SC24: International Conference for High Performance Computing, Networking, Storage and Analysis | 979-8-3503-5291-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/SC41406.2024.00089

Fire-Flyer AI-HPC: A Cost-Effective Software-Hardware Co-Design for Deep Learning

Wei An, Xiao Bi, Guanting Chen, Shanhuang Chen,Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Wenjun Gao, Kang Guan, Jianzhong Guo, Yongqiang Guo, Zhe Fu, Ying He, Panpan Huang, Jiashi Li, Wenfeng Liang, Xiaodong Liu, Xin Liu, Yiyuan Liu, Yuxuan Liu, Shanghao Lu, Xuan Lu, Xiaotao Nie, Tian Pei, Junjie Qiu, Hui Qu, Zehui Ren, Zhangli Sha, Xuecheng Su, Xiaowen Sun, Yixuan Tan, Minghui Tang, Shiyu Wang, Yaohui Wang, Yongji Wang, Ziwei Xie, Yiliang Xiong, Yanhong Xu, Shengfeng Ye, Shuiping Yu, Yukun Zha, Liyue Zhang*, Haowei Zhang, Mingchuan Zhang, Wentao Zhang, Yichao Zhang, Chenggang Zhao, Yao Zhao, Shangyan Zhou, Shunfeng Zhou, Yuheng Zou

DeepSeek-AI, Beijing, China
research@deepseek.com

***Abstract*—The rapid progress in Deep Learning (DL) and Large Language Models (LLMs) has exponentially increased demands of computational power and bandwidth. This, combined with the high costs of faster computing chips and interconnects, has significantly inflated High Performance Computing (HPC) construction costs. To address these challenges, we introduce the Fire-Flyer AI-HPC architecture, a synergistic hardware-software co-design framework and its best practices. For DL training, we deployed the Fire-Flyer 2 with 10,000 PCIe A100 GPUs, achieved performance approximating the DGX-A100 while reducing costs by half and energy consumption by 40%. We specifically engineered HFReduce to accelerate allreduce communication and implemented numerous measures to keep our Computation-Storage Integrated Network congestion-free. Through our software stack, including HaiScale, 3FS, and HAI-Platform, we achieved substantial scalability by overlapping computation and communication. Our system-oriented experience from DL training provides valuable insights to drive future advancements in AI-HPC.**

***Index Terms*—High Performance Computing, Cost-Effective, All-Reduce, Best Practices, Deep Learning, Machine Learning, Large Language Models, Artificial Intelligence Infrastructure**

I. Introduction

In recent years, Deep Learning (DL) [1] has developed rapidly and is widely used in image recognition, speech recognition, content generation, autonomous driving, and other areas. The rapid development of DL is fundamentally tied to the support rendered by data. Training with copious amounts of data demands massive computational resources. Relying on Moore's Law [2], computer speeds double every two years on average, but the pace of DL far exceeds this speed. In particular, Large Language Models (LLMs) [3]–[7] that have become popular in recent years have exploded the demand for computational resources and memory. The parameters of LLMs can reach tens to thousands of billions, requiring hundreds or thousands of GPUs for training. Although LLM training is challenging, the emerging capabilities resulting from more parameters have shown the benefits of continued model expansion. Since then, researchers have gone down the path of making models bigger and never looked back. To acquire more computational resources, people have to expand more nodes. This leads to a surge in the cost of building AI infrastructure. How to reduce the cost of new data centers and how to build cost-effective clusters are also hot and challenging problems. Moreover, more nodes lead to higher energy consumption, which contradicts this era's goal of reducing carbon emissions and achieving carbon neutrality. Reducing energy consumption is also a challenging problem.

In this paper, leverage our practical experience accumulated over the years to propose cost-effective strategies cost-effective strategies for constructing AI-HPC systems suitable for deep learning and LLMs.

***Fire-Flyer AI-HPC Architecture**:* We have deployed a cluster composed of 10,000 PCIe A100 GPUs for Deep Learning training purposes. Details about the GPU nodes and network topology are provided in Section III-A, where we compare our architecture to the NVIDIA DGX-A100 [8] in terms of cost-effectiveness and lower CO_2 emissions. In contrast, we must invest more in software optimizations to address the performance challenges of the PCIe architecture. The following sections will discuss about software-hardware co-design.

Key Technical Topics in our Architecture

- **Network Co-Design**: The Two-Layer Fat-Tree Network [9] integrates storage and computation network, as shown in Section III-B. The entire network is divided into two zones, and the platform supports cross-zone tasks. To prevent congestion, we employed various network tunings detailed in Section VI-A.
- **HFReduce**: Achieves computation-communication overlap via asynchronous allreduce on the CPU, outperforming NVIDIA Collective Communications Library (NCCL) [10] on our PCIe architecture, as discussed in Section IV.

Authors are listed in alphabetical order of their surnames.

SC24, November 17-22, 2024, Atlanta, Georgia, USA
979-8-3503-5291-7/24/$31.00 ©2024 IEEE

- **HaiScale**: As described in Section V, optimizes parallelism methods for our PCIe architecture, such as Data Parallelism (DP), Pipeline Parallelism (PP) [11] [12], Tensor Parallelism (TP) [13] [14], Experts Parallelism (EP) [15]–[17], Fully Sharded Data Parallel(FSDP) [18] and Zero Redundancy Optimizer(ZeRO) [19].
- **3FS Distributed File System**: Addresses I/O bottlenecks in big data AI tasks, configures with our communication and network tuning, reduces congestion in storage and computation integrated network topology, as detailed in Section VI-B.
- **HAI Platform** [20]: Offers task scheduling, fault handling, and disaster recovery, enhancing utilization and reducing costs. It provides an open-box solution for deep-learning researchers. It is already open-sourced: https://github.com/HFAiLab/hai-platform

Stability and Robustness: These are crucial topics in HPC. Our systems are equipped with robust mechanisms to handle hardware failures, minimizing downtime and impact on operations. These mechanisms, discussed in Section VII, include:

- Disaster recovery through our checkpoint manager
- A validator utility for detecting hardware failures
- An overview of real hardware failure data from our cluster over the past year.

We hope these insights will be beneficial to industry peers and researchers alike.

Discussion and Future Work: In Section VIII, we address some common questions regarding PCIe architecture, such as congestion control, maintenance cost, and stability compared with other architectures. In Section IX, we propose the next generation of PCIe architecture, which is aimed at Mixture-of-Experts Large Language Models training and primarily utilizes multi-NICs and a Multi-Plane network.

II. Background

A. Evolution of Deep Learning

The revolution in Machine Learning and Deep Learning began in 2012 with AlexNet [21], which outperformed traditional methods in image classification, marking the onset of big data utilization and increased computational demands. The emergence of ResNet [22], with its deeper layers, further broadened the horizons of image processing, truly bringing about the "deep" in "Deep Learning". At the same time, big data-driven model training nudged the evolution of data storage technology, leading to the advent of all-flash SSD distributed file systems.

Fast forward to 2017, Google's Transformer [23] made its grand entry, introducing the concept of "Attention is all you need", shaking up the field of Natural Language Processing (NLP). With the advent of more complex models like AlphaFold [24] and AlphaZero [25] highlighting the need for more computational power and memory, revealing the limitations of traditional FP64 / FP32 computing devices.

Entering the 2020s saw the rise of LLMs as a game-changer in the AI sector. Research indicates that an upscale in the number of language model parameters and computational budget

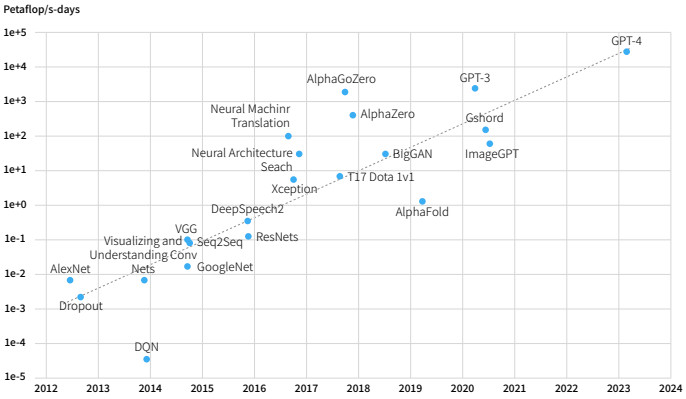


Figure 1: The Exponential Growth of Computational Power for Deep Learning.

can significantly enhance model performance, given adequate training data. Consequently, despite requiring colossal computational resources, efforts are being made to train large models on tens or hundreds of billions or even trillions of parameters. Pioneering examples include GPT-3 [26] and PaLM [27], which occupy close to 1TB of GPU Memory. Recognizing the potential, industry giants have set up large AI clusters to train LLMs while constantly investing in computational power chips.

The shift towards the Mixture-of-Experts (MoE) Models [28]–[30] architecture starting from GPT-4 [7], and the recent AI Generated Content (AIGC) multi-modal (Sora [31]) has amplified the demand for memory and computational resources. However, as AI development outpaces hardware development, leading to skyrocketing training costs, adopting cost-saving solutions has become imperative.

Figure 1 illustrates the exponential growth of computational power for DL. And as summarized in Figure 2 [32], while AI's demand for computational power is growing at 10x per year, Moore's Law lags behind with hardware FLOPs increase at only 3.0x every two years, DRAM bandwidth at 1.6x, and interconnect bandwidth at 1.4x. This disparity necessitates more machines, raising DL training costs, particularly for LLM training, where the computational power required surpasses that of traditional HPC applications.

B. Challenges and Solutions in Models Training

In Deep Learning training, a single task demands hundreds of GPUs and consumes substantial storage and network resources. This massive scale introduces system-level challenges:

1) Efficiency: Firstly, achieving efficient training at this magnitude is crucial. Model FLOPs Utilization (MFU), which assesses the ratio of observed throughput to theoretical maximum throughput (assuming 100% peak FLOPS), serves as the standard metric for evaluating training efficiency. Training LLMs involves dividing models among GPUs that communicate extensively for progress. Besides communication, factors like operation optimization, data pre-processing, and GPU memory consumption significantly influence MFU. Multiple parallel strategies are employed to enhance efficiency:

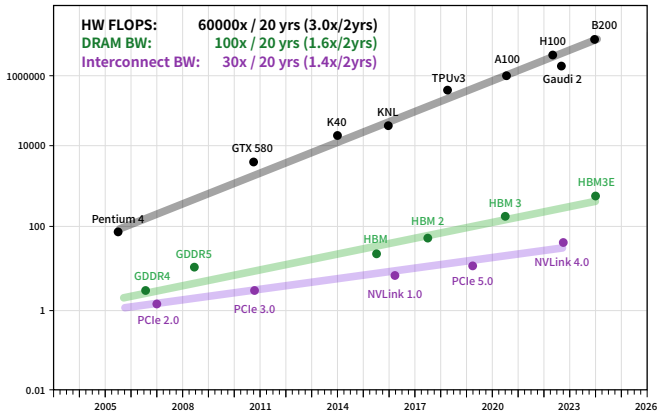


Figure 2: Scaling of Peak Hardware FLOPS, and Memory/Interconnect Bandwidth.

- **Data Parallelism (DP):** Models and optimizer states are replicated across multiple devices with data evenly distributed to all. For LLMs training, the Zero Redundancy Optimizer (ZeRO) [19] further enhances this method by sharding these states on each data parallel process and using allgather and reduce-scatter for parameter fetching and gradient calculation.
- **Pipeline Parallelism (PP):** Each device holds a portion of the model layers with each training batch divided into micro-batches for pipeline execution. Efficient scheduling strategies like GPipe [11], PipeDream 1F1B [12], and Zero Bubble Pipeline Parallelism (ZBPP) [33], are required to minimize "pipeline bubbles".
- **Tensor Parallelism (TP):** This involves placing a model layer on multiple GPUs that perform computations in parallel [13] [14]. It includes row-wise and column-wise parallelism, necessitating allgather and all2all for input splitting and output merging.
- **Expert Parallelism (EP)**: MoE Models' different expert models are distributed on different GPUs during MoE training [15]–[17]. The gate model selects tokens for allocation during input, with corresponding tokens sent to experts model via all2all communication.
- **Fully Sharded Data Parallel (FSDP)** is an implementation based on the ZeRO Stage 3 algorithm [19]. FSDP partitions the model's parameters, optimizer states, and gradients, distributing them across different GPUs, with each GPU retaining only $1/n$ of the total. During forward propagation, FSDP performs an allgather operation to assemble the complete parameters, which are then released after the forward pass is completed. Similarly, during backward propagation, FSDP conducts an allgather operation to obtain the complete parameters, followed by backward computation to calculate gradients. It then performs a reduce-scatter operation to synchronize gradients across all GPUs, resulting in each GPU holding $1/n$ of the reduced gradients. Finally, FSDP updates the $1/n$ parameters using each GPU's $1/n$ gradients and optimizer states. FSDP reduces GPU memory usage by maintaining only $1/n$ of the parameters, gradients, and optimizer states on each GPU, enabling training of larger-scale models.

There are additional strategies and algorithms to accelerate training or reduce memory usage, such as **Activation Recomputation** [34], [35], as well as enhanced communication and computation overlap during parallelism [36], among others.

2) Stability: The second challenge is achieving high-stability training at scale, i.e., maintaining efficient training throughout the process. Stability is vital from a production standpoint as training a big model with a trillion tokens may span several weeks. In DL training, stragglers and hardware failures are common occurrences rather than outliers. Stragglers can decelerate tasks involving hundreds of GPUs, emphasizing the importance of stability and task recovery time.

C. HPC and AI Clusters of This Era

1) HPC Inadequacies for AI Training: Traditional supercomputers such as TianHe-2A [37], Stampede 2 [38], and Sunway TaihuLight [39] primarily focus on double precision calculations and do not support the FP16 precision, rendering them unsuitable for DL training. Fugaku [40], despite its high performance, does not support tensor GEMM acceleration, a key component for DL workloads. Although these supercomputers may not be well-suited for DL training, their robust high-performance networks and extensive experience in large-scale cluster construction offer valuable insights and lessons for subsequent researchers.

2) GPU based HPC: Supercomputers like Frontier [41], Aurora [42], Summit [43] and Perlmutter [44] utilize high-performance GPUs to tackle large-scale computations. It's worth mentioning that Perlmutter utilizes an all-flash storage system, achieving a peak bandwidth of 5TB/s. Indeed, conducting DL training on these GPU-based HPCs yields significant performance.

3) GPU Clusters of Large Companies: Meta, formerly Facebook, has developed its AI-HPC using a software-hardware co-design approach, with one system utilizing IB and another employing RoCE [45], [46]. ByteDance initially implemented a DL cluster with a mix of CPU and PCIe GPU [47]. However, with the advent of the LLMs era, they adopted an architecture similar to DGX, building a cluster with over 10,000 GPUs [48]. Alibaba has developed its own HPN network [49] for LLMs training using NVIDIA H800 GPUs. NVIDIA also has its own AI-HPC Eos [50], which will feature 576 DGX H100 systems totaling 4,608 H100 GPUs. While this will provide a considerable boost in computational power for AI tasks, the high cost of DGX systems raises questions about economic viability.

4) AI DSA Clusters: Custom-designed AI DSA (Domain Specific Architecture) accelerators like Google's TPU [51] , utilize highly advanced optical switch reconfigurable networks. Alternatives to traditional GPU setups, like Intel Habana Gaudi [52], are also available. Teslahas introduced the Dojo [53], [54] supercomputer , which uses System on

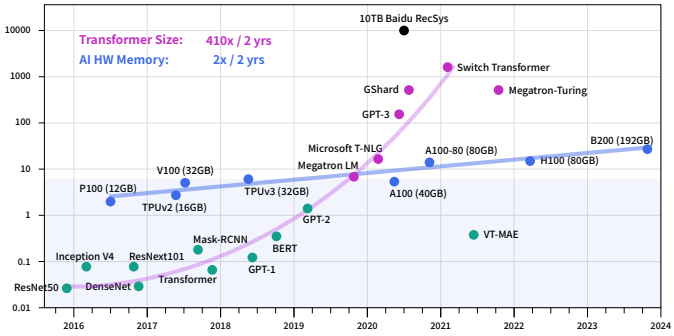


Figure 3: Size of Model Parameter and Accelerator Memory

Wafer technology to build an entire silicon wafer as a single chip. Huawei has designed the Ascend AI DSA chip [55], [56], which remains competitive with NVIDIA, as noted by NVIDIA CEO Jensen Huang. These accelerators are tailored for efficient execution of AI workloads, offering specialized features to optimize model training and inference. However, their software ecosystems, while progressing, still require further development to match the maturity of NVIDIA's offerings.

5) Cloud Service Providers: Cloud service providers, such as Azure, offer flexible and scalable resources for AI training. Despite their convenience and easy accessibility, the costs can accumulate significantly over time. For long-term projects spanning around two years, these costs could amount to purchasing an entire dedicated cluster. Therefore, this option may not be the most economical choice for extensive AI computations.

D. Challenges in AI Infrastructure

As models continue to grow larger, DL training requires thousands of GPUs. Additionally, researchers often need to train multiple models simultaneously. Therefore, a cluster with at least tens of thousands of GPUs can meet the needs of AI practitioners. In addition to increasing the scale of the cluster nodes and adding more GPUs, there's also a need to find ways to save on the overall system construction costs. These costs include but are not limited to power support, cooling, networking, storage, fault handling, disaster recovery, etc. Building a cost-effective AI-HPC system is a significant challenge. The question of how to construct a high-performance, efficient, economical, and environmentally friendly HPC to meet AI training requirements has become a hot topic. Some works, such as [57] analyzes the construction of HPC, discussed the interconnection of heterogeneous clusters, cooling systems. AI applications such as Peng Cheng Cloud Brain II [58] discussed their strategies for building and improving AI computing power and cluster communication efficiency, which helped them achieve first place in IO500 and AIPerf rankings.

Drawing from our extensive experience in Deep Learning spanning over the past decade, we have conducted considerable exploration in terms of cost-effectiveness. This work primarily discusses our practices for achieving cost-effectiveness and high-performance across different models and stages.

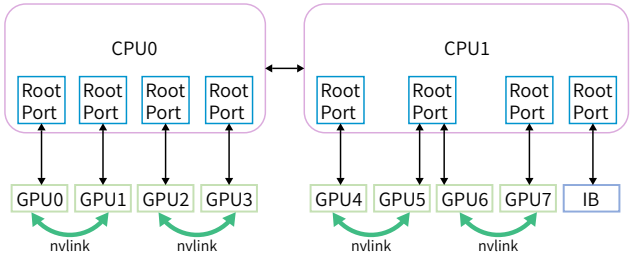


Figure 4: In-node Architecture: 8 PCIe GPUs and 1 InfiniBand (IB) NIC are directly connected to the CPU. Note that GPU5/6 share the same PCIe root port, while IB occupies one independently.

Table I: Our Arch and DGX-A100 Server Hardware Details

	Our PCIe Arch	DGX-A100
CPU	2 * AMD 32 Cores EPYC Rome/Milan CPU	2 * AMD 64 Cores EPYC 7742 CPU
Memory	512GB 16-Channels DDR4-3200Mhz	2048GB 16-Channels DDR4-3200Mhz
GPU	8 * PCIe-A100-40GB	8 * SXM-A100-40GB
NICs	1 * Mellanox InfiniBand cx6 200Gbps NIC	9 * Mellanox InfiniBand cx6 200Gbps NIC
NVLINK	600 GB/s between each pair of GPUs	600 GB/s interconnect among all 8 GPUs

III. Fire-Flyer 2: Our Approach For Deep Learning and Early LLM Training

As mentioned in the Background Section, LLMs generally require significant memory resources. In contrast, many other models necessitate considerably less memory, as illustrated in Figure 3. Popular models like ResNet [22], Mask-RCNN [59], BERT [60], MAE [61], among others, all have a parameter volume less than 1B, signifying relatively low memory requirements. Therefore, when designing a cluster primarily for deep learning model training, and with insights gleaned from our Fire-Flyer 1 experiments, we deemed it prudent to incorporate PCIe A100 GPUs, which proved to be sufficient during its construction in 2021.

A. Fire-Flyer 2: PCIe A100 GPU Architecture

In our training workloads, the bandwidth requirements for both storage IOs and computation communication across 8 NVIDIA PCIe A100 GPUs can be met by a single 200Gbps NVIDIA Mellanox ConnectX-6 (CX6) InfiniBand (IB) NIC. We employed the following computation node architecture , as shown in Figure 4:

- 8 NVIDIA A100 PCIe GPUs and 1 Mellanox CX6 200Gbps IB NIC: directly connect to the CPU, without using a PCIe switch
- IB NIC occupies a separate PCIe root complex, thus avoiding performance interference with the GPU.
- Reserved the possibility of NVLink Bridge addition in design: As expected, when the LLM era arrived, we indeed added an NVLink Bridge between PCIe cards.

Table I shows our arch details and compared with NVIDIA standard DGX-A100 server.

B. Network Topology: Two-Layer Fat-Tree with Storage and Computation Integrated

We selected the **Fat-Tree** [9] topology as our primary network architecture due to its exceptionally high bisection bandwidth, making it the preferred choice for AI-HPC and high-throughput storage environments. Although the Dragonfly topology [62], [63] also offers comparable cost-effectiveness and performance, its lack of sufficient bisection bandwidth makes it unsuitable for our **integrated storage and computation network design**. At the time of implementation, various RoCE (RDMA over Converged Ethernet) [64] technologies were not as mature as they are today, so we opted for InfiniBand (IB) as our network solution. Mellanox QM8700 InfiniBand Switch, offering 40 ports at 200 Gbps, was utilized. Our cluster, consisting of 10,000 A100 GPUs, includes approximately 1,250 GPU compute nodes and nearly 200 storage servers, although a Two-Layer Fat-Tree can accommodate up to 800 nodes (configured with 20 spine switches and 40 leaf switches).

To reduce costs, , we opted for a two-zone network configuration instead of a three-layer Fat-Tree solution, as shown in Figure 5. Each zone consists of an 800-port Fat-Tree connected to approximately 600 GPU compute nodes. Each storage server equipped with two IB NICs, respectively connected to different zones, hence all GPU compute nodes could share a set of storage services. Additionally, the two zones are interconnected with a limited number of links. Our HAI Platform scheduling strategy ensured that cross-zone computing tasks were limited to one at most. Whether using NCCL [10] or our in-house developed communication library HFReduce, it can be run across zones by using a double binary tree algorithm [65]. Our scheduler ensures that in this topology, only one pair of nodes communicates across zones. Consequently, even tasks requiring all nodes can efficiently run on the entire Fire-Flyer 2 AI-HPC.

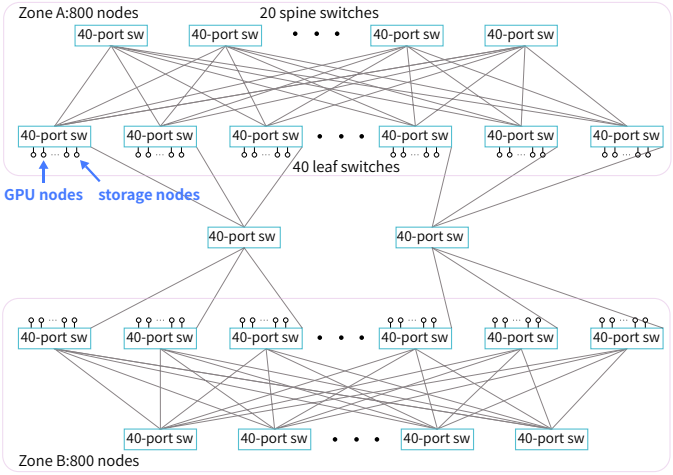


Figure 5: Network Topology: Two complete Two-Layer Fat-Tree connected together.

Table II: A100 PCIe Compared to DGX-A100.

	Our Arch	DGX Arch
TF32 GEMM (TFLOPS/GPU)	107	131
FP16 GEMM (TFLOPS/GPU)	220	263
Relative Performance	83%	100%
Node Relative Price	60%	100%
Cost–Performance Ratio	1.38	1
Power Consumption (Watts)	2500	4200

Table III: Relative Cost Comparison.

	Our Arch	PCIe Arch with Three Layer Fat-Tree	DGX Arch
Number of Switches	122	200	1320
Network Price	350	600	4000
Server Price	11250	11250	19000
Total Price	11600	11850	23000

C. Cost Performance of Our Architecture

Compared to the NVIDIA DGX-A100 [8] architecture, our approach using PCIe A100 achieves approximately 83% of the performance in TF32 and FP16 General Matrix Multiply (GEMM) benchmarks. However, it offers substantial reductions in both costs and energy usage, achieving 60% of the GPU cost and energy consumption, as detailed in Table II.

Contrasting with the DGX-A100 cluster, which necessitates a Three-Layer Fat-Tree encompassing 10,000 access points and involving 320 core switches, 500 spine switches and 500 leaf switches, amounting to 1,320 switches in total (as shown in Table III), our architecture only requires 122 switches. This arrangement is significantly more cost-efficient. Even when compared to a similarly sized three-layer Fat-Tree network with 1,600 access points, which includes 40 core switches and 160 spine and leaf switches (totaling 200 switches), our design facilitates a saving of 40% in networking costs.

Furthermore, by utilizing an 800-Ports Frame Switch, we have further reduced the cost of optical modules and cables. While there is a performance gap due to the inherent differences between PCIe card specifications and SXM, we generally achieved 80% of the DGX-A100 performance at merely 60% of the cost. Additionally, we managed to trim energy consumption by 40%, thereby reducing CO_2 emissions. In terms of cost-performance, we regard this approach as both effective and successful.

IV. HFReduce: Hardware Software Co-Design in Network

In large-scale deep learning training, allreduce is essential for aggregating gradients across GPUs. To optimize communication among PCIe GPUs in our architecture, we developed HFReduce, a library specifically designed for efficient allreduce operations. The core strategy of HFReduce, illustrated in Figure 6, involves performing intra-node reduction first, followed by inter-node allreduce of the reduced data from the 8

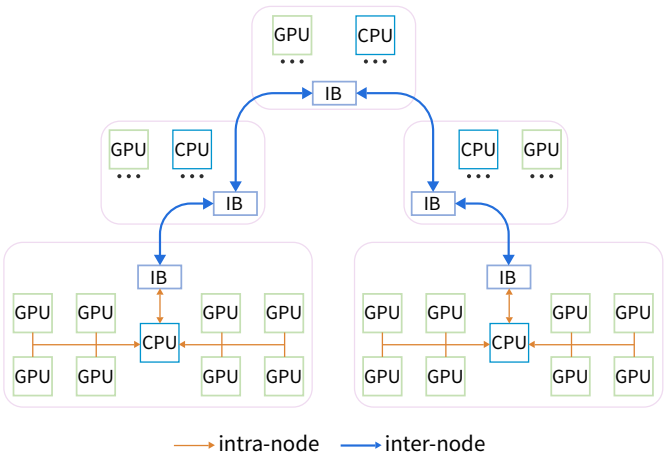


Figure 6: HFReduce Schematic: 1) do intra-node reduce, 2) do inter-node allreduce by CPU, 3) finally transfer reduced data to GPU.

GPUs within each node. This inter-node allreduce leverages a Double Binary Tree Algorithm [65], akin to NCCL, and is pipelined by dividing data into chunks for transfer via Remote Direct Memory Access (RDMA), ensuring high performance. HFReduce is versatile and can be applied to any scenario requiring allreduce, as well as general reduce and broadcast operations.

A. HFReduce Algorithm Steps

Intra-node reduction, as shown in the Algorithm 1:

1) When the gradients data on the GPUs require allreduce, HFReduce asynchronously transfers these data to the CPU memory. This Device-To-Host (D2H) Transfer can utilize GDRCopy [66] for small data and MemCpyAsync for larger data.
2) Upon the arrival of the gradients in memory, perform reduction add operation using CPU vector instructions.

Inter-node reduction, as shown in the Algorithm 2:

1) Use the Double Binary Tree Algorithm [65] for inter-node allreduce, facilitating transfers between nodes using RDMA verbs implementation.
2) Finally, the CPU returns reduced gradients to the GPU via PCIe (Host-To-Device Phase).

The final Host-To-Device (H2D) Transfer can be optimized by utilizing GDRCopy to write data to four GPUs within the same NUMA node, effectively reducing reads from host memory by threefold compared to MemCpyAsync. This efficiency is achieved because GDRCopy can read data from host memory and temporarily cache it in the CPU caches, allowing data to be written to the four GPUs without additional reads from host memory.

B. Advantages of HFReduce over NCCL

*1) **Reduced PCIe Bandwidth Consumption**:* Let n be the total number of GPUs involved in the communication. In NCCL's ring topology, each unit of data needs to go through $2n-1$ transmissions, each consuming one unit of inbound bandwidth of one GPU and one unit of outbound bandwidth of another GPU. This means for a single unit of data, it consumes $\frac{2n-1}{n}$ unit of PCIe bidirectional bandwidth. In contrast, for each unit of data, HFReduce only requires one D2H and one H2D data transfer, only one unit of PCIe bidirectional bandwidth is consumed. In our machine architecture, the performance of NCCL is mainly limited by PCIe bandwidth. Therefore, HFReduce can achieve better performance than NCCL.

*2) **No GPU Kernel Overhead**:* HFReduce utilizes the GPU's Copy Engine (CE) for PCIe asynchronous transfers. In contrast, NCCL's allreduce operation requires GPU kernel execution, which can affect other computational kernels on the GPU. HFReduce achieves complete asynchrony with no overhead.

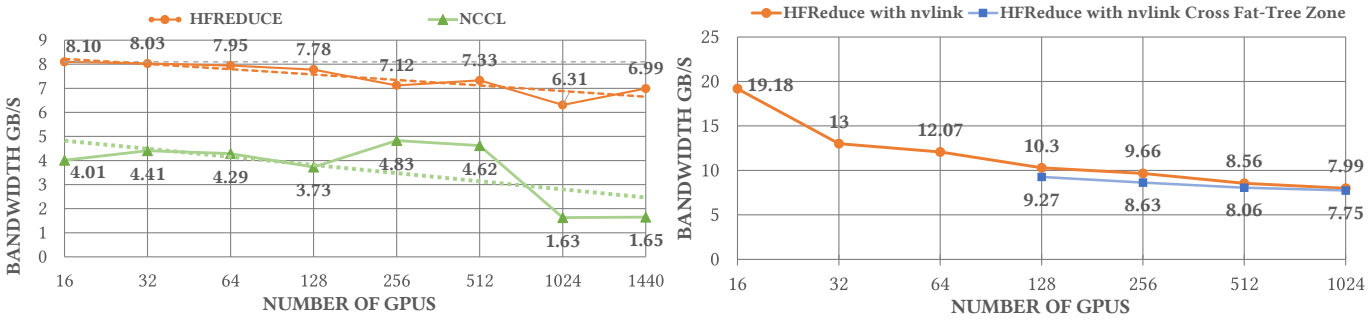
As demonstrated in Figure 7a, HFReduce can reach a inter-node bandwidths of 6.3-8.1GB/s when performing allreduce with a data size of 186 MiB on the Fire-Flyer 2 AI-HPC, while NCCL's inter-node bandwidth is only 1.6-4.8GB/s.

Algorithm 1: Intra Node Reduce.

```
Data: Dg: data need to allreduce
Result: Dc: data reduced in this node
Split Dg by Chunk_Size
for Dg-i in Splited-Dg do
    // Transfer GPU Memory Data to CPU memory
    Dc_i = MemCopyAsync Dg_i to CPU Mmeory
end
for i in Splited_Count do
    while every GPU's Dg_i → Dc_i finished do
        // Wait for chunk-i transfer finished in this node
        for j in GPU_Count do
            // do intra-node reduce
            Dc_i += GPU-j's Dc_i
        end
        // do inter-node reduce
        Do_Internode_reduce(Dc_i)
    end
end
```

C. Performance Improvements: HFReduce with NVLink

By installing the NVLink Bridge for PCIe A100 GPUs, efficient communication is enabled between paired GPUs via the 600 GB/s NVLink. To alleviate the memory bound issue of the original HFReduce, we implemented another allreduce pattern, termed **HFReduce with NVLink**. The core concept involves initially performing a reduction operation among GPUs interconnected by NVLink before passing the gradient to the CPU. Subsequently, when the CPU returns the result, it splits the result data and returns them to the paired GPUs connected by NVLink respectively, then performs allgather via NVLink. As illustrated in Figure 7b, HFReduce with NVLink achieves inter-node bandwidths exceeding 10 GB/s.



(a) HFRudce and NCCL Allreduce Speed.

(b) HFReduce with NVLink (Cross Fat-Tree Zone).

Figure 7: Strong Scalability: (a) Network Bandwidth when HFReduce and NCCL do allreduce test with 186MiB data, scale from 16 to 1440 GPUs. (b) HFReduce with NVLink, and Cross Fat-Tree Zone. Note that tasks utilizing fewer than 128 GPUs do not require cross-zone nodes and are restricted by platform defaults.

Algorithm 2: Inter Node Reduce.

```
Data: DL:local node reduced data by Algorithm 1
Data: DR: received other node reduced data
Result: Dg: reduced data transfer to GPU
// Pass 1: reduce data,individual thread
for i in Chunk_Size do
    receive data DR_i from prev node
    if DL_i ready then
        DL_i +=DR_i // reduce data
    end
    if Thread is root of Tree then
        send DL_i to prev node // Dg_i finished, go parse 2
    else
        send DL_i to next node
    end
end
//
// Pass 2: gather reduced data,individual thread
receive data DR_i from next node
for j in GPU_Count do
    Dg_i = MemCopyAsync DR_i to GPU-j // Dg_i is allreduced
end
send DL_i to prev node
```

D. Deep Analysis of HFReduce

1) Key Technical Strategies in Implementation:

- Using GDRCopy accelerate small data transfer in D2H, and educing reads from host memory by three times compared to MemCpyAsyn.
- Intra-Node Reduction: CPU utilizes SIMD instructions and supports FP32 / FP16 / BF16 / FP8 datatypes.
- NUMA Awareness: D2H destination memory is interleaved across two NUMA nodes for maximum bandwidth. Memory for CPU-added results and network-received data is bound to the IB-Nic's NUMA node to minimize latency.
- Inter-Node Reduce:Implements a Double Binary Tree allreduce algorithm [65] via ibverbs RDMA Write, avoiding additional overhead.

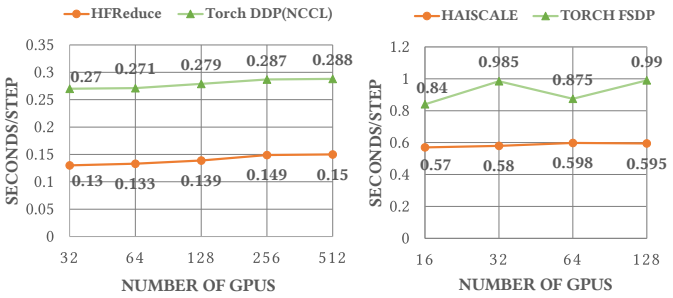
2) HFReduce Overcomes Limitations of EPYC Rome CPU: We consulted AMD and NVIDIA engineers to identify the root cause of NCCL's suboptimal performance on PCIe architecture, particularly with EPYC Rome CPU servers. It was determined that the Rome CPUs do not support the chained write feature, which can significantly accelerate PCIe peer-to-peer (P2P) transfers between GPUs and IB NICs. Our tests indicate that the maximum bandwidth between the GPU and IB NIC on Rome CPUs is approximately 9 GiB/s, making the observed 4GB/s all-reduce bandwidth for NCCL understandable. HFReduce circumvents this limitation by utilizing the CPU for reduction and transferring data through IB and host memory.

*3) **Bottlenecks of HFReduce**:* When considering the total memory operations on a single node during HFReduce, several factors contribute to its performance limitations:

1) D2H Phase requires 8 write operations.
2) Intra-node Reduce Add Phase involves 8 read operations and 1 write operation.
3) Inter-node Allreduce Phase: IB send demands 2 read operations, while IB receive requires 2 write operations, along with 1 read operation for reduce add.
4) H2D Phase Utilizing GDRCopy can reduce this to only 2 read operations, whereas MemCopy necessitates 8 read operations.

In total, the memory operations amount to 24 times the original data size in the GPU. A host equipped with 16 channels of DDR4-3200MHz memory can achieve a practical memory access speed of 320GB/s. Consequently, the theoretical maximum speed of HFReduce is approximately 13.3GB/s, but when considering the allreduce algorithm bandwidth and network bandwidth, this value realistically approximates 12GB/s. However, our tests only achieved slightly over 8GB/s.

The root cause of this discrepancy is another limitation of the EPYC CPUs. As previously mentioned, our GPU5 and



(a) HFReduce v.s. Torch DDP. (b) HaiScale v.s. Torch FSDP.

Figure 8: Weak Scalability: (a) Training VGG16, HFReduce compared to PyTorch DDP's NCCL backend (b) Training GPT2-Medium HaiScale compared to Torch, both using FSDP.

GPU6 are directly connected to the CPU via the same PCIe Root Complex Port (also known as the PCIe Host Bridge). In AMD EPYC Rome and Milan CPUs, the maximum bandwidth from the Root Complex Port to the CPU's internal bus is about 37.5GB/s. Although a PCIe 4.0 x16 port can achieve over 27GB/s from GPU to CPU, when two GPUs transfer data concurrently, the bandwidth is limited to around 37GB/s. Furthermore, if bidirectional data transfer occurs simultaneously, this bandwidth decreases even further. As a result, HFReduce does not reach its theoretical speed.

Employing NVLink with HFReduce offers a functional method to alleviate these bottlenecks. However, it is worth noting that the next-generation CPUs, such as the EPYC Genoa, still face issues with PCIe Host Bridge bandwidth, which cannot support two full-speed PCIe ports simultaneously. We hope AMD will address this issue in future iterations.

V. HaiScale: Special Optimization for Deep Leaning Models Training

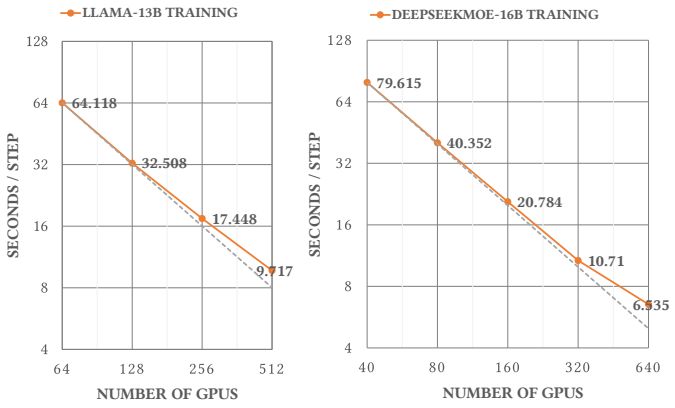
A. HaiScale DDP Overlap AllReduce in Training

HaiScale Distributed Data Parallel (DDP) is a training tool that utilizes HFReduce as its communication backend, in contrast to PyTorch's DDP [67] which employs NCCL as its backend. During the backpropagation phase, HaiScale DDP performs an asynchronous allreduce operation on the computed gradients, allowing this communication to overlap with the computation involved in backpropagation.

As previously mentioned, HFReduce does not depend on GPU Streaming Multiprocessors (SM) for reduction computation, enabling completely asynchronous allreduce without impacting performance. As shown in Figure 8a, training VGG16 model [68] with HFReduce takes only half the time compared to using Torch DDP's NCCL backend, achieving nearly 88% parallel scalability when scale from 32 GPUs to 512.

B. LLMs Training Optimization

Our HaiScale framework various parallelism strategies for training large language models (LLMs), similar to Megagron [69] and DeepSpeed [70]. We have made specific engineering optimizations for our PCIe architecture across Data Parallelism (DP), Pipeline Parallelism (PP) [11] [12], Tensor Parallelism (TP) [14], Expert Parallelism (EP) [15]–[17].



(a) Train LLaMa-13B (b) Train DeepSeekMoE-16B

Figure 9: Strong Scalability: (a) Train LLaMa-13B with a config of sequence length 2048, batch size 4096, pipeline parallel 4. (b)Train DeepSeekMoE-16B with a config of sequence length 4096, batch size 4608, pipeline parallel 10.

1) NVLink Bridge Enables Tensor Parallel between PCIe GPUs: With the advent of LLMs, we integrated the NVLink Bridge into our system. This addition established a bandwidth of 600GB/s between each pair of GPUs, enabling more efficient when performing Tensor Parallelism.

2) Pipeline Parallelism Optimization in PCIe Architecture: In our architecture, there is only one IB NIC for 8 GPUs on a single node, which can lead to network bandwidth contention during Pipeline Parallelism (PP). We solve this by configuring Data Parallelism (DP) rank, making the 8 GPUs on the same node belong to different DP ranks which staggers the timing of PP for each DP rank. As Figure 9a shown, when scaling from 64 GPUs to 512 GPUs, the training step time for LLaMa-13B [71] reduces significantly from 64.118 seconds to 9.717 seconds, achieving a parallel efficiency of **91%** at 256 GPUs.

We also benchmarked the training performance of our DeepSeekMoE-16B model [72] on Fire-Flyer 2 AI-HPC. As shown in Figure 9b, scaling from 40 GPUs to 640 GPUs reduced the time per training step from 79.615 seconds to 6.535 seconds, achieving a parallel efficiency of 76.14%. Notably, with 320 GPUs, the step time was 10.71 seconds, resulting in a parallel efficiency of **92.92%**, demonstrating excellent scalability.

3) Fully Sharded Data Parallel (FSDP): Both HaiScale's FSDP and PyTorch's FSDP [18] are implementations based on the ZeRO Stage-3 algorithm [19]. The details of this implementation are already discussed in Section II-B1.

HaiScale's FSDP offers better engineering implementation, optimizing memory management to reduce fragmentation specific to model adjustments. And we overlap allgather and reduce-scatter communication with forward and backward computation, split the optimization step during backward propagation for enhanced overlap. As shown in Figure 8b, training GPT2-medium [73], we achieve 95% parallel scalability when scaling from 16 to 128 GPUs. ompared to PyTorch's FSDP, HaiScale's FSDP reduces training time by nearly half.

C. Summary

Our AI-HPC design meets DL requirements, and with the addition of the NVLink Bridge, it meets the training needs of early-stage LLMs, reaching the utilization upper limit of PCIe GPUs. However, due to the inherent gap between PCIe card specifications and SXM, there is a certain performance discrepancy. Considering overall performance, basic setup cost, and energy consumption, we achieved 80% performance at half the cost. We believe Fire-Flyer 2 AI-HPC is a successful practice in terms of cost-effectiveness.

VI. Advanced Cost-Effective and Co-Design Optimizations

A. Ensuring Minimal Congestion in Our Computation-Storage Integrated Network

As previously stated, our cost-effective network integrated computation communication and storage traffics together. To achieve maximum bandwidth, it is essential to isolate interference between different types of traffic and control network congestion. In practice, we implemented the following measures:

1) Divergence of Different Traffics: In typical training tasks, there are four different types of traffic: HFReduce communication, NCCL communication, 3FS storage traffic, and other traffic. By using InfiniBand's Service Level (SL) technology [74] [75], we assign different value of SL when establishing connections between nodes and map SL to IB physical queues Virtual Lanes (VLs) [74] [75]. The use of Virtual Lanes ensures that flows in distinct lanes do not interfere with each other. Ultimately, we configured their proportions to implement traffic isolation, thereby preventing network congestion caused by Head-of-line (HOL) blocking [76] and different traffic collisions.

2) Topology Adjustment and Route Optimization: In high-throughput storage scenarios, there naturally exist many incast communication patterns, leading to certain congestion in the network. Under such circumstances, we observed that enabling adaptive routing would lead to more severe congestion spread in the network. Therefore, we opted for a static routing strategy. Based on the static routing scheme, to evenly disperse storage traffic into leaf $\rightarrow$ spine links, we distribute various nodes (storage, computation, management nodes) evenly disperse storage traffic into leaf $\rightarrow$ spine links.

3) NCCL Optimization: We adjusted the NCCL topology to route through the IB NIC and GPUs within the same NUMA node. This adjustment reduced PCIe congestion caused by CPU chiplet interconnects. Additionally, by using PCIe Relaxed Ordering [77], we further reduced congestion and increased bandwidth.

4) Network Tuning in 3FS: 3FS implements a request-to-send control mechanism to mitigate the congestion. Details are discussed in the next subsection, Key Techinical Points of 3FS.

B. High-Throughput Distributed File System: 3FS

1) Overview: 3FS is our in-house developed high performance distributed file system, akin to WekaFS [78], DAOS [79], [80], and BeeGFS [81]. However, the design and implementation of 3FS specifically focus on fully utilizing the high IOPS and throughput of NVMe SSDs and the RDMA network.

Table IV: Storage Node Hardware Details

CPU	1 * AMD 64 Cores EPYC 7742 CPU
Memory	512GB 8-Channels DDR4-3200Mhz
NICs	2 * Mellanox InfiniBand CX6 200Gbps NIC
Data SSDs	16 * 15.36TB PCIe 4.0x4

2) 3FS Storage Node Hardware: In Fire-Flyer 2 AI-HPC, we deployed 180 storage nodes, as shown in Table IV, each node contains 16 PCIe 4.0 NVMe SSDs and 2 Mellanox CX6 200Gbps InfiniBand HCAs. With totally 360 * 200Gbps outbound InfiniBand HCAs, the system can total provide 9TB/s outbound bandwidh, and we actually achieved total **read throughput of 8TB/s**. The total 2880 NVMe SSDs provide over 20PiB storage space with an mirror data redundancy.

3) Key Techinical Points of 3FS: The 3FS system comprises four roles: cluster manager, meta service, storage service and client. Meta and storage services send heartbeats to cluster manager. All services and clients poll cluster configuration and service status from the manager. Multiple cluster managers are present, with one elected as the primary.

File system meta data are stored in tables of a distributed key-value storage system. Each file or directory has a unique inode ID. The File inode/directory ID and meta data, such as file size and location information of the file content data, are stored as key-value pairs in the inode table. A separate directory entry table stores key-value pairs of $(parent\_dir\_inode\_id, entry\_name)$: $(entry\_inode\_id, ...)$ to support iterating entries in a directory and resolving file/directory paths. All states of meta services are persisted on the distributed key-value storage system. Several meta services run concurrently to handle meta requests from clients.

The storage service has an implementation of Chain Replication with Apportioned Queries (CRAQ) [82] to provide strong consistency. CRAQ's write-all-read-any approach helps to unleash the throughput and IOPS of all SSDs. File content are split into chunks, which are replicated over a chain of *storage targets*. A *chain table* contains an ordered set of chains. The meta service selects an offset in the chain table and a stripe size *k* for each file. The file chunks are assigned to the next *k* chains starting at the offset. To distribute read/write traffic evenly to all SSDs, each SSD serves multiple storage targets from different chains. The storage service runs on every storage node and manages a few storage targets.

The storage network has a Fat-Tree topology that provides full bisection bandwidth. By design, each 3FS client can access every storage service. At peak load, incast congestion is

observed on the client side. **To mitigate this congestion, a request-to-send control mechanism is implemented** in storage service and client [83]. After receiving a read request from a client, the service reads data from SSD and asks the client's permission to transfer the data. The client limits the number of concurrent senders. When a storage service is granted the permission to transfer, it sends the data with a RDMA WRITE followed by a RDMA SEND to notify the client. The request-to-send control increases end-to-end IO latency but it's required to achieve sustainable high throughput.

4) 3FS-KV: 3FS-KV is a shared-storage distributed data processing system built on top of 3FS, currently supporting three models: key-value, message queue, and object storage. It supports read-write separation and on-demand startup, allowing it to fully leverage the extremely high I/O throughput provided by 3FS. 3FS-KV supports DeepSeek's KV Context Caching on Disk technology [84], which reduces the cost of LLM serving by an order of magnitude.

C. HAI Platform: a Time-Sharing Scheduling Platform

The principle of time-sharing scheduling is applied to cluster resource management. Users submit tasks, such as running Python / bash code, starting development containers, etc., and the platform interrupts and loads tasks according to current resource requirements, cluster busyness, etc. Task code needs to follow the platform coding rules to ensure that it can be continued from breakpoints, with the specific process as follows:

- Accepting the interruption signal from the cluster;
- Saving checkpoints (model parameters, optimizer parameters, etc.);
- Notifying the cluster of interruptions;
- Recovering from the checkpoint and continuing to run.

The cluster deploying HAI Platform does not pool GPU resources, but classifies and marks them based on computing nodes as basic units, according to resource types, network areas, etc. The HAI Platform encourages users to fully utilize multiple GPUs simultaneously for parallel training, facilitating 99% utilization.

VII. Stability and Robustness

A. Checkpoint Manager

Training LLMs can span several months, during which unavoidable hardware failures may cause training interruptions. To minimize recovery time and support the HAI Platform's interrupt and recovery operations, we developed a checkpoint manager. Additionally, the substantial size of LLM checkpoints necessitated an efficient method for saving and loading them, leveraging the high throughput of 3FS. The checkpoint manager includes the following components:

- Parameters and optimization states are divided into chunks and written to 3FS using the 3FS batch write API, which is significantly faster than normal writes, achieving over 10 GiB/s per node. This enables saving to be completed in just a few seconds.
- Parameters and optimization states are asynchronously transferred from GPU to CPU host memory, with checkpoint saving performed periodically (typically every 5 minutes).
- During the saving process, each tensor is recorded with its index and the offset within the checkpoint., which makes the location of tensors more convenient during the loading process. With the 3FS batch read API, a loading process can be completed in just a few seconds.

Thanks to the high write throughput of 3FS, periodic saving operations can be completed asynchronously in a matter of seconds, without impacting the training process. In the event of hardware failures that interrupt training, only the last 5 minutes of progress are lost. For a cluster with thousands of nodes, this overhead from disaster recovery is minimal.

B. Validator

The best way to enhance device stability is to identify issues before they occur. Therefore, we have developed a set of validator tools to verify whether the hardware is functioning correctly. The platform's automatic operation and maintenance system runs the validator program weekly on nodes to verify their proper functionality. It removes the faulty nodes from the scheduling platform, ensuring that all scheduled nodes are operational. Diagnosing tools like hostping [85] also integrated in our platform, but to find root cause of Hardware Failures is still hard work for operation teams. The validator mainly consists of the following parts:

- Checking hardware frequency, link speed, and link status.
- Testing CPU stress and memory bandwidth.
- GPU Memory test: This involves checking each byte of GPU memory to ensure no data corruption has occurred.
- Running GEMM with full GPU memory occupancy, which can simultaneously check whether there are any operational logic faults in the GPU chip.
- Intra-node allreduce test: checking NVLink bandwidth through upper-level applications.
- Storage bandwidth stress test to make sure storage is functioning normally.

C. Hardware Failures Characterization in Fire-Flyer 2 AI-HPC

In supercomputers and data centers, hardware failures and chip errors can lead to floating-point overflow, non-convergence, or slow convergence during model training [86]. This paper [87] even directly points out that there is a substantial amount of Silent Data Corruption in data center processors, ultimately leading to a variety of complex issues that are difficult to replicate and locate. Indeed, in our practice, we have encountered computational errors and GPU memory errors not detected by Error Correction Code (ECC), which led to models' gradnorm spikes, loss explosions and even non-convergence. How to tackle these hardware failures, promptly identify and categorize them, is a key issue to improve the online rate and overall utilization of cluster nodes.

Table V: Type of GPU Xid Errors and Its Causes

Xid Errors	Analysis
Software Causes: Xid_13/31 Xid_43/45	Triggered by application programs, software-related Xid messages may indicate anomalies in GPU memory affecting code and data segments. However, it's crucial to consider other information for a comprehensive hardware functionality assessment.
NVLink Error: Xid 74	Xid74 indicates errors in NVLink. For PCIe A100, it's mainly occurred on the NVLink Bridge between two GPUs. Its occurrence rate is several orders of magnitude higher than other hardware faults. Apart from stress testing to exclude those that are constantly repeating errors, there isn't a good way to avoid the occurrence of Xid74 issues.
Memory ECC Error: Xid_63/64 Xid_94/95	Triggered when the GPU handles memory ECC errors on the GPU. With the introduction of row remapping technology in A100, most instances can be resolved by simply resetting the GPU to retain optimal performance.
Uncorrectable GPU Failures: Xid_44/48 Xid_61/62/69/79	Thease failures mean an uncorrectable error occurs on the GPU, which is also reported back to the user application. A GPU reset or node reboot is needed to clear this error.
Other Failures: Xid 119	Xid119 means GPU GSP module failed. These failures need to do fieldiag test, and most need to RMA.

*1) **GPU Xid Error**:* An Xid error [88] is a general GPU fault message that originates from the NVIDIA driver, logged into the kernel or event log of the operating system. We have categorized various types of Xid errors and analyzed the potential causes that may lead to such errors, as shown in the Table V.

Table VI in Appendix: Supplementary Characterization shows the Xid errors that have occurred in our Fire-Flyer 2 AI-HPC over the past year. In our PCIe-based system, Xid74, also known as NVLink errors, account for a significant proportion, comprising 42.57% of the total. This high frequency is due to the inherent fault rate of NVLink Bridge connectors, amplified by our extensive use of thousands of GPUs.

Software-related errors such as Xid13, Xid31, Xid43, and Xid45 suggest possible illegal memory access or instructions in user code. Notably, illegal memory access (Xid 43) accounts for 33.48%, highlighting the need for improved memory management. However, these errors may also result from memory data corruption, so hardware faults should be considered if software bugs are ruled out.

Additionally, GPU Memory ECC Errors, such as Xid63, Xid64, Xid94, and Xid95, require special attention as they represent about 2% of the total. Figure 10 illustrates the statistics related to ECC errors in our production cluster over the past six months. It is evident that the number of GPU ECC faults considerably surpasses those from the CPU. Therefore, it is crucial to promptly address GPU ECC faults to ensure that application performance and accuracy remain unaffected.

*2) **Network Flash Cut**:* In addition to CPU and GPU faults, network device malfunctions represent a significant portion of hardware issues. As shown in Figure10, IB link failures account for 30% of hardware faults excluding Xid74. Network

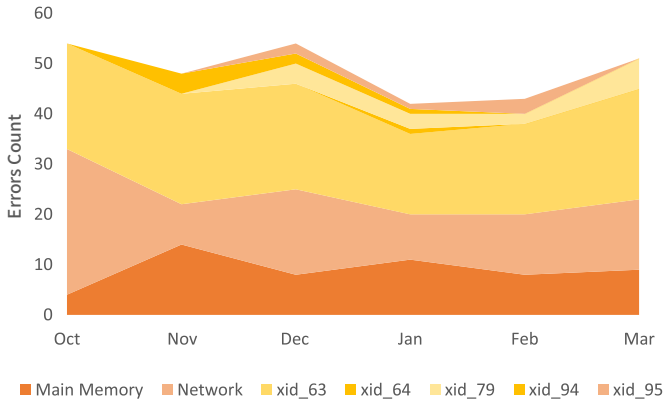


Figure 10: Trends of Memory and Network Failures from 2023 to 2024: "Main Memory" indicates CPU Memory ECC errors, "Network" indicates Network Flash Cuts, and "xids" are related to GPU memory ECC errors. Raw data is available in Appendix: Supplementary Characterization, Table VII

.

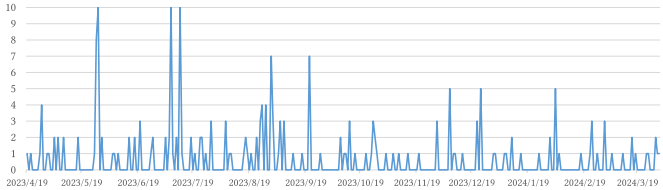


Figure 11: Trends of IB Network Failures: Link Flash Cuts

flash cuts can lead to application communication disruption, even task failures. Since most tasks run on multiple nodes, an issue on a single node can impact many others, further reducing cluster utilization. Figure 11 illustrates the IB link failures data over the past year, with raw data attached in Appendix: Supplementary Characterization, Table VIII, indicating that these issues can occur **randomly** throughout the cluster's operational period.

VIII. Discussion

A. Discussion on Congestion Control in RDMA Networks

Lossless RDMA networks offer several flow-control mechanisms, such as Priority Flow Control (PFC) [89] for RoCE networks and credit-based flow control [90] for IB networks. In network routing, static routing algorithms in IB or ECMP (Equal-Cost Multi-Path) [91] and AR (Adaptive Routing) [92] effectively handle routing issues. However, congestion can still occur when multiple servers send data to a single receiver, potentially blocking the entire network. To mitigate this, IB NICs use DCQCN (Data Center Quantized Congestion Notification) [93] as their congestion control algorithm. While Data Processing Units (DPUs), such as the NVIDIA BF series, allow users to customize congestion control algorithms (like HPCC [94] and TIMELY RTT-based CC [95]), they increase the cost and operational complexity of the cluster.

In practice, we chose to disable DCQCN to avoid its shortcomings, as it could not find parameters that simul-

taneously support HFReduce traffic and 3FS storage traffic in our Computation-Storage Integrated Network. Instead, we employed the network tuning methods mentioned in Section VI-A, ensuring our network operates without congestion control and remains congestion-free.

B. Discussion about NVLink Technology Choices

Initially, we did not use NVLink to avoid extra costs and maintain stability, as HFReduce was sufficient for training requirements at that time. However, as the demand for LLMs increased, we added NVLink specifically for LLM training purposes. The decision to install NVLink should be based on actual needs due to its potential drawbacks.

C. Maintaince Cost Overview

1) Construction Cost: Relative hardware costs are provided in Table II and III. Software costs, contributed by several dozen in-house developers, are just a fraction of the cost for thousands of GPU servers.

2) Power Consumption: The average power consumption comparision during ResNet training is provided in Table II. Including the overhead from IB switches and other nodes, the total energy consumption of the Fire-Flyer 2 AI-HPC does not exceed 4 MW, approximately just over 3 MW.

3) Operation Cost: Operating costs can be estimated by considering power consumption and rack rental costs. By multiplying this figure by the number of nodes and the PUE (Power Usage Effectiveness), the total operating costs can be calculated.

D. Stability Compared with Other Architectures

A recent paper [96] reportsthat NVLink-related failures account for approximately 52.42% (54 out of 103) of total failures, with raw data indicating 54 NVLink Errors, 21 CUDA Errors, 16 Node Failures, 12 ECC Errors, and 12 Network Errors. In comparison, our NVLink-related issues, primarily Xid-74 Errors, as mentioned in Section VII-C1, account for about 42.57% of GPU failures.

IX. Future Work

Future Arch and Integration with New GPU Models

Our next-generation PCIe architecture is designed for MoE (Mixture of Experts) LLM training, where all-to-all performance is crucial. Therefore, the next-gen nodes feature a 1:1 GPU to NIC ratio, comparable to DGX-H100/B100 systems, as illustrated in Figure 12.

We are considering implementing a multi-plane network to reduce costs while maintaining performance. Additionally, we are exploring the use of RoCE switches instead of IB switches, which can significantly lower network expenses. With a 128-port 400 Gbps RoCE switch, a 4-Plane Two-Layer Fat-Trees network can support up to 32,768 GPUs.

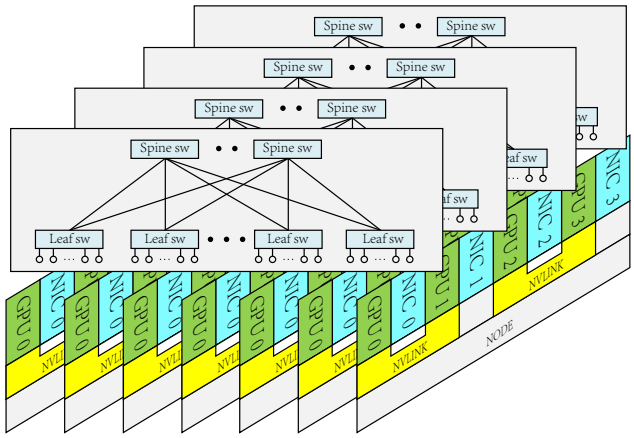


Figure 12: Next Generation PCIe Node Atchitecture with Multi-Plane Fat-Trees Network

X. Conclusions

In this paper, we have shared our experiences and insights from deploying and maintaining the Fire-Flyer 2 AI-HPC, which is equipped with 10,000 PCIe A100 GPUs.Our approach to PCIe architecture and storage-computation integrated network design has resulted in significant cost savings, effectively halving construction costs and demonstrating substantial cost-effectiveness.

In terms of software co-design, we introduced HFReduce and HaiScale to overcome hardware limitations, ensuring scalable performance of the PCIe architecture. Our in-house developed 3FS distributed file system,in conjunction with network co-design, facilitates traffic isolation for both 3FS and HFReduce allreduce traffic, effectively preventing congestion. The comprehensive software stack within the HAI Platform addresses a variety of system faults, from network congestion to hardware failures, thereby ensuring high stability and robustness.

Together, these software and hardware innovations enable our PCIe A100 architecture to achieve 80% the performance of NVIDIA's DGX-A100, while consuming less than 60% of its power. The practical knowledge we have accrued may prove valuable for both industrial and academic sectors. We hope that our work will serve as a reference for others aiming to build their own cost-effective and efficient AI-HPC clusters.

Acknowledgment

We extend our heartfelt gratitude to all our colleagues at DeepSeek-AI and High-Flyer Quant for their invaluable contributions to the Fire-Flyer 2 AI-HPC project. Throughout the four years of design, construction, and operation, their collaborative efforts have been crucial in overcoming numerous challenges. Consequently, Fire-Flyer 2 AI-HPC now supports the training tasks of the DeepSeek-AI series large language models [72], [97]–[101]. We extend a special thanks to the HFAiLab System Team and the Operations Team, whose core contributions have been essential to the success of this endeavor.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015. I

[2] R. R. Schaller, "Moore's law: past, present and future," *IEEE spectrum*, vol. 34, no. 6, pp. 52–59, 1997. I

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020. I

[4] Anthropic, "Introducing claude," 2023. [Online]. Available: https://www.anthropic.com/news/introducing-claude I

[5] Google, "An important next step on our ai journey," 2023. [Online]. Available: https://blog.google/technology/ai/bard-google-ai-search-updates/ I

[6] OpenAI, "Chatgpt: Optimizing language models for dialogue," 2022. [Online]. Available: https://openai.com/blog/chatgpt I

[7] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023. I, II-A

[8] NVIDIA, "Nvidia dgx platform the best of nvidia ai—all in one place." 2022. [Online]. Available: https://www.nvidia.com/en-us/data-center/dgx-platform/ I, III-C

[9] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 892–901, Oct 1985. I, III-B

[10] NVIDIA, "Nvidia collective communications library (nccl): Optimized primitives for collective multi-gpu communication," 2017. [Online]. Available: https://github.com/NVIDIA/nccl I, III-B

[11] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. V. Le, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *CoRR*, vol. abs/1811.06965, 2018. [Online]. Available: http://arxiv.org/abs/1811.06965 I, II-B1, V-B

[12] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: generalized pipeline parallelism for dnn training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ser. SOSP '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–15. [Online]. Available: https://doi.org/10.1145/3341301.3359646 I, II-B1, V-B

[13] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism," Mar. 2020, arXiv:1909.08053 [cs]. [Online]. Available: http://arxiv.org/abs/1909.08053 I, II-B1

[14] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "Efficient large-scale language model training on gpu clusters using megatron-lm," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476209 I, II-B1, V-B

[15] S. Singh, O. Ruwase, A. A. Awan, S. Rajbhandari, Y. He, and A. Bhatele, "A hybrid tensor-expert-data parallelism approach to optimize mixture-of-experts training," in *Proceedings of the 37th ACM International Conference on Supercomputing*, ser. ICS '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 203–214. [Online]. Available: https://doi.org/10.1145/3577193.3593704 I, II-B1, V-B

[16] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Aminabadi, A. Awan, J. Rasley, and Y. He, "Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale," *Proceedings of Machine Learning Research*, vol. 162, pp. 18 332–18 346, 2022, publisher Copyright: Copyright © 2022 by the author(s); 39th International Conference on Machine Learning, ICML 2022 ; Conference date: 17-07-2022 Through 23-07-2022. I, II-B1, V-B

[17] C. Hwang, W. Cui, Y. Xiong, Z. Yang, Z. Liu, H. Hu, Z. Wang, R. Salas, J. Jose, P. Ram *et al.*, "Tutel: Adaptive mixture-of-experts at scale," *Proceedings of Machine Learning and Systems*, vol. 5, 2023. I, II-B1, V-B

[18] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer *et al.*, "Pytorch fsdp: experiences on scaling fully sharded data parallel," *arXiv preprint arXiv:2304.11277*, 2023. I, V-B3

[19] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16. I, II-B1, II-B1, V-B3

[20] HFAiLab, "Hai platform: A high-performance deep learning training platform with task-level gpu compute time-sharing scheduling," 2023. [Online]. Available: https://github.com/HFAiLab/hai-platform I

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012. II-A

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. II-A, III

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017. II-A

[24] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Žídek, A. W. Nelson, A. Bridgland *et al.*, "Improved protein structure prediction using potentials from deep learning," *Nature*, vol. 577, no. 7792, pp. 706–710, 2020. II-A

[25] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018. II-A

[26] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, pp. 681–694, 2020. II-A

[27] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023. II-A

[28] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991. II-A

[29] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the em algorithm," *Neural computation*, vol. 6, no. 2, pp. 181–214, 1994. II-A

[30] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017. II-A

[31] T. Brooks, B. Peebles, C. Holmes, W. DePue, Y. Guo, L. Jing, D. Schnurr, J. Taylor, T. Luhman, E. Luhman, C. Ng, R. Wang, and A. Ramesh, "Video generation models as world simulators," 2024. [Online]. Available: https://openai.com/research/video-generation-models-as-world-simulators II-A

[32] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, "AI and Memory Wall," Mar. 2024, arXiv:2403.14123 [cs]. [Online]. Available: http://arxiv.org/abs/2403.14123 II-A

[33] P. Qi, X. Wan, G. Huang, and M. Lin, "Zero bubble pipeline parallelism," *arXiv preprint arXiv:2401.10241*, 2023. II-B1

[34] V. A. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, "Reducing activation recomputation in large transformer models," in *Proceedings of the Sixth Conference on Machine Learning and Systems, MLSys 2023, Miami, FL, USA, June 4-8, 2023*, D. Song, M. Carbin, and T. C. 0001, Eds. mlsys.org, 2023. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2023/hash/80083951326cf5b35e5100260d64ed81-Abstract-mlsys2023.html II-B1

[35] Z. Sun, H. Cao, Y. Wang, G. Feng, S. Chen, H. Wang, and W. Chen, "Adapipe: Optimizing pipeline parallelism with adaptive recomputation and partitioning," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 86–100. [Online]. Available: https://doi.org/10.1145/3620666.3651359 II-B1

[36] X. Hou, Y. Yuan, S. Ma, R. Xu, B. Wang, T. Li, W. Jiang, L. Wu, and J. Zhang, "Optimizing the parallelism of communication and computation in distributed training platform," in *Algorithms and Architectures for Parallel Processing: 23rd International Conference, ICA3PP 2023, Tianjin, China, October 20–22, 2023, Proceedings, Part I*. Berlin, Heidelberg: Springer-Verlag, 2024, p. 340–359. [Online]. Available: https://doi.org/10.1007/978-981-97-0834-5_20 II-B1

[37] J. Dongarra, "REPORT ON THE TIANHE-2A SYSTEM," 2017. II-C1

[38] D. Stanzione, B. Barth, N. Gaffney, K. Gaither, C. Hempel, T. Minyard, S. Mehringer, E. Wernert, H. Tufo, D. Panda, and P. Teller, "Stampede 2: The evolution of an xsede supercomputer," in *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, ser. PEARC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3093338.3093385 II-C1

[39] H. Fu, J. Liao, J. Yang, L. Wang, Z. Song, X. Huang, C. Yang, W. Xue, F. Liu, F. Qiao, W. Zhao, X. Yin, C. Hou, C. Zhang, W. Ge, J. Zhang, Y. Wang, C. Zhou, and G. Yang, "The Sunway TaihuLight supercomputer: system and applications," *Science China Information Sciences*, vol. 59, no. 7, p. 072001, Jul. 2016. [Online]. Available: http://link.springer.com/10.1007/s11432-016-5588-7 II-C1

[40] T. Shimizu, "Supercomputer Fugaku: Co-designed with application developers/researchers," in *2020 IEEE Asian Solid-State Circuits Conference (A-SSCC)*. Hiroshima, Japan: IEEE, Nov. 2020, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/document/9336127/ II-C1

[41] D. Schneider, "The Exascale Era is Upon Us: The Frontier supercomputer may be the first to reach 1,000,000,000,000,000,000 operations per second," *IEEE Spectrum*, vol. 59, no. 1, pp. 34–35, Jan. 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9676353/ II-C2

[42] A. N. Laboratory, "Argonne's aurora supercomputer," 2023. [Online]. Available: https://www.alcf.anl.gov/aurora II-C2

[43] C. B. Stunkel, R. L. Graham, G. Shainer, M. Kagan, S. S. Sharkawi, B. Rosenburg, and G. A. Chochia, "The high-speed networks of the Summit and Sierra supercomputers," *IBM Journal of Research and Development*, vol. 64, no. 3/4, pp. 3:1–3:10, May 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8961159/ II-C2

[44] J. Gu, G. Eisenhauer, S. Klasky, N. Podhorszki, R. Wang, and K. Wu, "Exploring large all-flash storage system with scientific simulation," in *Proceedings of the 34th International Conference on Scientific and Statistical Database Management*, ser. SSDBM '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3538712.3538734 II-C2

[45] D. Mudigere, Y. Hao, J. Huang, Z. Jia, A. Tulloch, S. Sridharan, X. Liu, M. Ozdal, J. Nie, J. Park, L. Luo, J. A. Yang, L. Gao, D. Ivchenko, A. Basant, Y. Hu, J. Yang, E. K. Ardestani, X. Wang, R. Komuravelli, C.-H. Chu, S. Yilmaz, H. Li, J. Qian, Z. Feng, Y. Ma, J. Yang, E. Wen, H. Li, L. Yang, C. Sun, W. Zhao, D. Melts, K. Dhulipala, K. R. Kishore, T. Graf, A. Eisenman, K. K. Matam, A. Gangidi, G. J. Chen, M. Krishnan, A. Nayak, K. Nair, B. Muthiah, M. khorashadi, P. Bhattacharya, P. Lapukhov, M. Naumov, A. Mathews, L. Qiao, M. Smelyanskiy, B. Jia, and V. Rao, "Software-Hardware Co-design for Fast and Scalable Training of Deep Learning Recommendation Models," Feb. 2023, arXiv:2104.05158 [cs]. [Online]. Available: http://arxiv.org/abs/2104.05158 II-C3

[46] A. Gangidi, R. Miao, S. Zheng, S. J. Bondu, G. Goes, H. Morsy, R. Puri, M. Riftadi, A. J. Shetty, J. Yang, S. Zhang, M. J. Fernandez, S. Gandham, and H. Zeng, "Rdma over ethernet for distributed training at meta scale," in *Proceedings of the ACM SIGCOMM 2024 Conference*, ser. ACM SIGCOMM '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 57–70. [Online]. Available: https://doi.org/10.1145/3651890.3672233 II-C3

[47] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo, "A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters." II-C3

[48] Z. Jiang, H. Lin, Y. Zhong, Q. Huang, Y. Chen, Z. Zhang, Y. Peng, X. Li, C. Xie, S. Nong, Y. Jia, S. He, H. Chen, Z. Bai, Q. Hou, S. Yan, D. Zhou, Y. Sheng, Z. Jiang, H. Xu, H. Wei, Z. Zhang, P. Nie, L. Zou, S. Zhao, L. Xiang, Z. Liu, Z. Li, X. Jia, J. Ye, X. Jin, and X. Liu, "MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs," Feb. 2024, arXiv:2402.15627 [cs]. [Online]. Available: http://arxiv.org/abs/2402.15627 II-C3

[49] K. Qian, Y. Xi, J. Cao, J. Gao, Y. Xu, Y. Guan, B. Fu, X. Shi, F. Zhu, R. Miao, C. Wang, P. Wang, P. Zhang, X. Zeng, E. Ruan, Z. Yao, E. Zhai, and D. Cai, "Alibaba hpn: A data center network for large language model training," in *Proceedings of the ACM SIGCOMM 2024 Conference*, ser. ACM SIGCOMM '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 691–706. [Online]. Available: https://doi.org/10.1145/3651890.3672265 II-C3

[50] NVIDIA, "Nvidia announces dgx h100 systems – world's most advanced enterprise ai infrastructure," 2023. [Online]. Available: https://nvidianews.nvidia.com/news/nvidia-announces-dgx-h100-systems-worlds-most-advanced-enterprise-ai-infrastructure II-C3

[51] N. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles, C. Young, X. Zhou, Z. Zhou, and D. A. Patterson, "TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*. Orlando FL USA: ACM, Jun. 2023, pp. 1–14. [Online]. Available: https://dl.acm.org/doi/10.1145/3579371.3589350 II-C4

[52] C. Zhang, B. Sun, X. Yu, Z. Xie, W. Zheng, K. Iskra, P. Beckman, and D. Tao, "Benchmarking and In-depth Performance Study of Large Language Models on Habana Gaudi Processors," Sep. 2023, arXiv:2309.16976 [cs]. [Online]. Available: http://arxiv.org/abs/2309.16976 II-C4

[53] E. Talpes, D. Williams, and D. D. Sarma, "Dojo: The microarchitecture of tesla's exa-scale computer," in *2022 IEEE Hot Chips 34 Symposium (HCS)*, 2022, pp. 1–28. II-C4

[54] E. Talpes, D. D. Sarma, D. Williams, S. Arora, T. Kunjan, B. Floering, A. Jalote, C. Hsiong, C. Poorna, V. Samant, J. Sicilia, A. K. Nivarti, R. Ramachandran, T. Fischer, B. Herzberg, B. McGee, G. Venkataramanan, and P. Banon, "The microarchitecture of dojo, tesla's exa-scale computer," *IEEE Micro*, vol. 43, no. 3, pp. 31–39, 2023. II-C4

[55] H. Liao, J. Tu, J. Xia, and X. Zhou, "Davinci: A scalable architecture for neural network computing," in *2019 IEEE Hot Chips 31 Symposium (HCS)*, 2019, pp. 1–44. II-C4

[56] H. Liao, J. Tu, J. Xia, H. Liu, X. Zhou, H. Yuan, and Y. Hu, "Ascend: a scalable and unified architecture for ubiquitous deep neural network computing : Industry track paper," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 789–801. II-C4

[57] D. Milojicic, P. Faraboschi, N. Dube, and D. Roweth, "Future of hpc: Diversifying heterogeneity," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 276–281. II-D

[58] Y. Su, J. Zhou, J. Ying, M. Zhou, and B. Zhou, "Computing infrastructure construction and optimization for high-performance computing and artificial intelligence," *CCF Transactions on High Performance Computing*, vol. 3, no. 4, pp. 331–343, Dec. 2021. [Online]. Available: https://doi.org/10.1007/s42514-021-00080-x II-D

[59] E. Hassan, N. El-Rashidy, and F. M. Talaa, "Review: Mask R-CNN Models," *Nile Journal of Communication and Computer Science*, vol. 3, no. 1, pp. 17–27, May 2022. [Online]. Available: https://njccs.journals.ekb.eg/article_280047.html III

[60] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *North American Chapter of the Association for Computational Linguistics*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:52967399 III

[61] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," *arXiv:2111.06377*, 2021. III

[62] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *2008 International Symposium on Computer Architecture*, 2008, pp. 77–88. III-B

[63] G. Feng, D. Dong, and Y. Lu, "Optimized mpi collective algorithms for dragonfly topology," in *Proceedings of the 36th ACM International Conference on Supercomputing*, ser. ICS '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3524059.3532380 III-B

[64] H. Subramoni, P. Lai, M. Luo, and D. K. Panda, "Rdma over ethernet — a preliminary study," in *2009 IEEE International Conference on Cluster Computing and Workshops*, 2009, pp. 1–9. III-B

[65] P. Sanders, J. Speck, and J. Träff, "Two-tree algorithms for full bandwidth broadcast, reduction and scan," *Parallel Computing*, vol. 35, pp. 581–594, 12 2009. III-B, IV, 1, IV-D1

[66] R. Shi, S. Potluri, K. Hamidouche, J. Perkins, M. Li, D. Rossetti, and D. K. D. K. Panda, "Designing efficient small message transfer mechanism for inter-node mpi communication on infiniband gpu clusters," in *2014 21st International Conference on High Performance Computing (HiPC)*, 2014, pp. 1–10. 1

[67] P. Foundation, "Tensors and dynamic neural networks in python with strong gpu acceleration," 2016. [Online]. Available: https://github.com/pytorch/pytorch V-A

[68] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," in *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, 2015, pp. 730–734. V-A

[69] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019. V-B

[70] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "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*, ser. KDD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 3505–3506. [Online]. Available: https://doi.org/10.1145/3394486.3406703 V-B

[71] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," *ArXiv*, vol. abs/2302.13971, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:257219404 V-B2

[72] DeepSeek-AI, A. Liu, B. Feng, B. Wang, B. Wang, B. Liu, C. Zhao, C. Dengr, C. Ruan, D. Dai, D. Guo, D. Yang, D. Chen, D. Ji, E. Li, F. Lin, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Xu, H. Yang, H. Zhang, H. Ding, H. Xin, H. Gao, H. Li, H. Qu, J. L. Cai, J. Liang, J. Guo, J. Ni, J. Li, J. Chen, J. Yuan, J. Qiu, J. Song, K. Dong, K. Gao, K. Guan, L. Wang, L. Zhang, L. Xu, L. Xia, L. Zhao, L. Zhang, M. Li, M. Wang, M. Zhang, M. Zhang, M. Tang, M. Li, N. Tian, P. Huang, P. Wang, P. Zhang, Q. Zhu, Q. Chen, Q. Du, R. J. Chen, R. L. Jin, R. Ge, R. Pan, R. Xu, R. Chen, S. S. Li, S. Lu, S. Zhou, S. Chen, S. Wu, S. Ye, S. Ma, S. Wang, S. Zhou, S. Yu, S. Zhou, S. Zheng, T. Wang, T. Pei, T. Yuan, T. Sun, W. L. Xiao, W. Zeng, W. An, W. Liu, W. Liang, W. Gao, W. Zhang, X. Q. Li, X. Jin, X. Wang, X. Bi, X. Liu, X. Wang, X. Shen, X. Chen, X. Chen, X. Nie, X. Sun, X. Wang, X. Liu, X. Xie, X. Yu, X. Song, X. Zhou, X. Yang, X. Lu, X. Su, Y. Wu, Y. K. Li, Y. X. Wei, Y. X. Zhu, Y. Xu, Y. Huang, Y. Li, Y. Zhao, Y. Sun, Y. Li, Y. Wang, Y. Zheng, Y. Zhang, Y. Xiong, Y. Zhao, Y. He, Y. Tang, Y. Piao, Y. Dong, Y. Tan, Y. Liu, Y. Wang, Y. Guo, Y. Zhu, Y. Wang, Y. Zou, Y. Zha, Y. Ma, Y. Yan, Y. You, Y. Liu, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Huang, Z. Zhang, Z. Xie, Z. Hao, Z. Shao, Z. Wen, Z. Xu, Z. Zhang, Z. Li, Z. Wang, Z. Gu, Z. Li, and Z. Xie, "DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model," Jun. 2024, arXiv:2405.04434 [cs]. [Online]. Available: http://arxiv.org/abs/2405.04434 V-B2, X

[73] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019. V-B3

[74] S.-A. Reinemo, T. Skeie, T. Sodring, O. Lysne, and O. Trudbakken, "An overview of qos capabilities in infiniband, advanced switching interconnect, and ethernet," *IEEE Communications Magazine*, vol. 44, no. 7, pp. 32–38, 2006. VI-A1, VI-A1

[75] D. Crupnicoff, S. Das, and E. Zahavi, "Deploying Quality of Service and Congestion Control in InfiniBand-based Data Center Networks." [Online]. Available: https://network.nvidia.com/sites/default/files/related-docs/whitepapers/deploying_qos_wp_10_19_2005.pdf VI-A1, VI-A1

[76] M. Scharf and S. Kiesel, "Nxg03-5: Head-of-line blocking in tcp and sctp: Analysis and measurements," in *IEEE Globecom 2006*, 2006, pp. 1–5. VI-A1

[77] R. Budruk, D. Anderson, and E. Solari, *PCI Express System Architecture*. Pearson Education, 2003. VI-A3

[78] Z. Liran, H. David, and M. Barbara, "Wekafs architecture white paper," 2021. [Online]. Available: https://www.weka.io/wp-content/uploads/files/2017/12/Architectural_WhitePaper-W02R6WP201812-1.pdf VI-B1

[79] Z. Liang, J. Lombardi, M. Chaarawi, and M. Hennecke, "Daos: A scale-out high performance storage stack for storage class memory," in *Supercomputing Frontiers*, D. K. Panda, Ed. Cham: Springer International Publishing, 2020, pp. 40–54. VI-B1

[80] M. Hennecke, "Understanding daos storage performance scalability," in *Proceedings of the HPC Asia 2023 Workshops*, ser. HPCAsia '23 Workshops. New York, NY, USA: Association for Computing Machinery, 2023, p. 1–14. [Online]. Available: https://doi.org/10.1145/3581576.3581577 VI-B1

[81] H. Frank and B. Sven, "Wekafs architecture white paper," 2018. [Online]. Available: https://www.beegfs.io/docs/whitepapers/Introduction_to_BeeGFS_by_ThinkParQ.pdf VI-B1

[82] J. Terrace and M. J. Freedman, "Object storage on CRAQ: High-Throughput chain replication for Read-Mostly workloads," in *2009 USENIX Annual Technical Conference (USENIX ATC 09)*. San Diego, CA: USENIX Association, Jun. 2009. [Online]. Available: https://www.usenix.org/conference/usenix-09/object-storage-craq-high-throughput-chain-replication-read-mostly-workloads VI-B3

[83] E. B. Nightingale, J. Elson, J. Fan, O. Hofmann, J. Howell, and Y. Suzue, "Flat datacenter storage," in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'12. USA: USENIX Association, 2012, p. 1–15. VI-B3

[84] DeepSeek-AI, "Deepseek api introduces context caching on disk, cutting prices by an order of magnitude," 2024. [Online]. Available: https://platform.deepseek.com/api-docs/news/news0802 VI-B4

[85] K. Liu, Z. Jiang, J. Zhang, H. Wei, X. Zhong, L. Tan, T. Pan, and T. Huang, "Hostping: Diagnosing intra-host network bottlenecks in RDMA servers," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 15–29. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/liu-kefei VII-B

[86] Y. He, M. Hutton, S. Chan, R. De Gruijl, R. Govindaraju, N. Patil, and Y. Li, "Understanding and Mitigating Hardware Failures in Deep Learning Training Systems," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*. Orlando FL USA: ACM, Jun. 2023, pp. 1–16. [Online]. Available: https://dl.acm.org/doi/10.1145/3579371.3589105 VII-C

[87] S. Wang, G. Zhang, J. Wei, Y. Wang, J. Wu, and Q. Luo, "Understanding Silent Data Corruptions in a Large Production CPU Population," in *Proceedings of the 29th Symposium on Operating Systems Principles*. Koblenz Germany: ACM, Oct. 2023, pp. 216–230. [Online]. Available: https://dl.acm.org/doi/10.1145/3600006.3613149 VII-C

[88] NVIDIA, "This document explains what xid messages are, and is intended to assist system administrators, developers, and faes in understanding the meaning behind these messages as an aid in analyzing and resolving gpu-related problems." [Online]. Available: https://docs.nvidia.com/deploy/xid-errors/ VII-C1

[89] "Priority flow control : Build reliable layer 2 infrastructure," 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:42645413 VIII-A

[90] S. Yan, G. Min, and I. Awan, "Performance analysis of credit-based flow control in infiniband interconnection networks," *Journal of Interconnection Networks*, vol. 07, no. 04, pp. 535–548, 2006. [Online]. Available: https://doi.org/10.1142/S0219265906001843 VIII-A

[91] E. Nepolo and G.-A. Lusilao Zodi, "A predictive ecmp routing protocol for fat-tree enabled data centre networks," in *2021 15th International Conference on Ubiquitous Information Management and Communication (IMCOM)*, 2021, pp. 1–8. VIII-A

[92] J. Rocher-González, E. G. Gran, S.-A. Reinemo, T. Skeie, J. Escudero-Sahuquillo, P. J. García, and F. J. Q. Flor, "Adaptive routing in infiniband hardware," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2022, pp. 463–472. VIII-A

[93] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, p. 523–536, aug 2015. [Online]. Available: https://doi.org/10.1145/2829988.2787484 VIII-A

[94] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu, "Hpcc: high precision congestion control," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 44–58. [Online]. Available: https://doi.org/10.1145/3341302.3342085 VIII-A

[95] R. Mittal, V. T. Lam, N. Dukkipati, E. R. Blem, H. M. G. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:9676894 VIII-A

[96] Q. Hu, Z. Ye, Z. Wang, G. Wang, M. Zhang, Q. Chen, P. Sun, D. Lin, X. Wang, Y. Luo *et al.*, "Characterization of large language model development in the datacenter," in *21st USENIX Symposium on Net-*

worked Systems Design and Implementation (NSDI 24), 2024, pp. 709–729. VIII-D

[97] DeepSeek-AI, X. Bi, D. Chen, G. Chen, S. Chen, D. Dai, C. Deng, H. Ding, K. Dong, Q. Du, Z. Fu, H. Gao, K. Gao, W. Gao, R. Ge, K. Guan, D. Guo, J. Guo, G. Hao, Z. Hao, Y. He, W. Hu, P. Huang, E. Li, G. Li, J. Li, Y. Li, Y. K. Li, W. Liang, F. Lin, A. X. Liu, B. Liu, W. Liu, X. Liu, X. Liu, Y. Liu, H. Lu, S. Lu, F. Luo, S. Ma, X. Nie, T. Pei, Y. Piao, J. Qiu, H. Qu, T. Ren, Z. Ren, C. Ruan, Z. Sha, Z. Shao, J. Song, X. Su, J. Sun, Y. Sun, M. Tang, B. Wang, P. Wang, S. Wang, Y. Wang, Y. Wang, T. Wu, Y. Wu, X. Xie, Z. Xie, Z. