# BaGuaLu: Targeting Brain Scale Pretrained Models with over 37 Million Cores

Zixuan Ma[1], Jiaao He[1], Jiezhong Qiu[1,4], Huanqi Cao[1], Yuanwei Wang[1], Zhenbo Sun[1], Liyan Zheng[1], Haojie Wang[1], Shizhi Tang[1], Tianyu Zheng[3], Junyang Lin[2], Guanyu Feng[1], Zeqiang Huang[3], Jie Gao[3], Aohan Zeng[1,4], Jianwei Zhang[2], Runxin Zhong[1], Tianhui Shi[1], Sha Liu[3], Weimin Zheng[1], Jie Tang[1,4], Hongxia Yang[2], Xin Liu[3], Jidong Zhai[1], Wenguang Chen[1]

Tsinghua University[1], DAMO Academy, Alibaba Group[2], Zhejiang Lab[3], Beijing Academy of Artificial Intelligence[4]

## Abstract

Large-scale pretrained AI models have shown state-of-the-art accuracy in a series of important applications. As the size of pretrained AI models grows dramatically each year in an effort to achieve higher accuracy, training such models requires massive computing and memory capabilities, which accelerates the convergence of AI and HPC. However, there are still gaps in deploying AI applications on HPC systems, which need application and system co-design based on specific hardware features.

To this end, this paper proposes BaGuaLu[1], the first work targeting training brain scale models on an entire exascale supercomputer, the *New Generation Sunway Supercomputer*. By combining hardware-specific intra-node optimization and hybrid parallel strategies, BaGuaLu enables decent performance and scalability on unprecedentedly large models. The evaluation shows that BaGuaLu can train 14.5-trillion-parameter models with a performance of over 1 EFLOPS using mixed-precision and has the capability to train 174-trillion-parameter models, which rivals the number of synapses in a human brain.

***CCS Concepts:*** • **Computing methodologies** → *Parallel computing methodologies.*

***Keywords:*** Artificial Intelligence, Supercomputers, Mixture of Experts, Heterogeneous Architecture

*PPoPP '22, April 2–6, 2022, Seoul, Republic of Korea*
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9204-4/22/04.
https://doi.org/10.1145/3503221.3508417

## 1 Introduction

Deep learning (DL) is of critical importance in many fields, including computer vision (CV), natural language processing (NLP), recommendation systems, and decision-making models. Pretraining of large DL models has reestablished state-of-the-art accuracy in a series of downstream tasks in NLP and CV, indicating that increasing model size can significantly enhance model accuracy [1, 2, 14, 21, 22, 32].

Recent studies have shown that a mixture-of-experts (MoE) approach enables models to scale to a gigantic number of parameters with a constant computation cost [3, 11]. An MoE layer accepts a token representation $x$ as input and routes it to the top-$k$ experts selected from a set of $N$ experts, as determined by a sparse-gated router. MoE has demonstrated broad success in NLP pretraining and downstream applications, especially in machine translation [3, 11, 28, 29].

Although large-scale pretrained models have shown state-of-the-art accuracy in a series of important applications, training large-scale pretrained models is still a very tricky task and requires massive computing, memory, and networking capabilities. GShard [11] and Switch Transformer [3] tackle the problems of complexity, communication cost, and training instability, scaling model size to over one trillion parameters with better accuracy. There is still potential for further scaling up the model size, even to over 100 trillion parameters, which would rival the number in synapses of a human brain.

Although large-scale models can bring promising accuracy, training such models brings unprecedented challenges. As models grow increasingly larger, enormous computing capability, memory capacity, and efficient global communication are desired for training tasks. On the other hand, high-performance computers demonstrate incredible computing power for scientific applications. For example, the latest machine in the Sunway family [4], the *New Generation Sunway Supercomputer*, which can achieve over 1 EFLOPS peak performance and is equipped with approximately 9 PB

[1]BaGuaLu is a magic stove in Chinese ancient mythology which could generate efficacious medicine.

memory, also provides an opportunity for extremely large-scale model training.

However, deploying training tasks on such large-scale HPC systems is far from straightforward. In spite of specific hardware and network topology designed for traditional HPC applications, the following concerns should also be addressed.

**Architecture challenges.** HPC systems often have specifically designed architectures to meet HPC application requirements. Applications and systems must be co-designed to meet specific features of the architecture so that computation resources can be fully utilized to achieve high performance.

**Huge memory capacity.** Training large models requires enormous memory capacity to store model parameters and intermediate results during training. However, different partitioning strategies result in different communication patterns. How to partition parameters, optimizer states, and gradients among workers significantly affects memory usage and is therefore critical when scaling up the model size.

**Parallel strategy.** In order to scale training tasks to a full-scale HPC system, parallel strategies must be redesigned because current strategies are inefficient when scaled to such large systems. For example, MoE brings huge *All-to-All* communication requirements for directing inputs to experts, whereas *data parallelism* introduces a huge *All-Reduce* communication for parameter updating with averaged gradients. Considering that *Sunway* has as many as 96,000 nodes and network bandwidths differ between intra- and inter-supernodes, communication during training must be carefully designed and optimized. Moreover, load imbalance also becomes severe when scaling the model to a whole supercomputer.

**Mixed-precision.** Unlike traditional HPC applications that mainly use double-precision computations, most DL applications use single- or mixed-precision to maximize computation throughput. Because *Sunway* provides support for different types of floating-point computations, including FP64, FP32, FP16, and BF16, how to combine different types efficiently and effectively for mixed-precision training on an unprecedentedly large model is critical for both research and industry.

To address these issues, this paper proposes BaGuaLu, the first work targeting training brain scale models on an exascale supercomputer, called the *New Generation Sunway Supercomputer*. BaGuaLu enables training up to 14.5-trillion-parameter models with up to 1.002 EFLOPS. Additionally, BaGuaLu has the capability to train models with up to 174 trillion parameters, which rivals the number of synapses in a human brain.

The main contributions of this work are listed as follows:

- Efficient hardware-specific intra-node optimizations for the *New Generation Sunway Supercomputer*, including core scheduling, memory segmentation, and memory access.
- A parallel strategy, *MoDa*, which combines *MoE parallelism* and *data parallelism* to scale the model to the whole supercomputer.
- A distributed optimizer, *ParO*, which effectively reduces computing time and memory usage.
- A new load balancing strategy, *SWIPE*, for MoE to reduce the waste of computation resources.
- A layer-wise mixed-precision strategy that can optimize the training process without affecting convergence.
- A demonstration that BaGuaLu can train models with up to 14.5 trillion parameters with over 1 EFLOPS performance. BaGuaLu also shows the capability to train brain scale models (over 100 trillion parameters).

## 2 Background and Related Work

### 2.1 Transformer Model and its Training

This subsection introduces the transformer model[31] as the application target of this paper. The model accepts input sequences and feeds them into multiple transformer blocks. Each transformer block contains a multi-head attention layer followed by a point-wise feed-forward network (FFN). GShard[11] and Switch Transformer[3] further replace the FFNs with mixture-of-experts (MoE) layers as described by Shazeer et al.[29]. An MoE layer routes each token to top-$k$ experts (FFNs) selected from multiple experts, determined by a sparse-gated router. The overall structure of the transformer model is leveraged with MoE.

Considering the training process of such a gigantic transformer model, the following here discussion addresses its computation and communication characteristics. The computation flow of the training process is shown in Figure 1. The training data are distributed to the memory of each node. In each training iteration, a batch of sampled image and text data are concatenated into sequences as the model inputs. As experts are sharded across nodes, the MoE layer contains two *All-to-All* communications to dispatch and combine tokens. The computation on the layers introduces both large and batched smaller matrix multiplications at a precision of FP32 or lower. After the forward pass, the language model loss is evaluated for gradient computation using back-propagation. The computation and communication patterns of a backward pass are similar to those of a forward pass. After the backward pass, the gradients are accumulated by an *All-Reduce* operation, and the parameters are updated by gradient descent. The above process is repeated for multiple rounds until the model converges.

