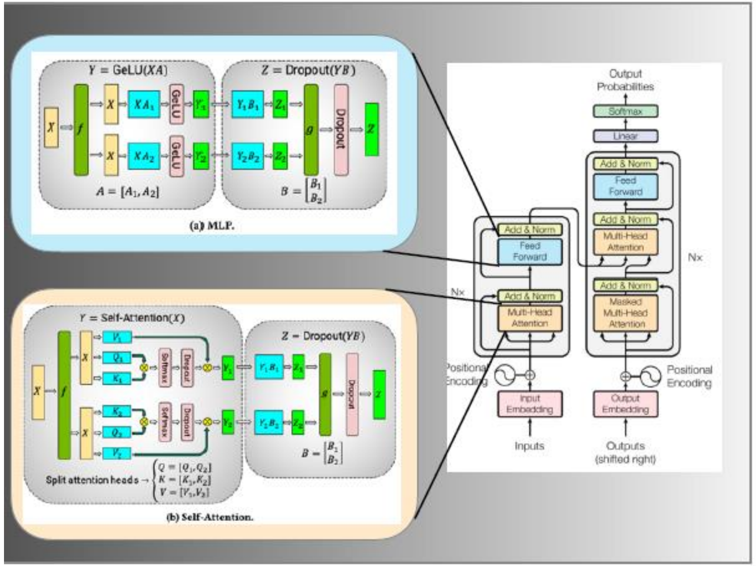
Breakdown of GPT computation time

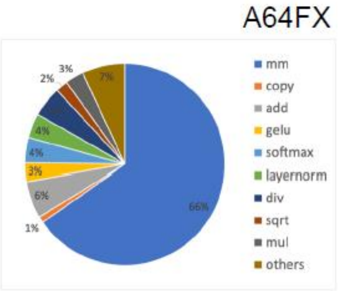
○ Much of the computation is the product of dense matrix-dense sequences

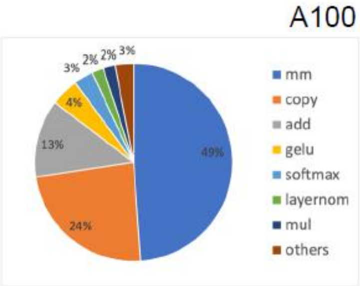
→ 66% of the time was spent on the A64FX and 49% on the A100

→ The performance was 1/3 of the theoretical peak.

By applying each transformation matrix to a common input X, we obtain $K = XW_K$, $Q = XW_K$, $V = XW_V$. Extract the degree of attention with the obtained elements.







Source: The 2nd Computational Science Forum 2022
https://hpcic-kkf.com/forum/2022/kkf_02/data/yokota_kkf2022-02_v2.pdf

Credit: Yokota Lab., Institute of Science Tokyo

Performance of dense matrix-dense matrix products

FUJITSU

A64FX

- Theoretical peak: 6.14~6.76 TFlop/s
- Measured dense matrix product: 0.66~5.86 Tflop/s (Efficiency: 10~87%)

- Performance depends on the size of the matrix.
- Fast implementation from the framework is needed
- It is also necessary to check whether it can be called multiple times

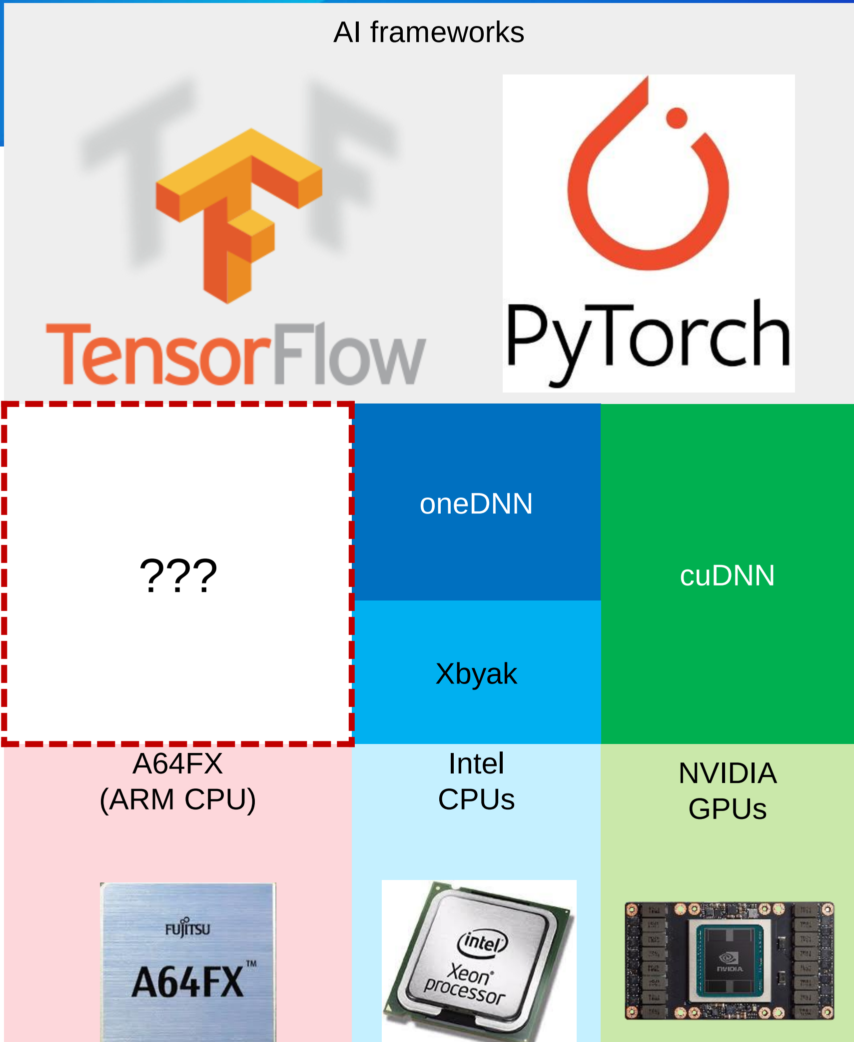
```
T=(0,0) M|N|K=(1152,2048,6912) ldA|B|C=(1152,6912,1152)
Avg gflop/s: 4925.0345834969618055555 and abs time: 15.2596031350000000000
T=(0,0) M|N|K=(4608,2048,6912) ldA|B|C=(4608,6912,4608)
Avg gflop/s: 5747.4801561723090277777 and abs time: 52.2987149500000000000
T=(0,0) M|N|K=(6912,2048,3456) ldA|B|C=(6912,3456,6912)
Avg gflop/s: 5855.0114315169270833333 and abs time: 38.5031557600000000000
T=(0,0) M|N|K=(6912,2048,4608) ldA|B|C=(6912,4608,6912)
Avg gflop/s: 5800.3770888750000000000 and abs time: 51.8221436100000000000
T=(0,0) M|N|K=(96,2048,2048) ldA|B|C=(3456,2048,96)
Avg gflop/s: 655.86067345153356481481 and abs time: 9.57510412620000000000
```

Credit: RIKEN

Language		PyTorch1.13	benchdnn latest	benchdnn gpt_fugaku	C++	Py
CPU		A64FX	A64FX	A64FX	A64FX	Xe @
Data type		fp32	fp32	fp32	fp32	fp3
m		384	384	384	384	
n		64	64	64	64	
k		384	384	384	384	
# of heads		16	16	16	16	
# of batch		1	1	1	1	
# of operations		301,989,888	301,989,888	301,989,888	301,989,888	
CPU specification	# of operation units	2	2	2	2	
	# of ops. (add + mul)	2	2	2	2	
	Clock frequency	2,000,000,000	2,000,000,000	2,000,000,000	2,000,000,000	
	# of SIMD lanes	16	16	16	16	
	# of cores	48	48	48	48	
Theoretical performance[TFLOPS]	1 core	0.13	0.13	0.13	0.13	
	12 cores	1.54	1.54	1.54	1.54	
	24 cores	3.07	3.07	3.07	3.07	
	48 cores	6.14	6.14	6.14	6.14	
Processing time if theoretical performance is achieved[ms]	1 core	2.359	2.359	2.359	2.359	
	12 cores	0.197	0.197	0.197	0.197	
	24 cores	0.098	0.098	0.098	0.098	
	48 cores	0.049	0.049	0.049	0.049	
Measured processing time[ms]	1 core	4.313	39.711	4.426	4.031	
	12 cores	0.672	4.207	0.392	0.478	
	24 cores	0.483	2.113	0.231	0.450	
	48 cores	0.499	1.095	0.227	0.480	
Achieved performance[%]	1 core	54.7%	5.9%	53.3%	58.5%	
	12 cores	29.2%	4.7%	50.2%	41.1%	
	24 cores	20.4%	4.7%	42.5%	21.8%	
	48 cores	9.8%	4.5%	21.6%	10.2%	
Achieved performance[TFLOPS]	1 core	0.070	0.008	0.068	0.075	
	12 cores	0.449	0.072	0.771	0.632	
	24 cores	0.625	0.143	1.305	0.671	
	48 cores	0.605	0.276	1.329	0.629	
Note		SSL2	oneDNN latest	oneDNN latest	SSL2	one
		cblas_sgemm?	?	JIT	cblas_sgemm	cbl
		iterate 10*10 times	iterate 5 seconds	iterate 5 seconds	iterate 100 times	iter

Deep learning software stack (before)

- AI frameworks work in a variety of environments
  - Popular AI frameworks: TensorFlow, PyTorch
  - Architectures: x86_64 CPU, ARM CPU, NVIDIA GPU, AMD GPU
- Optimized DNN libraries are essential for fast AI processing
  - NVIDIA: cuDNN, Intel: oneDNN, ...
- There was no DNN library for ARM
  - In particular, there was no library to efficiently execute ARM's SVE (SIMD) instructions.



Deep learning software stack (our development)

- **ARM extension of oneDNN**
  - We added features to oneDNN for ARM CPUs
  - Highly efficient layer processing using SVE instruction
- **Development of Xbyak_aarch64**
  - OneDNN uses Xbyak intenally
    - Xbyak is a C++ library for writing assembly code
    - It can dynamically generate machine instructions
      - DL code often has different parameters
      - It can generate efficient instruction sequences using parameters known only at runtime
  - We developed aarch64 version of Xbyak
    - Dynamic function generation with Xbyak_aarch64
  - We also developed translator for automatically porting OneDNN functions for x86_64 to funtions for aarch64

For more information, please refer to Fujitsu Research technology blog.
https://blog.fltech.dev/entry/2020/11/18/fugaku-onednn-deep-dive-ja

(Reference) Performance evaluation on ResNet-50

FUJITSU

- We accelerated oneDNN for ARM delivers 9.2 times faster

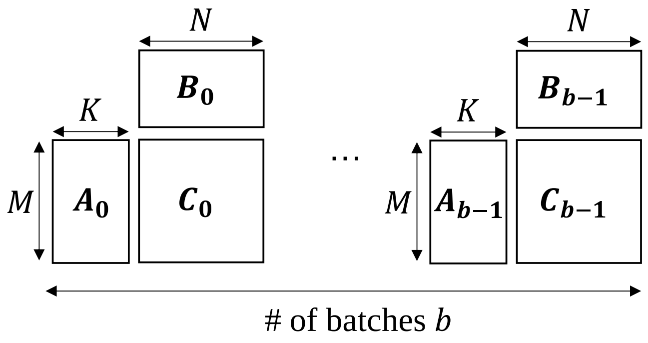
For more information, please refer to Fujitsu Research technology blog.
https://blog.fltech.dev/entry/2020/11/18/fugaku-onednn-deep-dive-ja

Implementation of Batch Matrix Multiplication

FUJITSU

- Large Language Models (LLMs) represent a significant leap
- LLM training is also carried out on the supercomputer Fugaku.
- LLM process is dominated by matrix multiplications
  - 2 types of matrix multiplication
    - A large size matrix multiplication
    - Batch Matrix Multiplication (BMM): performing multiple matrix multiplications

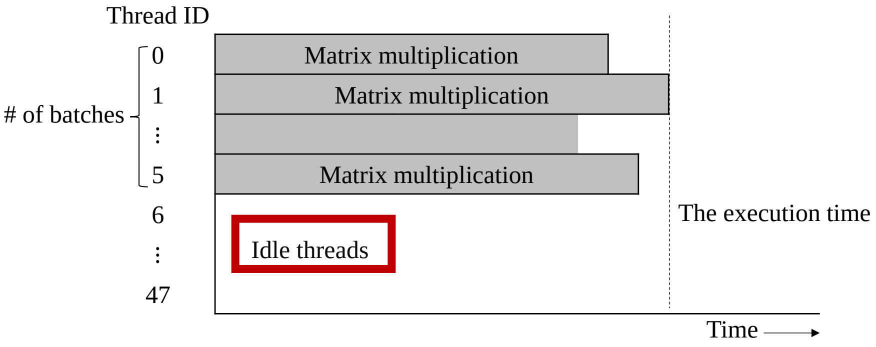
We propose **an efficient BMM implementation for A64FX CPUs.**



Implementation of Batch Matrix Multiplication for Large Language Model Training on A64FX CPUs, Hiroki Tokura et al., COOL Chips 27

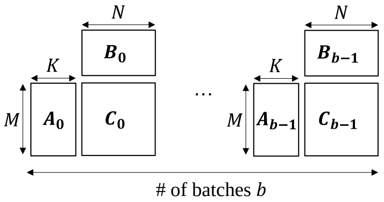
PyTorch original implementation

- PyTorch uses BLAS routines to accelerate matrix multiplications in LLMs
- A matrix multiplication is assigned per thread
  - The performance is degraded if the number of matrix multiplications is less than the number of cores



Batch matrix multiplication patterns appeared in the LLM training of our evaluation.

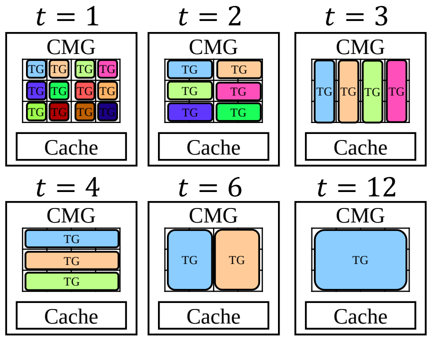
Pattern	1	2	3	4	5
Transpose of A	No	No	No	Yes	Yes
Transpose of B	No	Yes	Yes	No	No
# of matrix multiplications	6	6	6	6	6
M	144	144	2048	2048	2048
N	2048	2048	144	2048	2048
K	2048	2048	2048	144	144
LDA	2592	864	2048	2592	2592
LDB	2048	2048	2592	2592	864
LDC	144	144	2048	2048	2048



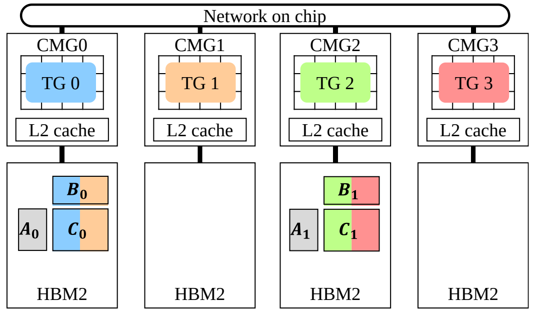
Implementation of Batch Matrix Multiplication for Large Language Model Training on A64FX CPUs, Hiroki Tokura et al., COOL Chips 27

Our proposed implementation

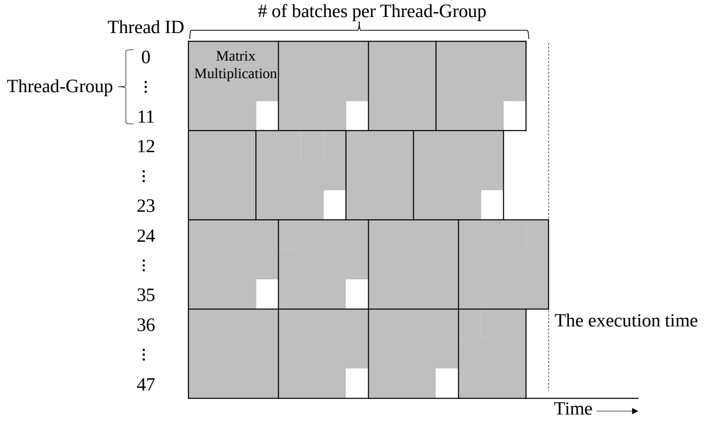
- A64FX CPU has 48 cores
  - 48 Threads are divided into Thread-Groups(TGs) every t threads (t is the number of threads in a TG)
- A matrix multiplication is computed by g TGs (g is the number of TGs to be computed in parallel)
  - We experimentally determine the optimal values of t and g



The patterns of Thread-Groups



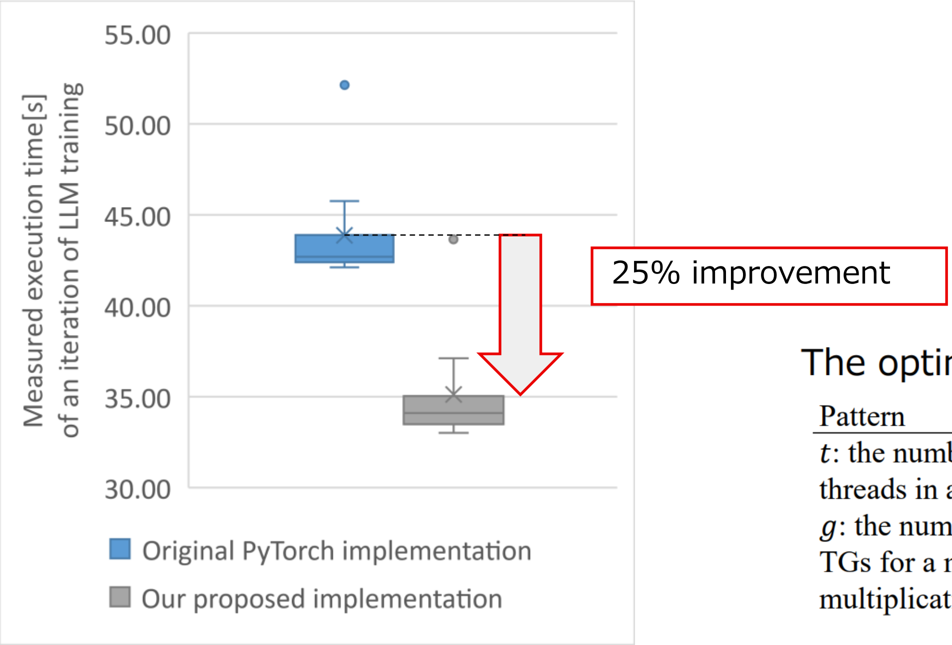
Example of TG assignment for $g = 2$



Implementation of Batch Matrix Multiplication for Large Language Model Training on A64FX CPUs, Hiroki Tokura et al., COOL Chips 27

Evaluation result of overall LLM training

Our proposed implementation contributes to a 25% improvement in overall LLM training



CPU: A64FX(48cores, 2.2GHz)
The language environment: tcsds-1.2.38

Fujitsu Processor A64FX Specifications

Cores	48
Frequency[GHz]	2.2
FP32 Peak Flops[TFLOPS]	6.7584

The optimal values of t and g of each pattern.

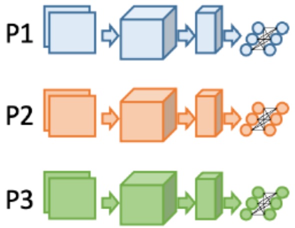
Pattern	1	2	3	4	5
t: the number of threads in a TG	12	4	4	4	4
g: the number of TGs for a matrix multiplication	2	2	2	2	2

Implementation of Batch Matrix Multiplication for Large Language Model Training on A64FX CPUs, Hiroki Tokura et al., COOL Chips 27

Parallel training method of GPT

- In order to train GPT efficiently in Fugaku, it is important to properly combine three types of parallelization.

Data parallel



Data is distributed

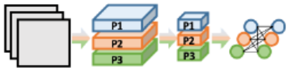
Model is redundant

Gradient collective communication

Challenge: Generalization performance decreases as batch size increases

Solution: Regularization and optimization techniques

Tensor parallel



Data is redundant

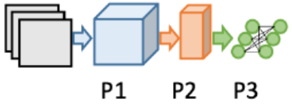
Model is distributed

Activation collective communication

Challenge: Increased communication frequency

Solution: Apply algorithms to reduce communication, such as SUMMA

Pipeline parallel



Data is redundant

Model is distributed

Activation one-to-one communication

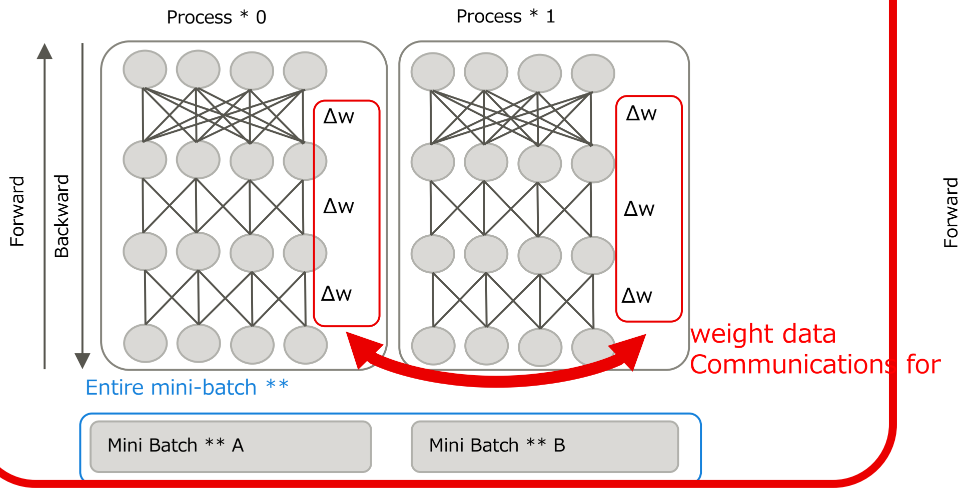
Challenge: Pipeline bubble

Solution: Apply bidirectional pipeline

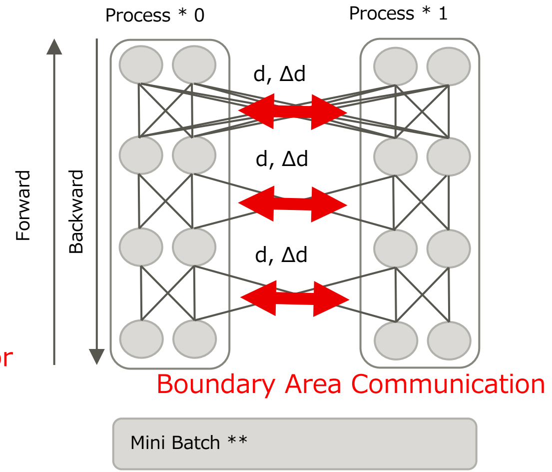
Data parallel and tensor parallel

We use data parallelism first

- Data Parallel: Compute multiple data in parallel
  - **Advantage: Less sensitive to communication time**
  - **Cons: Too much batch size slows down training**



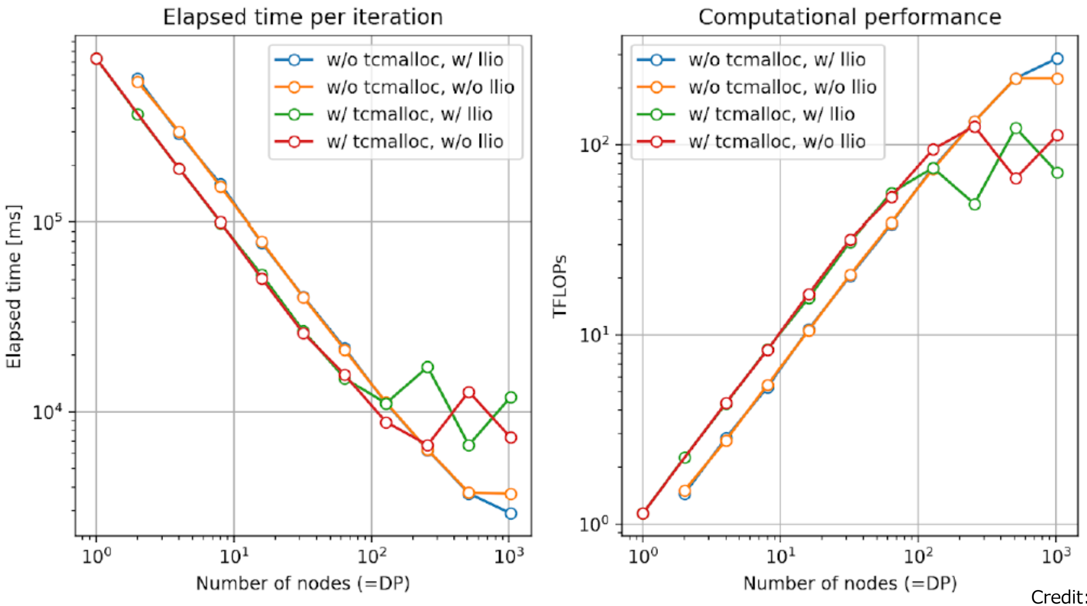
- Tensor Parallel: Split NN and compute in parallel
  - **Pros: No accuracy loss**
  - **Cons: Susceptible to communication time**



*Process: Unit of processing assigned to a processor (multiple processes can be assigned to a processor)
** Mini-batch: Number of samples of input data (images, etc.) to be processed at one time

Data parallel scalability

sequence-length=1024
per-cpu-batchsize=1, global-batch-size=1024
gradient-accumulation-steps=1024/#nodes
#parameters=**124M**



Credit: RIKEN

DP: Number of nodes in data parallel

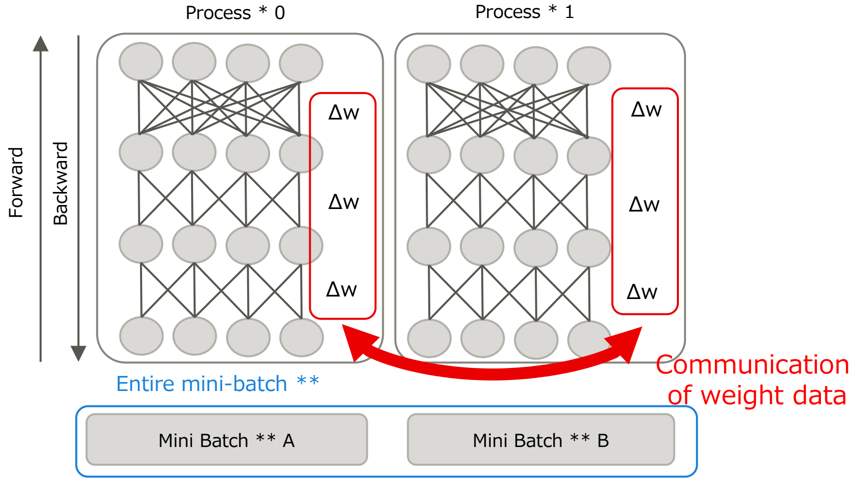
The 2nd Computational Science Forum 2022
https://hpcic-kkf.com/forum/2022/kkf_02/data/yokota_kkf2022-02_v2.pdf

Data parallel and tensor parallel

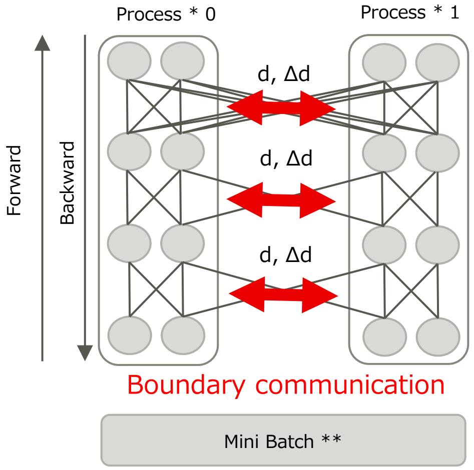
FUJITSU

We use tensor parallel together with data parallel

- Data Parallel: Compute multiple data in parallel
  - **Advantage: Less sensitive to communication time**
  - **Cons: Too much batch size slows down training**

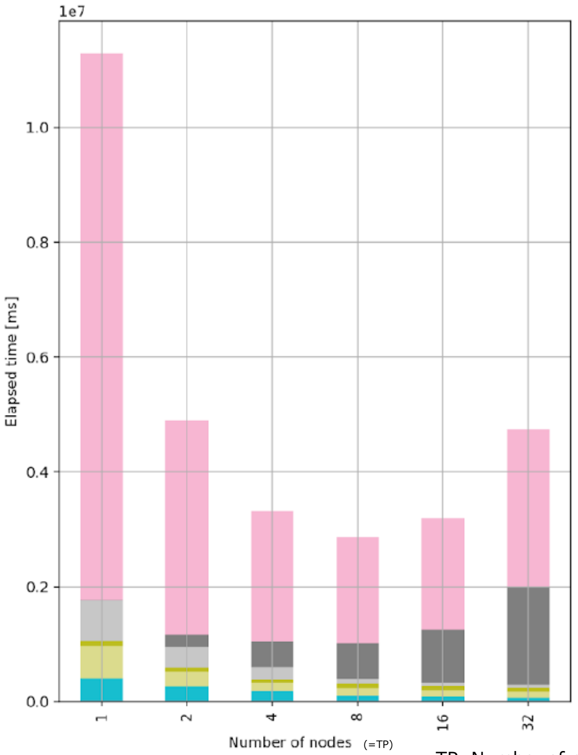


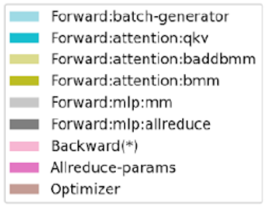
- Tensor Parallel: Split NN and compute in parallel
  - **Pros: No training loss**
  - **Cons: Susceptible to communication time**



*Process: Unit of processing assigned to a processor (multiple processes can be assigned to a processor)
** Mini-batch: Number of samples of input data (images, etc.) to be processed at one time

Tensor parallel scalability (execution time breakdown)





Backward time scales until 8 nodes
Forward time scales well
AllReduce time increases rapidly

TP: Number of nodes in tensor parallel

Credit:Yokota Lab. Institute of Science Tokyo

The 2nd Computational Science Forum 2022
https://hpcic-kkf.com/forum/2022/kkf_02/data/yokota_kkf2022-02_v2.pdf

Performance of data parallel and tensor parallel combinations

FUJITSU

sequence-length=1024
per-cpu-batchsize=1, global-batch-size=1024
gradient-accumulation-steps=1024/#DP
#parameters=**1.3B**

DP: Number of nodes in data parallel
TP: Number of nodes in tensor parallel

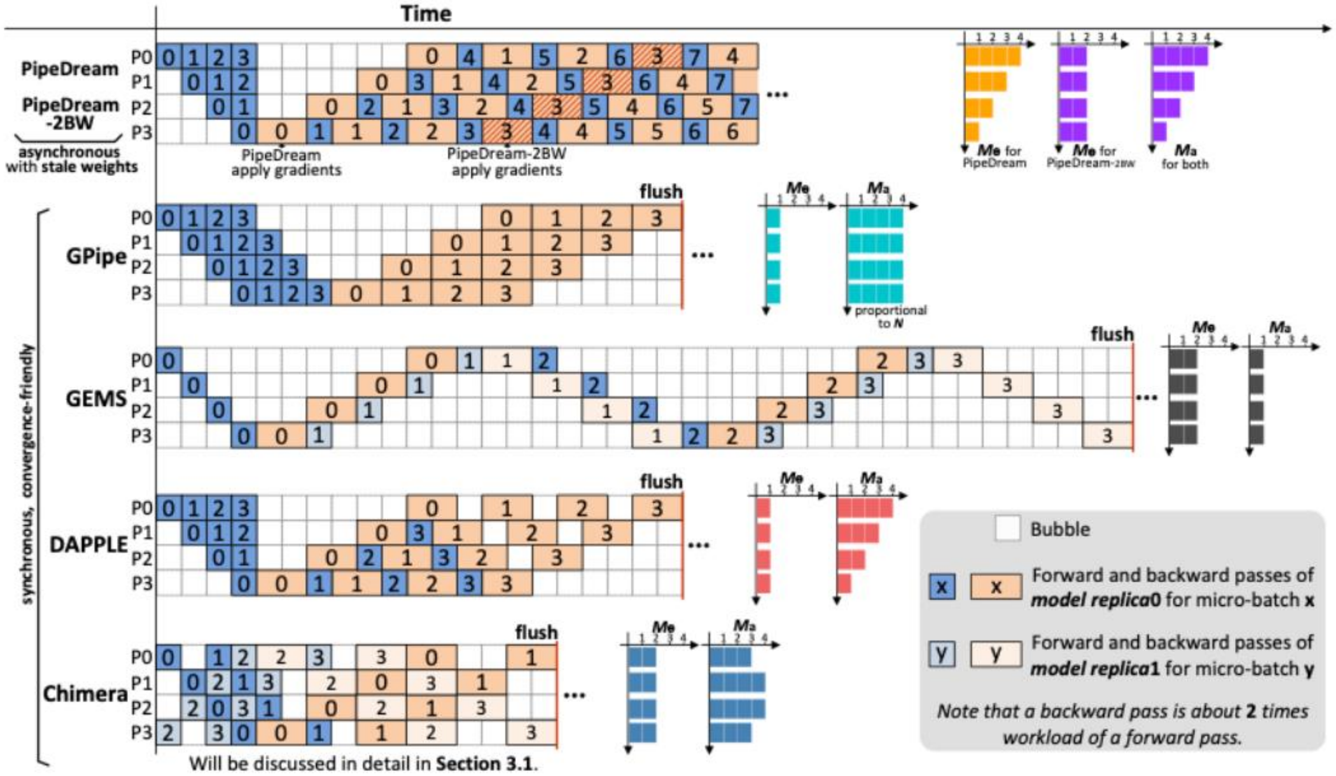
# CPUs	# DP	# TP	Achieved teraFLOPs per CPU	Percentage of Theoretical Peak FLOPS	Aggregated petaFLOPs per System	Equivalence to # of A100s (compared to 1.7B set-up)
1	1	1	0.99	16%	0.001	0.01
4	1	4	0.86	14%	0.003	0.02
64	16	4	0.84	14%	0.053	0.38
256	64	4	0.79	13%	0.198	1.44
1024	256	4	0.59	10%	0.590	4.31
2048	512	4	0.49	8%	0.980	7.15
4096	1024	4	0.41	**7%**	**1.640**	**11.97**

We are only getting around 10% of the theoretical peak of A64fx at the moment

Supplied:
Dear Hiroyuki Kojima, Kotoba Technologies
Mr. Kazuto Ando, RIKEN

The 2nd Computational Science Forum 2022
https://hpcic-kkf.com/forum/2022/kkf_02/data/yokota_kkf2022-02_v2.pdf

Pipeline parallel



Chimera: Efficiently Training Large-Scale Neural Networks with Bidirectional Pipelines, https://arxiv.org/abs/2107.06925

Pipeline parallel

- We apply interleaved 1F1B pipeline parallel implemented in Megatron-LM

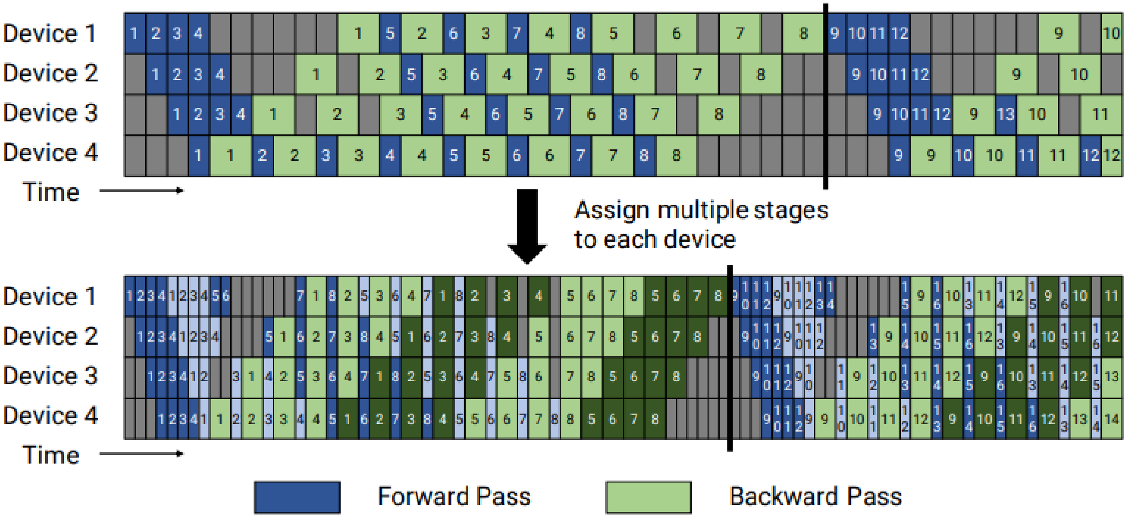
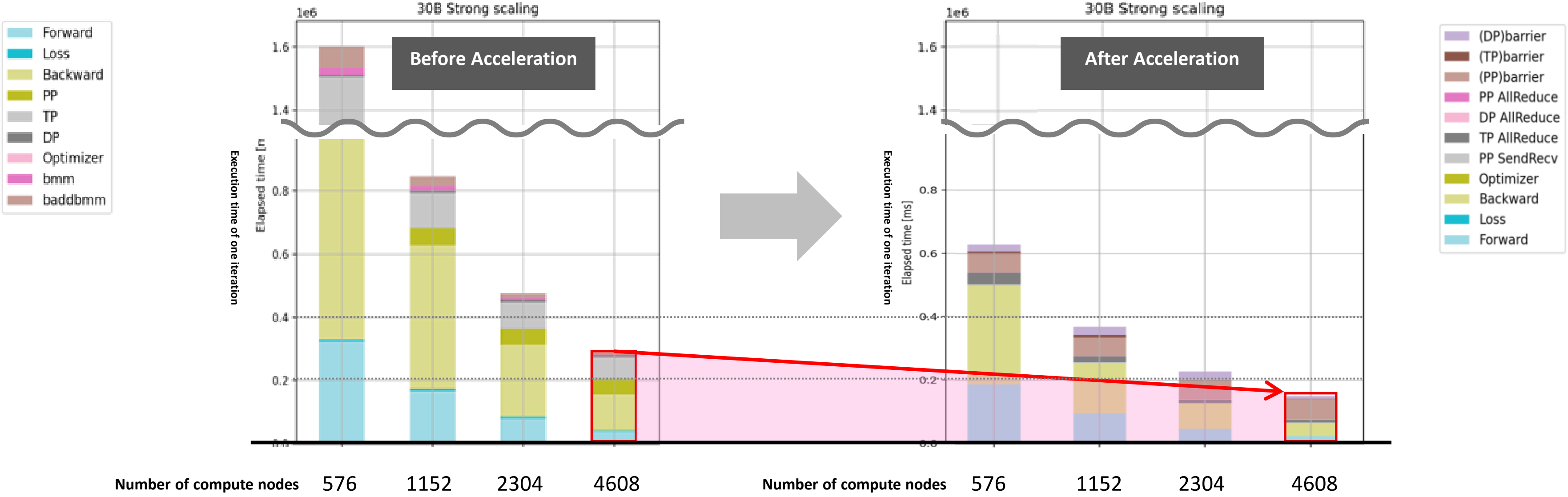


Figure 4: Default and interleaved 1F1B pipeline schedules. The top figure shows the default non-interleaved 1F1B schedule. The bottom figure shows the interleaved 1F1B schedule, where each device is assigned multiple chunks (in this case, 2). Dark colors show the first chunk and light colors show the second chunk. The size of the pipeline bubble is smaller (the pipeline flush happens sooner in the interleaved timeline).

https://arxiv.org/pdf/2104.04473.pdf

Computational performance of training with 30B parameters

- Increased computational efficiency from 10% at the beginning of development to about 20%

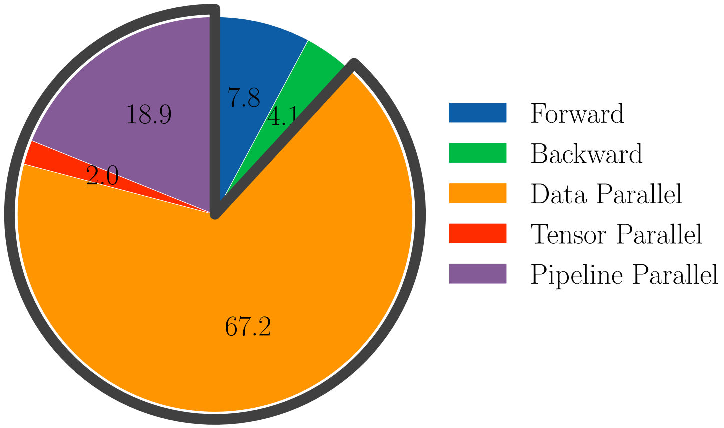


Credit: Yokota Lab., Institute of Science Tokyo

DP: Number of nodes for data parallel
TP: Number of nodes for tensor parallel
PP: Number of nodes for pipeline parallel

Percentage of time in training language models

- About 90% of the time is related to communications
  - Reducing the percentage of communication time leads to faster speeds
- Tensor Parallel and Data Parallel incurs Allreduce communications
- Pipeline Parallel:
  - Adjacency communication with Send/Recv
  - Include wait time (bubble)



Percentage of time in GPT-13B training on Fugaku
TP=6, PP=8, DP=64

DP: Number of nodes for data parallel
TP: Number of nodes for tensor parallel
PP: Number of nodes for pipeline parallel

Accelerating All-reduce Communication in Large-Scale Machine Learning on Fugaku, Nakamura et al., IPSJ-HPC-193

Accelerating All-reduce Communication in Large-Scale Machine Learning on Fugaku

FUJITSU

❶ Accelerating AllReduce on Fugaku

- Bidirectional Ring-AllReduce
- 6D Mesh/Torus rankmap
- Accelerate iterated calculations
- Computation time hiding

❷Accelerating training of language models with the accelerated Allreduce

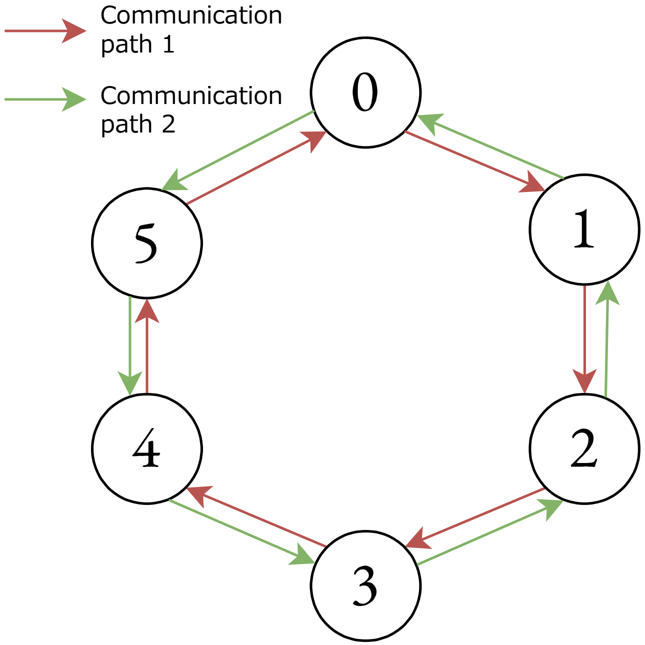
- PyTorch integration
- Rankmap for 3D parallelism
- Accelerating for large message size

Accelerating All-reduce Communication in Large-Scale Machine Learning on Fugaku, Nakamura et al., IPSJ-HPC-193

Proposed Method: Bidirectional One-Dimensional Ring-AllReduce

Two-way independent communication between nodes is suppored on Fugaku

Data Partitioning + Bidirectional Ring-like Path
→ Maximize bandwidth utilization



Example of a Bidirectional Ring-AllReduce Channel

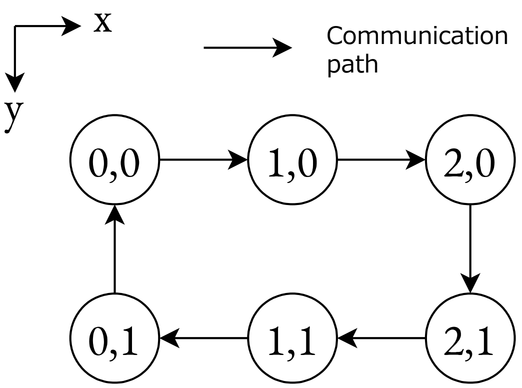
Accelerating All-reduce Communication in Large-Scale Machine Learning on Fugaku, Nakamura et al., IPSJ-HPC-193

Usage Method: Rankmap

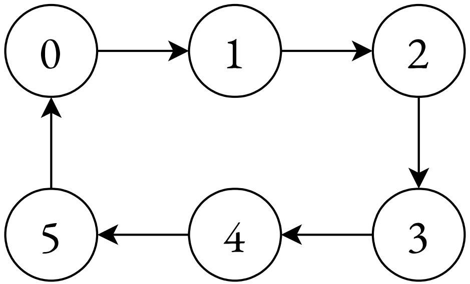
- Rankmap: Correspondence between coordinates and rank

- Ring-AllReduce communicates with adjacency rank

→ Physically adjacent nodes are adjacent by rank
We call it "one-stroke route"



(a) Coordinates and channels



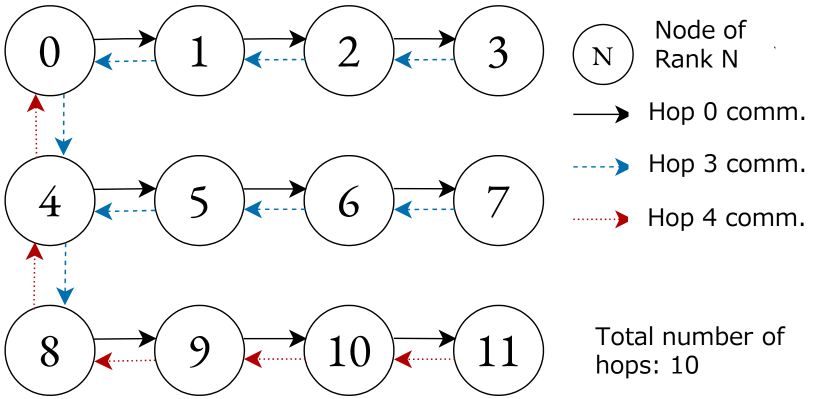
(b) Rank and channel

2 × 3 rankmap example

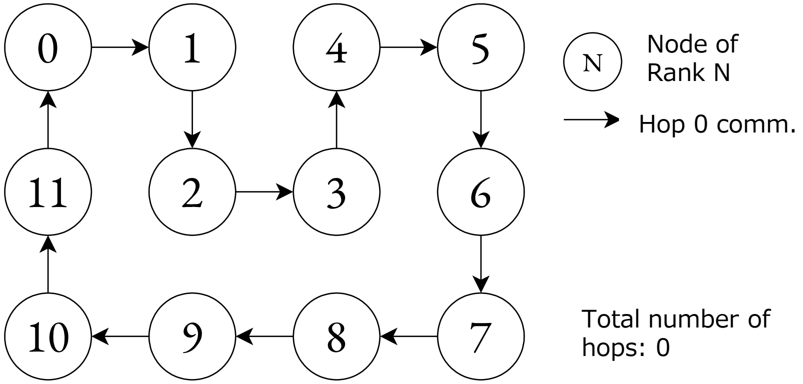
Accelerating All-reduce Communication in Large-Scale Machine Learning on Fugaku, Nakamura et al., IPSJ-HPC-193

Benefits of Rankmap

- Default Assignment: Order of rank as traversing each dimension
  - Communication that is not 0 hop occurs
  - High latency
  - Overlapping communication paths



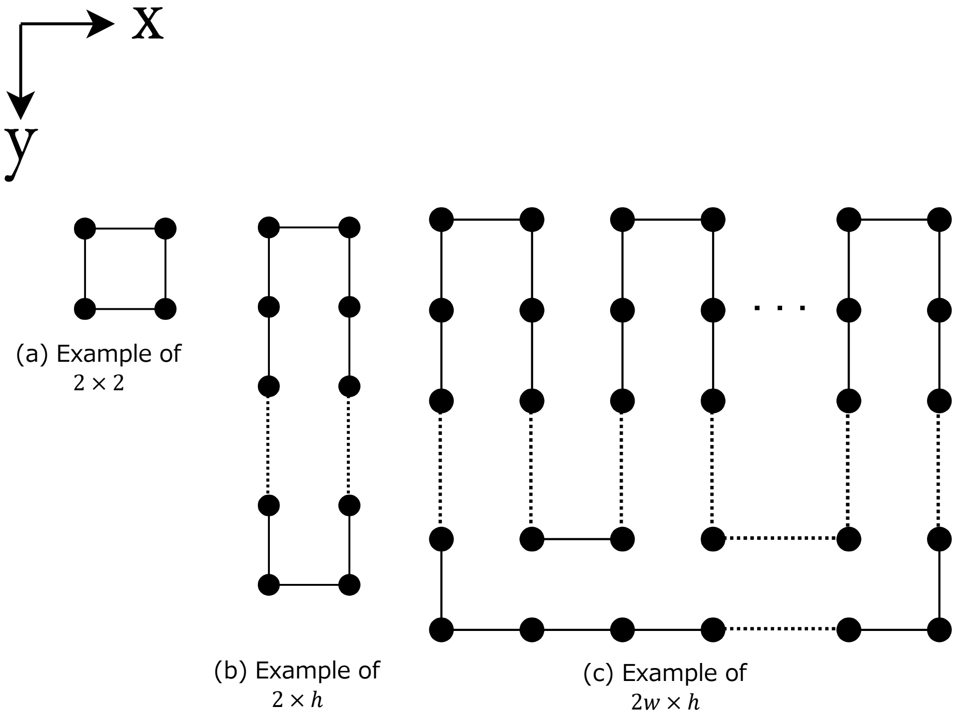
- Using Rankmap to sort rank and coordinates
  - All 0 hop communication is possible.
  - Communication paths do not overlap



Accelerating All-reduce Communication in Large-Scale Machine Learning on Fugaku, Nakamura et al., IPSJ-HPC-193

Proposed Method: Creating a Two-Dimensional rankmap

- Case 2×2
  - As shown in Figure (a)
- Case $2 \times h$
  - Stretch in the y-axis direction by $h - 2$ from figure (a)
- Case $2w \times h$
  - Increases the number of convexities along the x-axis by $w - 1$ from (b)
- <u>Cover except odd x odd</u>



Example of Generating a Two-Dimensional Rank Map

Accelerating All-reduce Communication in Large-Scale Machine Learning on Fugaku, Nakamura et al., IPSJ-HPC-193

Other Speedup Implementation we applied

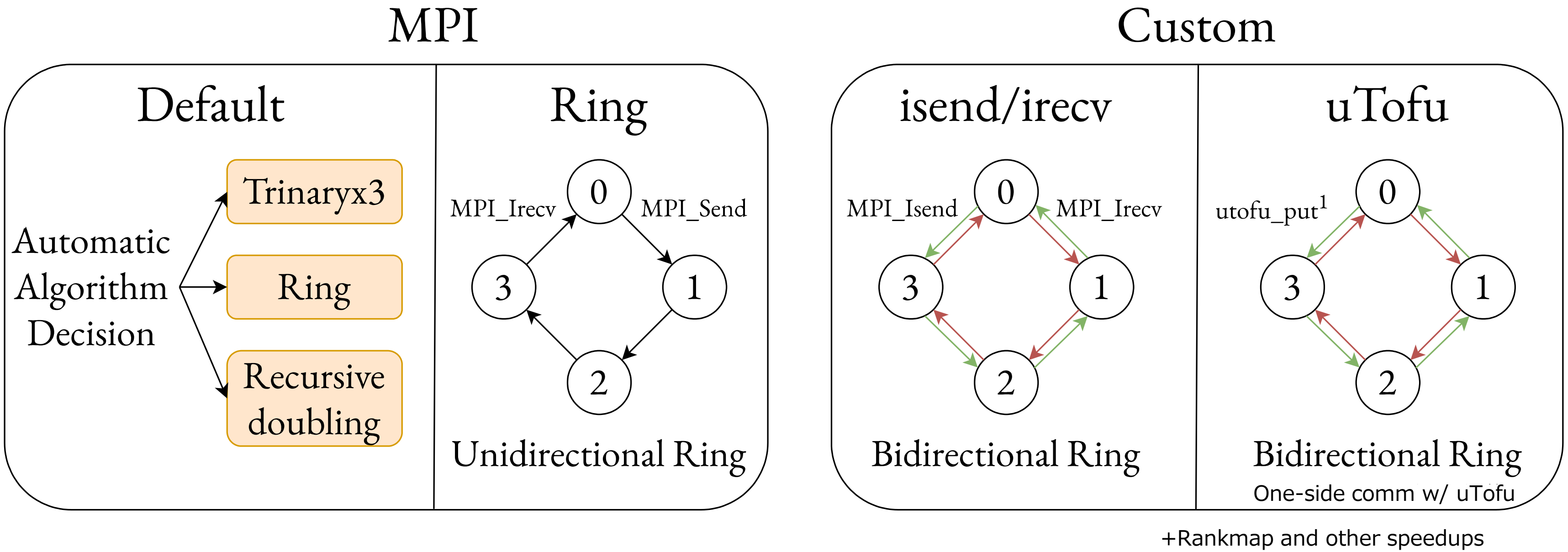
1. Double buffering
   - Hided aggregate calculation time into communication time

2. OpenMP+SIMD
   - Accelerated continuous addition and assignment operations

3. Static allocation of buffers
   - Statically allocate buffer space for inplace operations

Accelerating All-reduce Communication in Large-Scale Machine Learning on Fugaku,
Nakamura et al., IPSJ-HPC-193

Existing and proposed methods

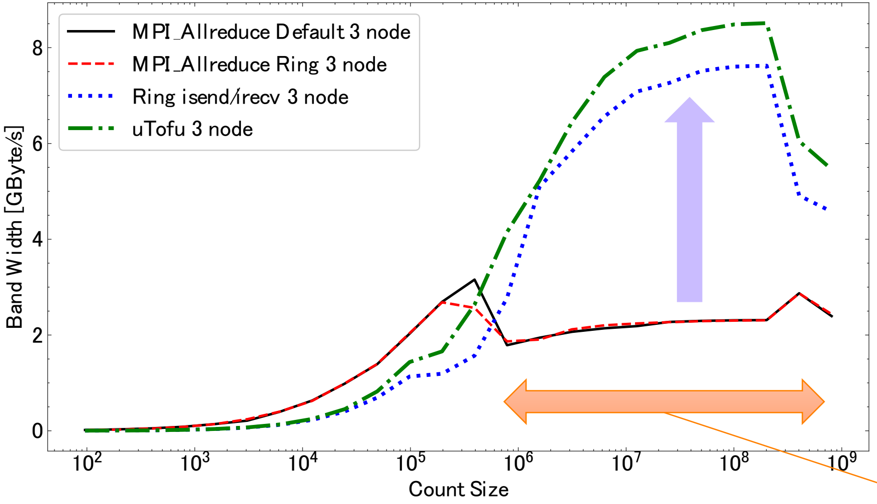


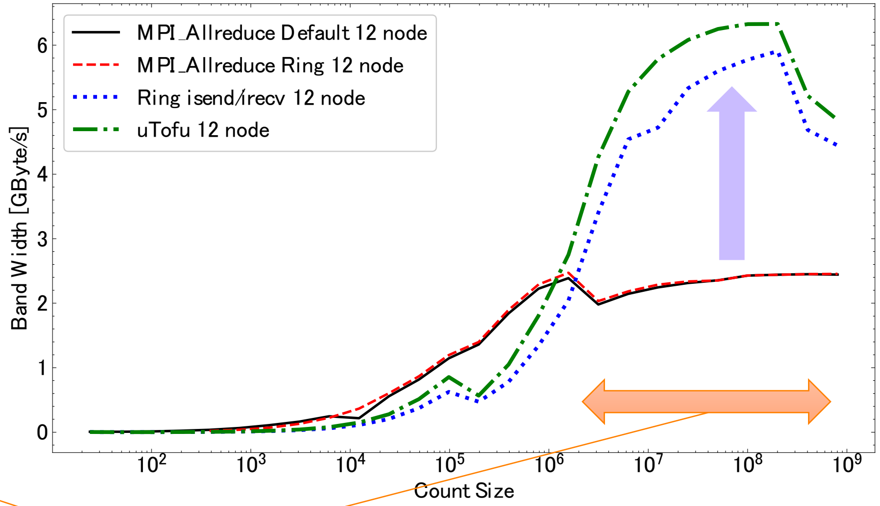
Comparison of existing and proposed methods

Accelerating All-reduce Communication in Large-Scale Machine Learning on Fugaku, Nakamura Akumi et al., IPSJ-HPC-193

Experiment 1: Results (3 nodes, 12 nodes)

FUJITSU

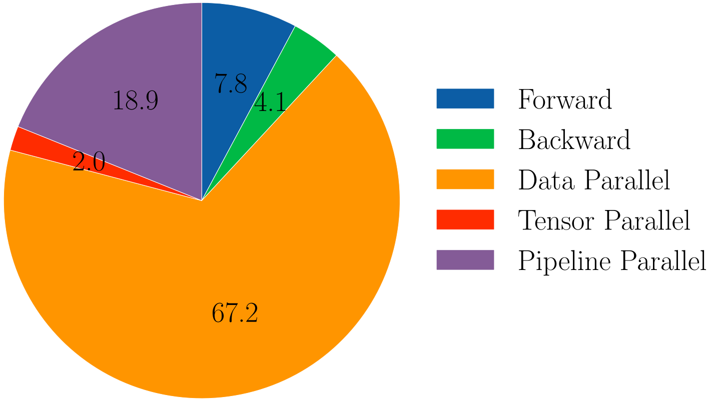




Exceed in the large message length area

Accelerating All-reduce Communication in Large-Scale Machine Learning on Fugaku, Nakamura et al., IPSJ-HPC-193

Experiment 2: Speed performance of language models



13B Model Time Breakdown
TP=6, PP=8, DP=64

$$\mathrm{DP} \times \mathrm{TP} \times \mathrm{PP}$$

AllReduce

adjacency communication

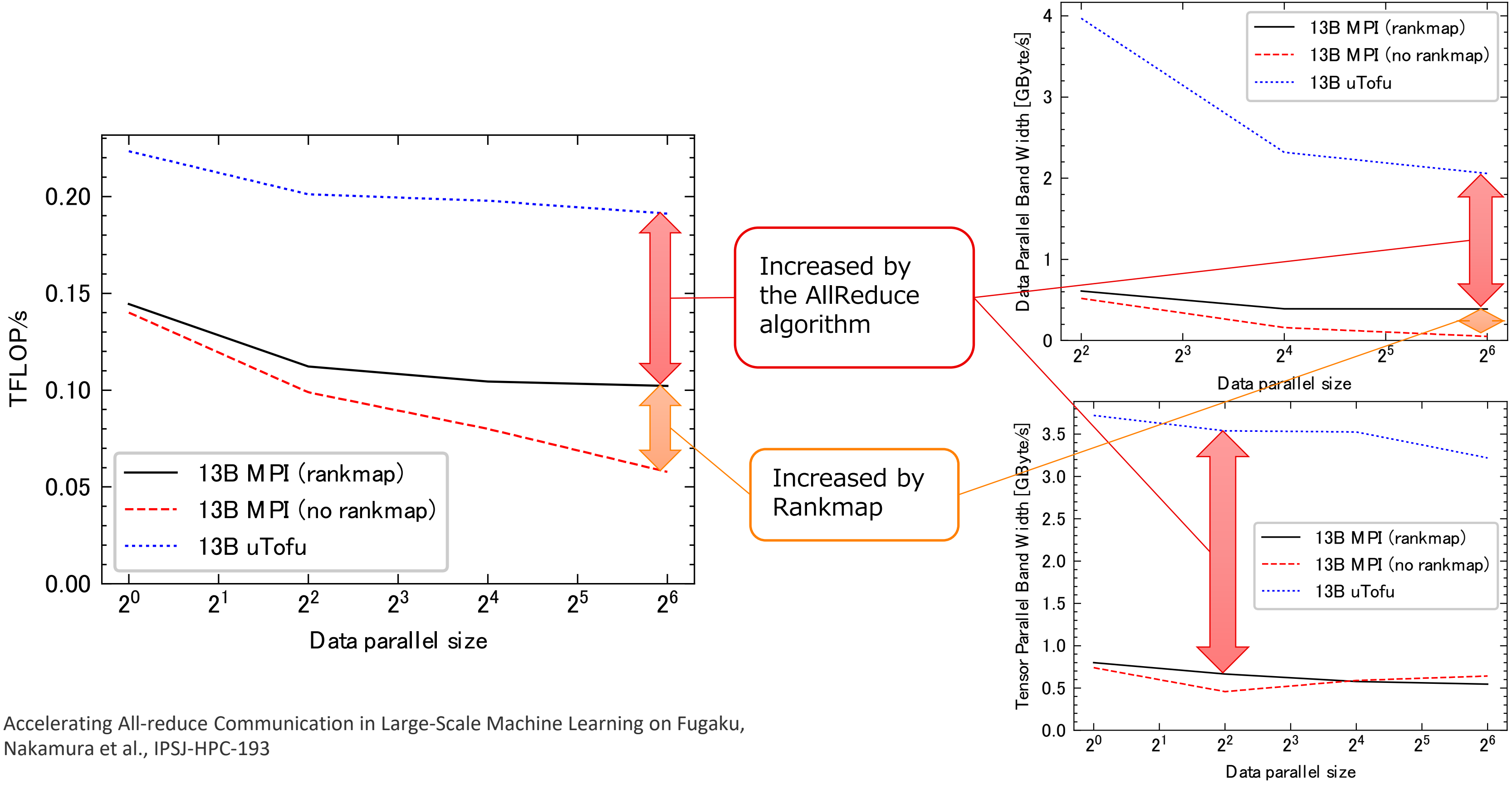
Acceleration methods

AllReduce: Rankmap + our proposed AllReduce
Adjacency communication: Rankmap

DP: Number of nodes for data parallel
TP: Number of nodes for tensor parallel
PP: Number of nodes for pipeline parallel

Accelerating All-reduce Communication in Large-Scale Machine Learning on Fugaku, Nakamura et al., IPSJ-HPC-193

Experiment 2: Results (13B model)



Accelerating All-reduce Communication in Large-Scale Machine Learning on Fugaku, Nakamura et al., IPSJ-HPC-193

Research Results (1): Significantly Improved Computational Performance for Training in Large-Scale Language Models on the Supercomputer "Fugaku"

・Deep learning framework Megatron-DeepSpeed is ported to Fugaku to speed up matrix library on CPU
->**Achieved Six times acceleration (18 seconds instead of 110 seconds)**

1693389241.318550480.fcc.pytorch.y.r1.13_for_a64fx.tar

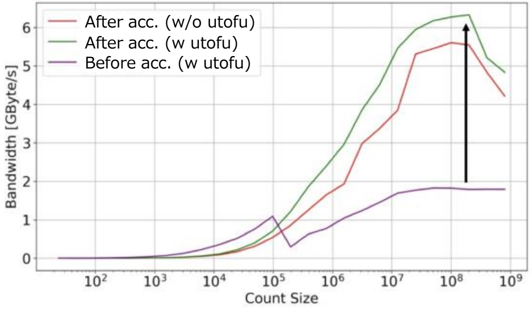
Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	# of Calls
aten::bmm	18.07%	110.819s	18.08%	110.845s	24.055ms	4608
aten::bmm	18.17%	108.802s	18.17%	108.832s	23.618ms	4608
aten::bmm	18.53%	110.858s	18.53%	110.890s	24.065ms	4608
aten::bmm	19.15%	110.594s	19.16%	110.625s	24.007ms	4608
aten::bmm	18.33%	108.646s	18.34%	108.679s	23.585ms	4608

1701935794.711074240.fcc.pytorch.y.r1.13_for_a64fx.tar.gz

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	# of Calls
aten::bmm	3.56%	18.273s	3.57%	18.302s	3.972ms	4608
aten::bmm	3.64%	18.394s	3.64%	18.423s	3.998ms	4608
aten::bmm	3.57%	18.154s	3.57%	18.185s	3.946ms	4608
aten::bmm	3.58%	17.959s	3.59%	17.990s	3.904ms	4608
aten::bmm	3.61%	18.341s	3.62%	18.373s	3.987ms	4608

・Combining three types of parallelization for Fugaku to optimize communication performance and accelerate collective communication on Tofu Interconnect D
->**Achieved three times higher communication speed than before**

Three times speedup



・GPUs are commonly used to train large language models, and the shortage of GPUs around the world has become a social problem. The demonstration that a large language model can be trained with Fujitsu's domestic CPU in Fugaku is an important achievement from the viewpoint of economic security.

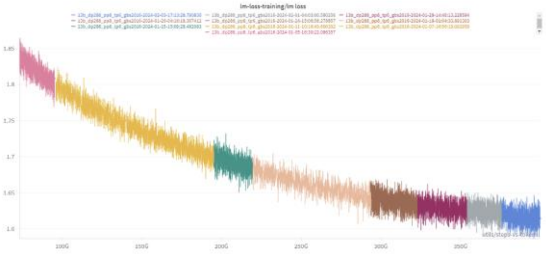
Research Results (2): A large language model with 13 billion parameters that ensures transparency and security, is easy to use and has excellent Japanese performance

・"Fugaku-LLM", a 13 billion parameter model, was trained from scratch using original data.

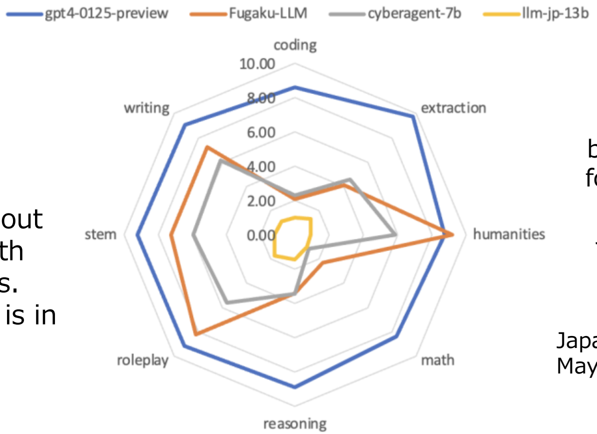
→ While many domestic models use Japanese data for continual training with open models, "Fugaku-LLM" was trained from scratch using its own data, enabling the entire training process to be grasped, with superior transparency and safety.

・Fugaku's 13,824 compute nodes were used for training, and approximately 400 billion tokens approximately 60% of the training data was trained using Japanese content and other combinations of English, math, and code.
(Approx. 2 months of pre-training, Approx. 2 months of post-learning)

→ This results in the highest performance for open models that are Japanese proficient and training on proprietary data in Japan, with an average score of 5.5 on Japanese MT-Bench.
→ The benchmark performance of 9.18 is particularly high for humanities and social studies tasks, and it is expected to engage in dialogue rooted in Japanese language and culture.



Completed training about 400 billion tokens with 13 billion parameters. About 60% of the data is in Japanese.



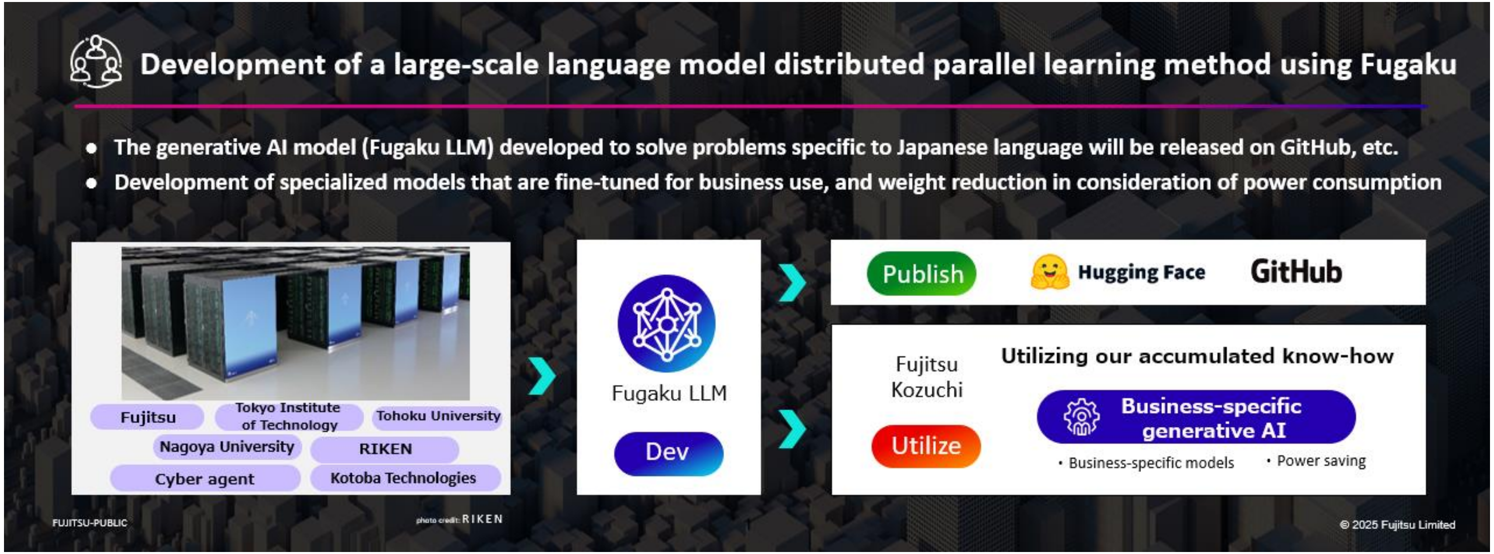
In particular, it shows a high benchmark performance of 9.18 for humanities and social studies tasks. Dialogue rooted in Japanese language and culture is expected

Japanese MT-Benchmark result as of May, 2024

Future Developments

・The seven parties has made their work available to researchers and engineers around the world to develop large-scale language models through GitHub and Hugging Face, which anyone can use for research and commercial purposes under a license.
→ Also, Fujitsu launched Fugaku-LLM on May 10, 2024 through Fujitsu Research Portal, a free trial of Fujitsu's advanced technologies.

・We expect that the participation of many researchers and engineers in the improvement of basic models and new applied research will lead to the creation of efficient methods, and to the "AI for Science" that utilizes AI basic models in scientific research, such as the dramatic acceleration of the scientific research cycle through the collaboration of scientific simulation and generated AI, and to the next generation of innovative research and business results.



Acknowledgement

- This achievement is based on the Government-Initiated Projects of Supercomputer Fugaku "Development of Distributed Training Method for Large Language Models on Fugaku." (Project ID: hp230254).

Thank you