### 2.2 Related Work

**Growing size of pretrained models** The size of deep learning models has been growing rapidly in recent years. As

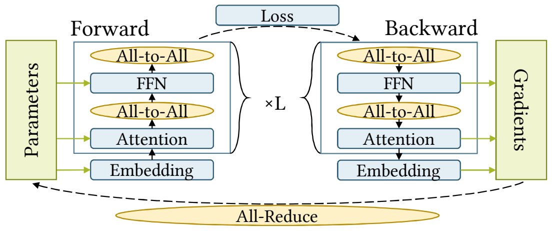

**Figure 1.** Simplified computing process of the proposed model.

shown in Table 1, the model size has developed from ELMo with 94 million parameters to Switch Transformer with 1.6 trillion parameters, with an increase by 1 or 2 orders of magnitude each year. Therefore, training systems for extremely large-scale models are desired.

**Table 1.** Large-scale models released in recent years.

| Name | Model Size | Date |
|---|---|---|
| ELMo [19] | 94 million | Feb 2018 |
| GPT [20] | 110 million | Jun 2018 |
| BERT-Large [2] | 340 million | Oct 2018 |
| Transformer (Mesh-TensorFlow) [28] | 5 billion | Nov 2018 |
| Transformer (GPipe) [6] | 6 billion | Nov 2018 |
| GPT-2 [21] | 1.5 billion | Feb 2019 |
| GPT-2 like (Megatron-LM) [30] | 8.3 billion | Sep 2019 |
| Transformer (GShard) [11] | 600 billion | Jun 2020 |
| Turing-NLG [23, 25] | 17 billion | Feb 2020 |
| GPT-3 [1] | 175 billion | May 2020 |
| Switch Transformer [3] | 1.6 trillion | Jan 2021 |
| **BaGuaLu (ours)** | 174 trillion | April 2021 |

**Large-scale model training** Training dramatically large-scale models faces enormous challenges that various approaches and systems have been proposed to address. Mesh-TensorFlow [28] provides a language to help implement *model parallelism*, which can train transformer models with up to 5 billion parameters at a sustained performance of 6 PFLOPS. GPipe [6] is a framework for implementing *pipeline parallelism*. A transformer model with 6 billion parameters was trained using GPipe. Megatron-LM [30] implements a hybrid parallel strategy for language models, including *model parallelism*, *data parallelism*, and *pipeline parallelism* across GPUs among different GPU servers. They reported the training of a larger model with 8.3 billion parameters at a sustained 15.1 PFLOPS performance. GPT-3 [1] is a model with 175 billion parameters, trained using a mixture of *model parallelism* and *pipeline parallelism*.

ZeRO [23] from DeepSpeed [25], on the other hand, partitions optimizer states, gradients, and/or model parameters among workers to reduce memory usage. ZeRO does not change the parallel strategy but enables *data parallelism* on larger models. Turing-NLG, a model with 17 billion parameters, was trained using ZeRO. However, ZeRO cannot fit into the parallel strategy requirement here, which is beyond plain vanilla *data parallelism*.

To further scale up the model size, MoE is a promising approach that improves scalability by means of a dynamic model design. However, the dynamic mechanism poses challenges for systems. GShard [11] implemented a transformer model with MoE. Instead of using traditional parallel strategies across workers, GShard takes advantage of the scalability of MoE by assigning one MoE expert to each worker. The model consists of over 600 billion parameters, and the training was scaled out to 2, 048 TPUs. Switch Transformer [3], implemented based on Mesh-TensorFlow, further pushes the size limit. Although Switch Transformer provides MoE with *data parallelism* and *model parallelism*, the mixture has only been applied to a small-sized model. Their largest model, with 1.6 trillion parameters, uses only MoE.

**Deep learning on supercomputers** Training large-scale DNN models requires huge computing capability and memory capacity, which converges with the advantage of HPC systems. Various approaches have been proposed to deploy training tasks on HPC systems in recent years, as listed in Table 2. Kurth et al. [8] proposed a deep learning system for solving scientific pattern classification problems, which obtained a sustained (peak) 13.27 PFLOPS (15.07 PFLOPS) throughput. Kurth et al. [7] trained a DeepLabv3+ model at a sustained (peak) 999.0 PFLOPS (1.13 EFLOPS) throughput. Patton et al. [17, 18] achieved sustained 152.5 PFLOPS and 1.301 EFLOPS throughput with a system named MENNDL, which discovers neural models automatically. Laanait et al. [9] trained a fully convolutional neural network at a sustained (peak) 1.54 (2.14) EFLOPS throughput.

**Table 2.** Models with corresponding training platforms and training FLOPS

| Work | Platform | Sustained FLOPS | Peak FLOPS |
|---|---|---|---|
| Kurth et al. [8] | Cori Phase-II | 13.27 PFLOPS | 15.07 PFLOPS |
| Patton et al. [17] | Summit | 152.5 PFLOPS | - |
| Kurth et al. [7] | Summit | 999.0 PFLOPS | 1.13 EFLOPS |
| Patton et al. [18] | Summit | 1.30 EFLOPS | - |
| Laanait et al. [9] | Summit | 1.54 EFLOPS | 2.15 EFLOPS |

Unlike the approaches discussed above, BaGuaLu scales the training tasks up to 96, 000 nodes, which is far more than previous works and poses more challenges for communication. Moreover, the pretrained model used in this study also brings significantly more complicated communication requirements that no previous work has encountered.

## 3 System Architecture

Primarily targeting traditional HPC workloads, as the latest machine in the Sunway family, the *New Generation Sunway Supercomputer* starts to provide support for AI-based workloads. As the first full-scale AI training task on *Sunway*, we

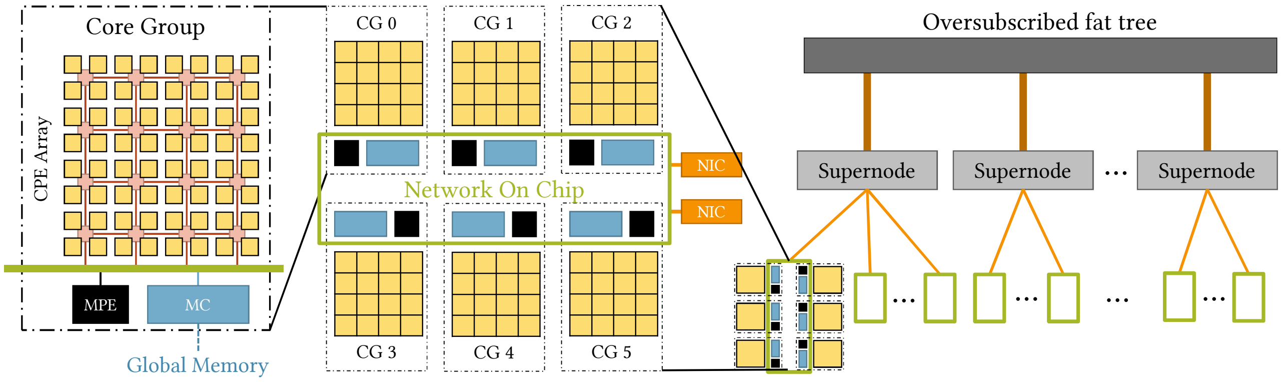

**Figure 2.** Architecture of the *New Generation Sunway Supercomputer*.

here list the challenges brought about by this new architecture at an unprecedented scale.

**Compute node** In *New Generation Sunway Supercomputer*, each compute node has one *SW26010-Pro* heterogeneous CPU, which consists of 6 core groups (CG). These CGs are connected by a network on chip (NoC). Each CG consists of 1 management processing element (MPE) and an array of 64 computing processing elements (CPE). The MPE is a fully functioned 64-bit RISC processor core that can take responsibility for resource management and operating system functions. The CPE array will be discussed in detail later. A group of DRAM channels are also attached to each CG but are visible for all CGs through NoC.

**Memory segment** The memory of *Sunway* can be classified into three segments: the *cross segment*, *share segment*, and *private segment*. The memory space in the cross segment is interleaving addressed to all six CGs, which supports synchronous memory access from six CGs. The memory space in the share segment is shared memory space for both MPE and CPEs within the same CG, which supports synchronous memory access from one CG. The memory space in the private segment is private memory space for each CPE. To achieve higher scalability, programmers need to use this node-level parallelism rather than simply creating one process on each CG.

**CPE array architecture** In a CPE array, each CPE is a simplified RISC core with 512-bit SIMD extension for high floating-point-processing performance but limited functionality. The 512-bit SIMD extension instruction set of CPE cores enables 8 FP64/FP32 or 32 FP16/BF16 data to be processed in a single instruction. Each CPE has an on-chip SRAM that can be configured into a combination of scratchpad memory (i.e., local data memory or LDM) and cache (i.e., local data cache or LDC). Efficient communication between CPEs and DRAM can be implemented with asynchronous direct memory access (*DMA*) to LDM or accesses through LDC. In addition to sharing data through DRAM, CPEs in the same CG are connected by an intra-CG network, enabling remote memory access (*RMA*) with explicit one-sided primitives, "get" and "set", copying between LDMs of different CPEs. This unique accelerate architecture, together with limited DRAM bandwidth, poses great challenges in designing highly optimized implementations for operators used in our models.

**Interconnection network and I/O** In *New Generation Sunway Supercomputer*, compute nodes are connected with layered homegrown interconnect fabrics. The topology used is a 16/3× oversubscribed fat tree[10]. A non-blocking fabric is provided within a supernode of 256 contiguous compute nodes to ensure full peer-to-peer bandwidth. The communication bandwidth across supernodes is 3/16× of that within a supernode. To use the interconnect efficiently, programmers must carefully map various levels of parallelism to different compute nodes in applications. Alongside the interconnect, the I/O subsystem connected to global storage uses a distinct network, making it possible to overlap I/O with communication without losing performance.

## 4 Methodology

BaGuaLu is the first work aiming at efficiently training brain-scale pretrained models on an entire exascale supercomputer. By leveraging HPC techniques, hardware specializations, and application characteristics, we propose four key innovations: *hardware-specific intra-node optimization*, *an efficient hybrid parallel strategy*, *an efficient I/O implementation*, and *mixed-precision training*. Overall, BaGuaLu focuses on efficient and adaptive training for large-scale models exploiting the features of different system levels, from underlining processor cores to an entire system.

BaGuaLu offers user-friendly programming interfaces and a portable model format by leveraging a PyTorch frontend. We implement a carefully optimized operator library customized for the *New Generation Sunway Supercomputer* to deliver high performance on top of hardware specifications. Our operator library includes over 60,000 lines of C/C++ code to enable all the optimization. BaGuaLu also provides a high-performance checkpoint mechanism for fault tolerance.

The rest of this section introduces the techniques involved in BaGuaLu.

### 4.1 Hardware-Specific Intra-node Optimization

To fully utilize the computing capability of the *New Generation Sunway Supercomputer*, the intra-node performance should first be optimized. Careful design is needed to meet the hardware specification.

**Core scheduling** The cross segment of memory within one compute node provides programmers the opportunity to access the same memory space among different CPEs. We manage all the CPEs within one compute node using one process running on only one MPE. There are several benefits in this way: (1) Communication between processes requires message passing on *Sunway*, which is performed across NICs. Using one process to control all the CPEs in one node can effectively avoid extra communication overhead. (2) The remaining five MPEs can be used to handle communication, I/O, and other lightweight tasks to further reduce overhead, as shown in Figure 3. (3) The number of total processes is reduced from 576,000 (96,000 × 6) to 96,000, which can significantly reduce the I/O pressure of loading BaGuaLu processes.

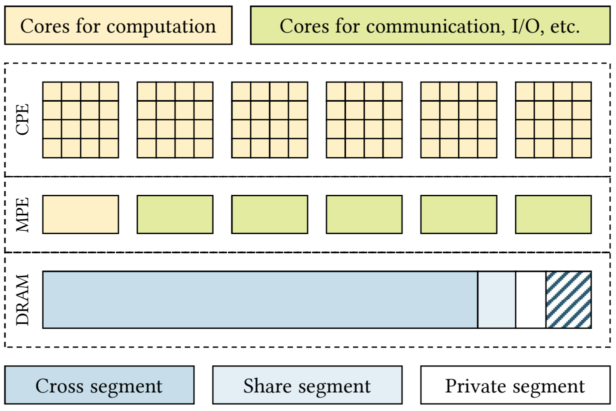

**Figure 3.** Core scheduling and memory segmentation.

**Memory segmentation** To use memory efficiently, the 96 GB memory of one node is partitioned into three parts: 84 GB for the cross segment, 3,600 MB (600 × 6, 600 MB for each CG) for the share segment, and 768 MB (2×64×6, 2 MB for each CPE) for the private segment. The cross segment is used for computation and communication, whereas the share and private segments are used for OS libraries and CPE libraries, respectively. The rest of the memory is managed by the OS.

**Memory access** On *SW26010-Pro*, global load/store can only reach a bandwidth of 0.24 GB/s. To fully utilize memory bandwidth, *DMA* and *RMA* (Section 3) are used instead of naive global load/store to improve the performance of operators.

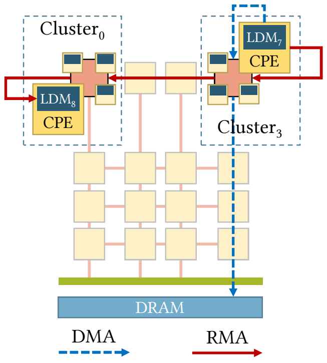

**Figure 4.** *DMA* and *RMA* on *SW26010-Pro*

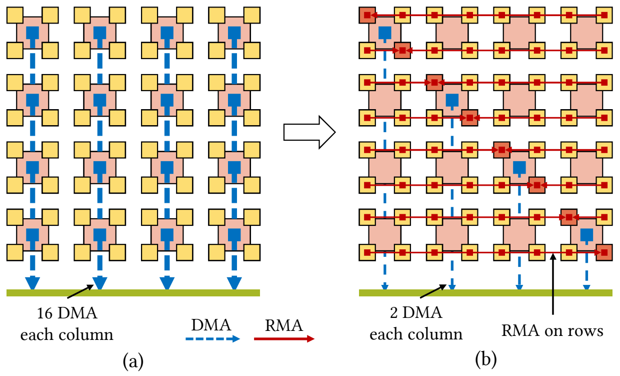

**Figure 5.** Memory access from CPE using *DMA* and *RMA*.

As shown in Figure 4, on *SW26010-Pro*, four adjacent CPE cores are organized as a cluster. Clusters are connected by links on rows and columns. A CPE accesses another CPE's data using *RMA* through row links or column links and accesses data on DRAM using *DMA* through column links and NoC.

The CPEs on the diagonal are responsible for all *DMA* requests of the entire CPE array, with each CPE on the diagonal being responsible for its row. Taking a *DMA* load as an example, as shown in Figure 5, the data are first transferred to the CPEs on the diagonal through *DMA* and then transferred to each CPE through *RMA*. Compared to calling *DMA* on every CPE, *DMA+RMA* can efficiently reduce the number of *DMA* calls on NoC (384 to 48 with 6 CGs) and balance them between column links (2 on each link). Meanwhile, this method increases the granularity of *DMA* calls which can potentially optimize memory bandwidth.

In cross memory, the memory space is addressed to all six memory devices by interleaving. By rearranging *DMA* calls, the load of 6 devices can be balanced for each *DMA* call. In this way, the cross memory can be effectively used.

After the above optimization, the memory bandwidth can approach the theoretical value. Taking matrix multiplication as an example, the computation overlaps with asynchronous communication and memory access to hide latency. In this way, 89.2% and 85.8% of the peak performance are obtained for single- and half-precision *GEMM*, respectively.

### 4.2 Efficient Hybrid Parallel Strategy

To scale BaGuaLu to the entire *Sunway*, we design an efficient hybrid strategy including the *hybrid parallel strategy MoDa*, the *load balance policy SWIPE*, and the *memory-efficient optimizer ParO.*

**Hybrid MoE parallelism and data parallelism strategy (MoDa)** To scale model training to brain scale with promising computing efficiency, we propose *MoDa*, a hybrid parallel strategy of *MoE parallelism* and *data parallelism*, which amply meets both the hardware specification and the application characteristics. There are 3 types of communication in each training task: 1) *All-to-All* feature exchange in forward and backward passes within each MoE group; 2) *All-Reduce* gradient synchronization for each expert crossing an MoE group; 3) global *All-Reduce* gradient synchronization for other parts of neural networks. Communications occur in rows and columns with the *data parallelism* groups and MoE exchange groups.

An optimal mapping from the communication graph to the network topology in *Sunway* is necessary to achieve high performance. In this specific case, we map 2 types of communication traffic, i.e., *All-to-All* for *MoE parallelism* ($T_{a2a}$) and *All-Reduce* for ($T_{ar}$), to 2 different networks, i.e., communication within a supernode with a bandwidth of $W$ and communication across supernodes with a bandwidth of $W_{sub} = \frac{3}{16}W$. To minimize the overall latency, we should assign $T_{a2a}$ and $T_{ar}$ to $W$ and $W_{sub}$ which contains two approaches. One is *All-to-All* within a supernode and *All-Reduce* across supernodes, which has a theoretical latency of $\frac{T_{a2a}}{W} + \frac{T_{ar}}{W_{sub}}$. The other is *All-Reduce* within a supernode and *All-to-All* across supernodes, which has a theoretical latency of $\frac{T_{ar}}{W} + \frac{T_{a2a}}{W_{sub}}$. According to the calculation, in our target large-scale models, the expert *All-Reduce* communication is 16 to 24× larger than the *All-to-All* communication between experts, i.e., $T_{ar} > T_{a2a}$, indicating $\frac{T_{ar}}{W} + \frac{T_{a2a}}{W_{sub}} < \frac{T_{a2a}}{W} + \frac{T_{ar}}{W_{sub}}$. Therefore, as shown in Figure 6(a), although the *All-to-All* operations are critical in latency and are more affected by oversubscription on *Sunway* by our micro-benchmark testing, we decide to place the *data parallelism* ranks within one supernode and leave the MoE communication globally across supernodes. This strategy is opposite to the intuition that *All-to-All* communication should be assigned to the faster intra-supernode interconnection.

**SunWay Imbalance Proficiently Eliminated (SWIPE)** Load balance between experts is critical to achieving high throughput, especially for an extremely scaled system. We observe that the load balance strategies used in GShard [11] and Switch Transformer [3] still left certain experts with only 30% utilization, leading to very low system efficiency.

To address this challenge, we propose *SWIPE*, a load balance strategy targeting super-large systems. It proficiently distinguishes load imbalance by re-assigning input items to experts with a strict limit that each expert must receive an equal number of input items. Because latency of performing assignments is also critical to overall performance, *SWIPE* is designed as a distributed algorithm that requires a constant number of communication operations, which is significantly less than a voting-based algorithm [12] that requires up to $O(n)$ rounds of communication.

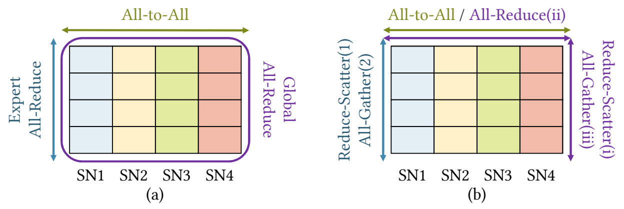
All-to-All
Expert All-Reduce
Global All-Reduce
SN1 SN2 SN3 SN4
(a)
All-to-All / All-Reduce(ii)
Reduce-Scatter(1) All-Gather(2)
Reduce-Scatter(i) All-Gather(iii)
SN1 SN2 SN3 SN4
(b)

**Figure 6.** (a) *MoDa* with inefficient global *All-reduce*; (b) *MoDa* enhanced by *ParO*, in which expert *All-Reduce* is replaced by *Reduce-Scatter(1)* and *All-Gather(2)*, global *All-Reduce* is replaced by *Reduce-Scatter(i)* within a supernode, *All-Reduce(ii)* across supernodes, and *All-Gather(iii)* within a supernode.

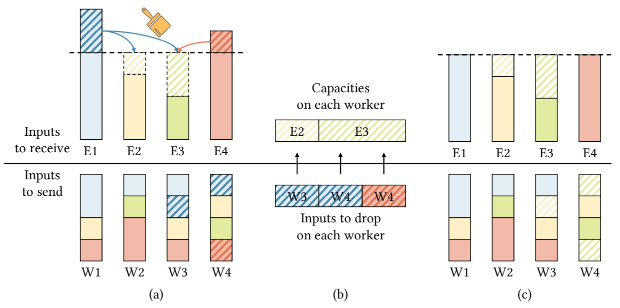

**Figure 7.** Demonstration of how *SWIPE* balancing MoE workload.

In detail, *SWIPE* works as follows. Assume that there are $n$ workers. Each worker has one expert and $b$ inputs to dispatch. $n \times b$ scores are given for each input item to decide its expert. We first attempt to assign every input item to the expert with the highest score. When an expert is overloaded, i.e., it receives more than $b$ inputs, it drops the rest of its incoming inputs, which will be re-assigned, as shown in Figure 7(a). At the same time, each expert works out the capacities after the first attempt. Then, all workers iterate a local lookup table, which records each expert's capability and dropped inputs, and use an identical deterministic algorithm to map the dropped inputs to available experts, as shown in Figure 7(b), which consumes $O(n + b)$ time. In this way, each worker is guaranteed to receive exact $b$ inputs, as shown in Figure 7(c). Note that *SWIPE* can be extended to allow each input to go to its top-$k$ highest scored experts, where $k$ is a given constant, commonly 2 in existing systems [3, 11].

**Parallel partition-based optimizer (ParO)** To scale model size to brain scale, memory capacity is the main challenge that should be addressed. For each parameter, the parameter itself, its gradient, and the variables in the optimizer, e.g., *moments* in Adam[15], must be stored. Among them, the optimizer status can take several times the memory space of the parameter itself, i.e., 6× when using the Adam optimizer with mixed-precision training, leading to huge memory usage.

To solve this problem, we propose *ParO*. Similar to DeepSpeed [23], we replace the *All-Reduce* in a *data parallelism* group with *Reduce-Scatter* and *All-Gather* to partition the optimization states across workers. However, unlike DeepSpeed on GPU clusters, *ParO* works on a much larger supercomputer and interacts with *MoDa*, which introduces synchronization between different groups of parameters across different groups of workers. Given that there can be up to 240 ranks in a *data parallelism* group for each expert, the space taken by optimizer states becomes negligible, e.g., less than 5%, compared to the parameters, gradients, and intermediate results.

In addition, in the proposed transformer model, one-seventh to one-tenth of the local parameters are shared across all workers. Because the global synchronization can be very expensive given the fat-tree oversubscription, we use a hierarchical partition and replication strategy. Workers are placed in a 2D grid. Gradients are first *Reduce-Scattered* in one dimension, then *All-Reduced* in another dimension. After updating, the parameters are sent back along the first dimension through an *All-Gather*. The first dimension is mapped to the processes within a supernode, and the second is mapped across supernodes. As shown in Figure 6(b), *All-Reduce* on gradients of globally shared parameters and later optimization is replaced with topology-aware *ParO* optimization.

*ParO* also makes it possible to recover a completed copy of the model parameters from the optimizer, simplifying checkpointing by saving only metadata and optimizer data on each rank.

## 4.3 Efficient I/O Implementation

To improve the availability of BaGuaLu, an effective I/O module for AI training is needed on top of the current HPC file systems to support both the data loader and checkpoints.

**Data loader** Unlike traditional HPC applications, the machine learning dataset is organized into collections of samples that are randomly selected from the dataset during training, which leads to random I/O accesses using straightforward solutions.

Taking advantage of the petascale distributed memory and efficient interconnection network of *Sunway*, we design a general data loader for AI training and convert random I/O accesses to highly-efficient collective communications over a high-speed interconnection network. We first pre-process the dataset into a number of fragments (files), which were stored in a global file system. The worker on each compute node uniformly partitions the fragments, loads them into memory, and assigns a unique ID for each sample. Targeting the oversubscription network, we proposed a hierarchical batch generator to simultaneously obtain performance and randomness. During each iteration, workers cooperate by MPI collective communications(mainly *All-to-All*) to select samples and generate batches. Within an epoch, a worker only obtains inputs from the workers in the same supernode. For each epoch, workers randomly re-partition the dataset by reloading it from the file system. Therefore, the communications will not be oversubscribed and achieve higher performance, preventing the data loader from becoming the system's bottleneck.

**Checkpoints** In the global file system in *Sunway*, the bandwidth is influenced by both the number of processes and the number of supernodes. Therefore, the straightforward approach of using one *MoE parallelism* group to write the whole checkpoint is performance unfriendly. We need to use the full system to utilize the bandwidth. As illustrated in Section 4.2, we place the *data parallelism* within a supernode and *MoE parallelism* across supernodes. As each expert maintains different parameters, the ranks in each supernode need to write independently. To scale the number of supernodes, we need to partition the parameters and optimizer states and save the checkpoints with each supernode. As the parameters and optimizer states are already partitioned in *ParO*, as illustrated in Section 4.2, each node only needs to write its partition to the global file system, which utilizes the bandwidth of the global file system and keeps all the data written only once.

## 4.4 Mixed-precision Training

*SW26010-Pro* supports different types of floating-point computations: FP64, FP32, FP16, and BF16. The first two use the same computing units, leading to the same performance, and so do the last two. The throughput of FP64 and FP32 is 14.03 TFLOPS, and the throughput of FP16 and BF16 is 55.30 TFLOPS. Therefore, training with mixed-precision leads to significant performance improvement.

However, existing methods are impractical for direct use on the *Sunway*. Most existing work is based on GPUs and has not been verified on such large-scale model training. For example, NVIDIA APEX [16] has four optimization levels over FP32 training:

- O0: FP32 training.
- O1: Use FP16 in certain operators, such as *GEMM* or convolution. The inputs will be cast before and after the mixed-precision operation.
- O2: Cast weights and input data to FP16 and maintain an FP32 master weight for optimization.
- O3: FP16 training.

Usually, when training regular models on NVIDIA GPUs, O1- and O2-level training can converge to the same loss as O0. However, on *Sunway*, when the APEX method is used directly, neither of the O1- and O2-level optimizations

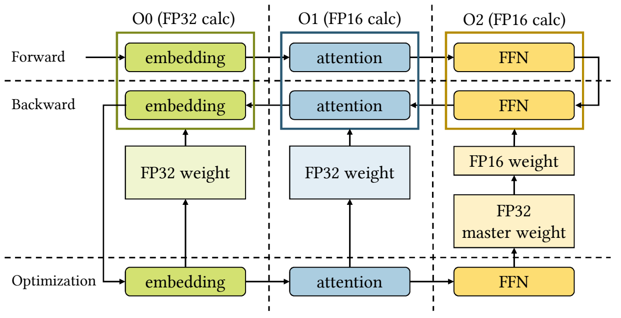

**Figure 8.** Mixed-precision training on BaGuaLu.

works. O1-level optimization cannot achieve promising performance because it requires much data type casting, which will significantly degrade its performance on *SW26010-Pro* due to the limited memory bandwidth. O2-level optimization on *Sunway* cannot converge because it uses too much FP16 data causing massive data overflow. Therefore, mixed-precision training must be explored on extremely large-scale models based on the *Sunway* to gain better performance.

It was observed that different layers have different data patterns, leading to different contributions to the model convergence. Therefore, we classify all the layers into different categories, and for each category, we carefully tune the optimization level and choose the best one so that we can achieve both high performance and quick convergence. As shown in Figure 8, we use O0 on the embedding layer, O1 on attention layers, and O2 on FFN (feed-forward) layers.

Moreover, dynamic loss scaling [33] was used to prevent the underflow of FP16. To guarantee numerical stability and avoid possible overflow and underflow, we use FP64 for several specific operators such as *reduction*, *exp*, *sqrt*, *gelu*, *softmax*, and *layer_norm*, based on the profiling results obtained. In *All-Reduce* communication, we employ an online average algorithm instead of the simple sum-and-divide algorithm to improve numerical stability.

## 5 Evaluation

This section presents an evaluation of BaGuaLu. We first introduce the multi-modality model and dataset in Section 5.1. Then, we validate the core technique, including intra-node optimizations (Section 5.4), mixed-precision (Section 5.5), and parallel strategy (Section 5.6 and Section 5.7). The loss curve of the proposed model is shown in Section 5.8. Finally, we discuss the scalability and breakdown for the proposed system in Section 5.9 and Section 5.10.

### 5.1 Model Setup

The performance of the various models is presented with the proposed hybrid strategy for *MoE parallelism*and *data parallelism* (*MoDa*), where *All-to-All* communication is assigned to inter-supernode connections and *All-Reduce* communication is assigned to intra-supernode connections. Unless indicated, all models use the proposed parallel partition-based optimizer (*ParO*). The detailed model hyper-parameters, as well as the model size, are listed in Table 3. We use a hidden dimension size of $d_{\text{model}} = 4,096$ for all models, but different FFN inner layer dimensionalities $d_{\text{ff}}$.

**Table 3.** Models used for evaluation. $d_{\text{model}}$ and $d_{\text{ff}}$ represent the hidden dimension size and the FFN inner layer size, resp.

| Model | Params | Layers | Heads | $d_{\text{model}}$ | $d_{\text{ff}}$ | Experts |
|---|---|---|---|---|---|---|
| *MoDa-1.16T* | 1.160T | 12 | 8 | 4096 | 4096*12 | 240 |
| *MoDa-1.93T* | 1.934T | 12 | | | 4096*12 | 400 |
| *MoDa-14.5T* | 14.50T | 10 | | | 4096*18 | 2400 |
| *MoDa-174T* | 173.9T | 3 | | | 4096*18 | 96000 |

**Table 4.** Statistics of the pretraining dataset.

| Source | #Img | #Tok | #Psg | Img Size | Tok Size |
|---|---|---|---|---|---|
| Encyclopedia | 6.5M | 7.9B | 10.4M | 0.1TB | 15.0GB |
| Web pages | 46.0M | 9.1B | 106.0M | 1.5TB | 70.0GB |
| E-commerce | 8.0M | 0.5B | 8.5M | 0.3TB | 12.2GB |
| Total | 60.5M | 17.5B | 124.9M | 1.9TB | 97.2GB |

**Multi-modality model** We implement the Multi-Modality Model based on M6 (Multi-Modality-to-Multi-Modality Multitask Mega-transformer [13]) proposed by Alibaba. The input contains text and image data. As shown in Figure 9, an input image is split into patches, and its features are extracted with pretrained backbone models such as ResNet [5]. The features are then concatenated with word embeddings to form a sequence. The model consisting of transformer layers processes the sequence and generates high-level representations.

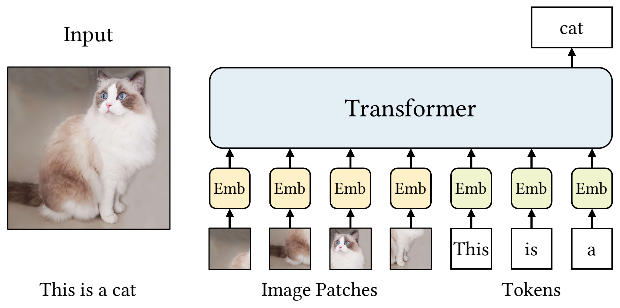

**Figure 9.** Demonstration of the model architecture.

**Dataset** The model was pretrained on the largest multi-modal dataset in Chinese, M6-Corpus [13]. Specifically, because the original dataset consists of plain text data and multi-modal data, only the latter were selected for pretraining. The data are collected from different sources, including encyclopedias, e-commerce platforms, and other crawled web pages. The detailed statistics of the final processed dataset are reported in Table 4, where "#Img" refers to the number of distinct images, "#Tok" to the number of distinct tokens,

"#Psg" to the number of passages, "Img Size" to the size of images, and "Tok Size" to the size of texts. Note that after the images transformed to features, the final product was a dataset of size 16 TB.

**Model objective** We pretrain the model with the objective of image-based language modeling, referring to next-token prediction based on the previous context, including images and text. That can be formulated as $\mathcal{L} = -\log p(x_t|x_{<t})$. As shown in Figure 9, given an image of a running dog and the text "This is a", the model is encouraged to predict the next word, "cat".

## 5.2 Evaluation Setup

The following evaluations were deployed on the *New Generation Sunway Supercomputer*.

**Timer** We instrument PyTorch to obtain the execution time for each iteration as well as the breakdown time for forward pass, backward pass, optimizer, etc.

**FLOPS** We use the performance counter provided by *Sunway* to count the performance FLOPS. Each MPE and each CPE is counted separately, and the counts are then accumulated as the result of total FLOPS.

## 5.3 Overall Performance

**Table 5.** Sustained performance

| Model | Sustained FP32 FLOPS | Sustained Mixed FLOPS |
|---|---|---|
| *MoDa-1.93T* | 647 PFLOPS | 1.180 EFLOPS |
| *MoDa-14.5T* | 525 PFLOPS | 1.002 EFLOPS |
| *MoDa-174T* | 198 PFLOPS | 230 PFLOPS |

In our experiments, we evaluate the performance and scalability of various models with different parameters on different scales. The sustained performance obtained is reported in Table 5. We evaluate the models *MoDa-1.93T*, *MoDa-14.5T*, and *MoDa-174T* for both single- and mixed-precision. *MoDa-1.93T* reached 647 PFLOPS and 1.180 EFLOPS in single- and mixed-precision, respectively. *MoDa-14.5T* achieved 525 PFLOPS and 1.002 EFLOPS in single- and mixed-precision, respectively. The *MoDa-174T* model is the largest of the proposed models, with 173.9 trillion parameters. During the training of the *MoDa-174T* model, we achieve 198 PFLOPS and 230 PFLOPS in single- and mixed-precision, respectively.

## 5.4 Intra-node Optimization

To verify the efficiency of the intra-node optimization, we compare different single-node implementations on *Sunway*: Single CG using share segment memory (SS), multiple CGs using share segment memory with 6 processes (MS), single CG using cross segment memory (SC), and multiple CGs using cross-segment memory with 1 process (MC). To fit the model into the share segment memory (SS), we choose a small 12-layer transformer model with hidden dimension of 1, 536, attention head 6, and batch size 12 for the single-CG experiments (SS and SC), while using batch size 72 for the multiple-CG experiments (MS and MC).

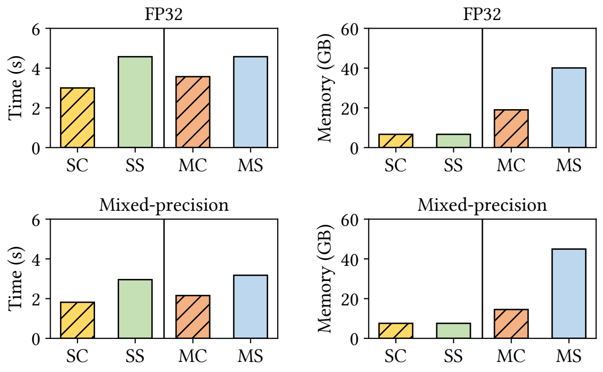

**Figure 10.** Time per iteration and peak memory usage in a single-node for different implementations. Note that the workload on each CG remains the same in each group of experiments.

Figure 10 illustrates the efficiency of intra-node optimizations, including node scheduling, memory segmenting, and memory access. With a single CG, compared with using only the share segment (SS), using the cross segment (SC) achieves 1.52× and 1.63× speedup via single- and mixed-precision, respectively. With 6 CGs, using the cross segment can effectively reduce inter-procedural communication, which leads to 1.28× and 1.47× speedup via single- and mixed-precision, respectively.

Moreover, using the cross segment can also reduce the number of shared parameters used by different processes, meaning that less memory is occupied. As shown in Figure 10, comparing the memory usage of MS and MC, using cross memory can reduce memory occupancy by 53% and 68% via single- and mixed-precision, respectively.

We evaluate the memory bandwidth with different optimization levels. The workload is set to 192 MB with a DMA granularity of 512 bytes. Figure 12 shows the result of memory bandwidth optimizations. GLD/GST means using global load and store directly. DMA means using *DMA* on each CPE. DMA+RMA represents using *DMA* together with *RMA* as mentioned in Section 4.1. MC means using DMA+RMA after rearranging *DMA* calls. The results show that the optimization effect of the proposed method is obvious. When using global load and store, the memory bandwidth could only reach 0.24 GB/s and 0.024 GB/s, respectively. *DMA* significantly expanded memory bandwidth to 45.3 GB/s and 41.9 GB/s. When using *DMA* together with *RMA*, the bandwidth could be increased to 169.1 GB/s and 162.4 GB/s due to the increase in *DMA* granularity and the decrease in the number of *DMA* calls. After adding the rearrangement for *DMA* calls, the memory bandwidth could be further boosted to 240.2 GB/s and 235.4 GB/s, respectively. This optimization makes full use of the characteristics of the cross memory and effectively uses the multi-core group to increase the memory bandwidth, which reached a bandwidth close to the theoretical limit on *Sunway*.

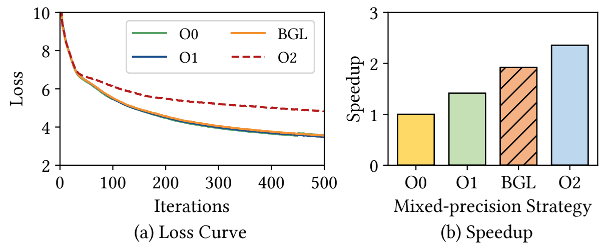

**Figure 11.** Loss curve and relative speedup of different mixed-precision approaches. O0, O1, O2 are different optimization levels of APEX, and BGL is the strategy we proposed.

## 5.5 Mixed-precision

To evaluate the proposed mixed-precision strategy, we deploy *MoDa-1.93T* on 9, 600 nodes (1/10 of the full system). We also evaluate different mixed-precision strategies on this model, for example, FP32 training (O0), APEX O1 (O1), APEX O2 (O2), and the strategy we proposed (BGL). Figure 11(a) shows the loss curve during the first 500 training iterations. It is apparent that BGL has a similar convergence trend to FP32 training and APEX O1, but that APEX O2 could not converge as fast as other strategies.

Figure 11(b) shows that, compared to FP32 training, APEX O1 and the proposed strategy can achieve 41.5% and 92.0% speedup, respectively, which illustrates that the proposed mixed-precision strategy can significantly accelerate training without degrading convergence.

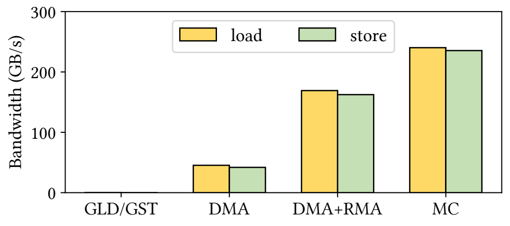

**Figure 12.** Memory access optimization breakdown.

## 5.6 MoDa

We evaluate the effectiveness of *MoDa* by comparing the *MoDa-1.16T* model with the mapping strategy mentioned in Section 4.2 (*MoDa*) and with another strategy that maps the two dimensions of communication inversely (*Inv*), i.e., *All-to-All* communication happens within a supernode and *All-Reduce* communication happens across supernodes. Both models are trained with single-precision on 240 supernodes, and each supernode contains 240 nodes.

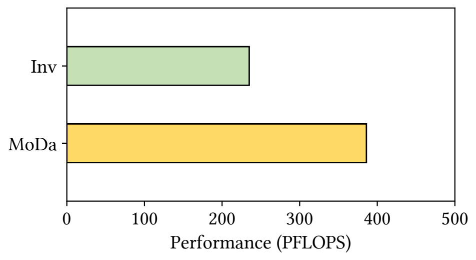

**Figure 13.** Performance of *MoDa* and *Inv*.

As shown in Figure 13, *MoDa* and *Inv* achieve 386 PFLOPS and 235 PFLOPS, respectively. This result implies that the proposed parallel strategy *MoDa* can achieve better performance than the straightforward approach that maps *All-to-All* communication to intra-supernode transmission and *All-Reduce* communication to inter-supernode transmission.

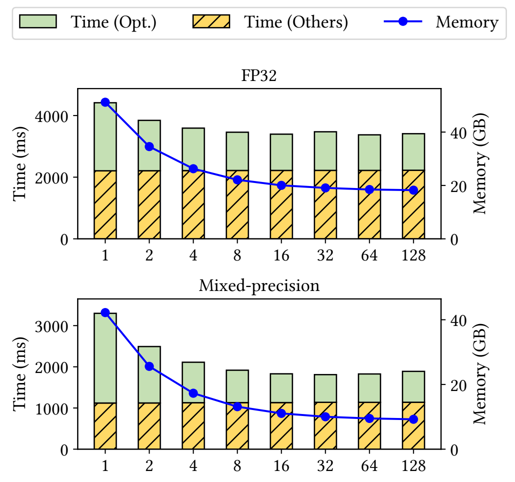

**Figure 14.** Time per iteration and peak memory usage at different *data parallelism* scales.

## 5.7 ParO

We compare the memory occupancy and gradient updating time using *ParO* to demonstrate the efficiency of *ParO*. The evaluation is performed using 1, 2, 4, 8, 16, 32, 64, and 128 nodes. The model we used for this experiment is a 5-layer transformer with hidden dimension of 6, 144 and attention head 6.

We test the performance and memory consumption using the same model at different *data parallelism* scales to evaluate the efficiency of *ParO*, as shown in Figure 14. As the number of nodes grows, the forward and backward computation cost and communication cost remains constant, while the optimizing computation spreads over nodes. Therefore optimization time consumption is notably reduced. Alongside the reduced optimizing time consumption, memory consumption is also significantly reduced, confirming our expectations of *ParO*.

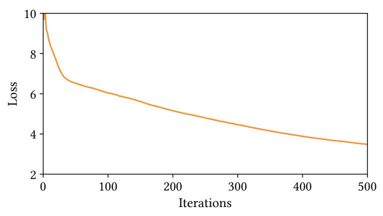

**Figure 15.** Loss curve

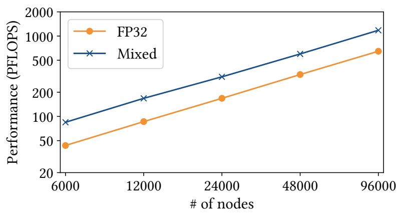

**Figure 16.** Weak scalability

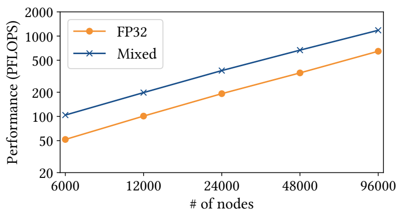

**Figure 17.** Strong scalability

## 5.8 Loss Curve

Due to budgetary concerns, we use 1/2 of the entire system to obtain the loss curve for *MoDa-1.93T*. To maintain the equivalence of training, we train *MoDa-1.93T* using the same *global batch size* of the model running on the entire system by means of a two-step gradient accumulation. Therefore, we can obtain the exact same training process on 1/2 machine with the training process running on the entire system since we have the same batch size and the same input data.

We validate BaGuaLu by inspecting the training loss of the *MoDa-1.93T*. The loss curve is illustrated in Figure 15. The training loss of *MoDa-1.93T* model decreases to 3.46 after 500 iterations. According to [13], the training loss is near convergence, and it demonstrates low perplexity.

## 5.9 Scalability

We examine both the weak and strong scalability of BaGuaLu. The scalability experiments are scaled from 6,000 nodes (1/16 of the entire system) to 96,000 nodes (the entire system). We define the *problem size* as *both the model size and the training data size*. With the same problem size, we evaluate the time consumed for iterating over the same size of data and updating the same model accordingly.

*For weak scalability*, we scale BaGuaLu to train *MoDa-1.93T*/16, *MoDa-1.93T*/8, *MoDa-1.93T*/4, *MoDa-1.93T*/2, and *MoDa-1.93T* on 1/16, 1/8, 1/4, 1/2, and the full system accordingly. The *MoDa-1.93T*/$x$ model simply stands for the model with $1/x$ of the experts in *MoDa-1.93T*. Fixing the micro batch size, the model size, and the global batch size both grow with the system scale.

*For strong scalability*, we fix both the number of experts and global batch size, and measure the time of one step in training the model. We choose the minimum model that could be trained properly at full scale and fix the micro batch size. For example, on 48,000 nodes (1/2 system), each node consumes 2 micro batches with 2× data in each iteration and update once; on 24,000 nodes (1/4 system), each node consumes 4 micro batches each iteration and update once.

For weak scalability, as shown in Figure 16, when scaling from 1/16 system (6,000 nodes) to full system (96,000 nodes), given balanced computation workload being identical between scales, we achieve nearly linear weak scalability.

For strong scalability, as shown in Figure 17, we observe that with the setup scales from 1/16 system to full system, the speedup is 12.5× and 11.3× for single- and mixed-precision, respectively. The computation workload and cost remains constant across scales, while the communication cost increases. With the greater number of processes associated with MoE *All-to-all* communication, the communication granularity is lower, causing higher overhead. Meanwhile, the reduction of gradients in optimizing the model requires more communication traffic. Therefore, linear scalability is not accomplished.

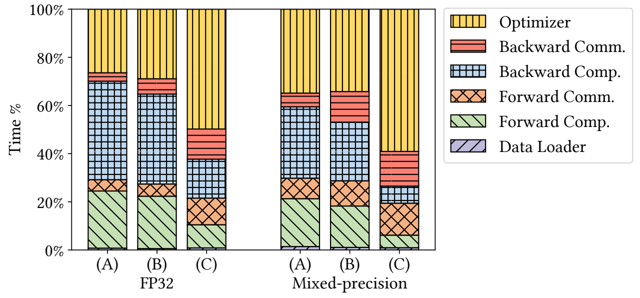

**Figure 18.** Performance breakdown. (A), (B), and (C) denote *MoDa*-1.93*T*, *MoDa*-14.5*T*, and *MoDa*-174*T*, respectively. FP32 and Mixed denote single- and mixed-precision training.

## 5.10 Performance Breakdown

To better understand the application performance, we measure the performance breakdown of BaGuaLu. We use *MoDa-1.93T*, *MoDa-14.5T*, and *MoDa-174T* on the full system in this experiment.

We profile the training of *MoDa-1.93T*, *MoDa-14.5T*, and *MoDa-174T*. The result time breakdown is shown in Figure 18. We observe that the *All-to-All* communication between experts becomes the communication bottleneck when the model scales to 14.5 trillion parameters: the forward and backward communication cost extends significantly. When we scale the model to 174 trillion, with the *data parallelism* group not presented, the optimizer step dominates the training cost even if *All-to-All* performs worse given a larger MoE group size. This is because *ParO* is not available, causing the optimizer to work on all parameters, which leads to notably lower performance than other models.

## 6 Discussions

BaGuaLu is the first application of training up to 174-trillion parameter models on leading supercomputers. The implications of BaGuaLu have several aspects.

The work on *MoDa* has exposed the inefficiency of the current state of the interconnection of HPC systems for the latest pretrained models. Although the oversubscribed network topology reduced the effort required to build a large-scale system and is a good fit for many traditional HPC applications, MoE requires massive *All-to-All* communications that break the optimal process mapping for neighbor communication. New advanced interconnect techniques are needed to address AI training workloads. We expect that our exploration of the communication patterns of training models at such a scale can benefit future studies in both AI and HPC.

Furthermore, our work also has many empirical implications for both model and system design for training extremely large models. We are the first to investigate mixed-precision training in brain scale pretrained models. We also explore the use of large-batch training in optimization. In general, our practical experience in brain scale pretraining sheds light on AI model training and demonstrates a successful co-design of model and system. Our large scale multi-modal pretraining is expected to have broad and far-reaching impacts on various applications, including, but not limited to, image/video captioning, image/video generation, cross-modal retrieval, visual question answering, visual commonsense reasoning, object referral, multi-modal dialog system, and multi-modal translation.

Beyond applications in computer vision and natural language processing, the paradigm of MoE parallelization and large-batch training brings new research questions to the computer science community. Although these techniques are key contributors to the proposed large-scale model, their underlying mechanism remains undiscovered from the perspective of machine learning and optimization. In the longer term, our practical experience in training large-scale models can be further adapted to other domains such as biology [24, 27] and chemistry [26].

## 7 Conclusion

This paper proposed BaGuaLu, the first work targeting training brain scale models on an entire exascale supercomputer. This is an unprecedented demonstration of algorithm and system co-design on the convergence of AI and HPC. BaGuaLu enables decent performance and scalability on extremely large models by combining hardware-specific optimizations, hybrid parallel strategies, and mixed-precision training. Evaluation shows that BaGuaLu can train 14.5-trillion-parameter models with a performance of over 1 EFLOPS, and has the capability of training brain scale models with up to 174 trillion parameters.

## Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments. This work is supported by the National Key R&D Program of China under Grant 2021ZD0110202, the National Natural Science Foundation of China (U20A20226), and the NSFC for Distinguished Young Scholar (61825602). Wenguang Chen, Jidong Zhai, Xin Liu, Hongxia Yang, and Jie Tang are the corresponding authors of this paper.

## References

[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL]

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[3] William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. arXiv:2101.03961 [cs.LG]

[4] Haohuan Fu, Junfeng Liao, Jinzhe Yang, Lanning Wang, Zhenya Song, Xiaomeng Huang, Chao Yang, Wei Xue, Fangfang Liu, Fangli Qiao, et al. 2016. The Sunway TaihuLight supercomputer: system and applications. *Science China Information Sciences* 59, 7 (2016), 1–16.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity Mappings in Deep Residual Networks. In *ECCV 2016*. 630–645.

[6] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Xu Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. 2019. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 103–112.

[7] Thorsten Kurth, Sean Treichler, Joshua Romero, Mayur Mudigonda, Nathan Luehr, Everett Phillips, Ankur Mahesh, Michael Matheson, Jack Deslippe, Massimiliano Fatica, et al. 2018. Exascale deep learning for climate analytics. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 649–660.

[8] Thorsten Kurth, Jian Zhang, Nadathur Satish, Evan Racah, Ioannis Mitliagkas, Md. Mostofa Ali Patwary, Tareq Malas, Narayanan Sundaram, Wahid Bhimji, Mikhail Smorkalov, Jack Deslippe, Mikhail Shiryaev, Srinivas Sridharan, Prabhat, and Pradeep Dubey. 2017. Deep Learning at 15PF: Supervised and Semi-Supervised Classification for Scientific Data. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '17)*. Association for Computing Machinery, New York, NY, USA, Article 7, 11 pages. https://doi.org/10.1145/3126908.3126916

[9] Nouamane Laanait, Joshua Romero, Junqi Yin, M Todd Young, Sean Treichler, Vitalii Starchenko, Albina Borisevich, Alex Sergeev, and Michael Matheson. 2019. Exascale deep learning for scientific inverse problems. *arXiv preprint arXiv:1909.11150* (2019).

[10] Charles E Leiserson. 1985. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE transactions on Computers* 100, 10 (1985), 892–901.

[11] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668* (2020).

[12] Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. 2021. BASE Layers: Simplifying Training of Large, Sparse Models. arXiv:2103.16716 [cs.CL]

[13] Junyang Lin, Rui Men, An Yang, Chang Zhou, Ming Ding, Yichang Zhang, Peng Wang, Ang Wang, Le Jiang, Xianyan Jia, Jie Zhang, Jianwei Zhang, Xu Zou, Zhikang Li, Xiaodong Deng, Jie Liu, Jinbao Xue, Huiling Zhou, Jianxin Ma, Jin Yu, Yong Li, Wei Lin, Jingren Zhou, Jie Tang, and Hongxia Yang. 2021. M6: A Chinese Multimodal Pretrainer. *CoRR* abs/2103.00823 (2021).

[14] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint arXiv:1907.11692* (2019).

[15] Ilya Loshchilov and Frank Hutter. 2017. Fixing Weight Decay Regularization in Adam. *CoRR* abs/1711.05101 (2017).

[16] NVIDIA. 2021. Apex (A PyTorch Extension). https://nvidia.github.io/apex/

[17] Robert M Patton, J Travis Johnston, Steven R Young, Catherine D Schuman, Don D March, Thomas E Potok, Derek C Rose, Seung-Hwan Lim, Thomas P Karnowski, Maxim A Ziatdinov, et al. 2018. 167-pflops deep learning for electron microscopy: from learning physics to atomic manipulation. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 638–648.

[18] Robert M Patton, J Travis Johnston, Steven R Young, Catherine D Schuman, Thomas E Potok, Derek C Rose, Seung-Hwan Lim, Junghoon Chae, Le Hou, Shahira Abousamra, et al. 2019. Exascale deep learning to accelerate cancer research. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 1488–1496.

[19] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *CoRR* abs/1802.05365 (2018). arXiv:1802.05365 http://arxiv.org/abs/1802.05365

[20] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. (2018).

[21] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. (2019).

[22] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21 (2020), 1–67.

[23] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: Memory Optimizations toward Training Trillion Parameter Models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Atlanta, Georgia) *(SC '20)*. IEEE Press, Article 20, 16 pages.

[24] Roshan Rao, Jason Liu, Robert Verkuil, Joshua Meier, John F Canny, Pieter Abbeel, Tom Sercu, and Alexander Rives. 2021. Msa transformer. *bioRxiv* (2021).

[25] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3505–3506.

[26] Philippe Schwaller, Teodoro Laino, Théophile Gaudin, Peter Bolgar, Christopher A Hunter, Costas Bekas, and Alpha A Lee. 2019. Molecular transformer: a model for uncertainty-calibrated chemical reaction prediction. *ACS central science* 5, 9 (2019), 1572–1583.

[27] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. 2020. Improved protein structure prediction using potentials from deep learning. *Nature* 577, 7792 (2020), 706–710.

[28] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. 2018. Mesh-tensorflow: Deep learning for supercomputers. *arXiv preprint arXiv:1811.02084* (2018).

[29] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538* (2017).

[30] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053* (2019).

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[32] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *arXiv preprint arXiv:1906.08237* (2019).

[33] Ruizhe Zhao, Brian Vogel, and Tanvir Ahmed. 2019. Adaptive Loss Scaling for Mixed Precision Training. *CoRR* abs/1910.12385 (2019). arXiv:1910.12385 http://arxiv.org/abs/1910.12385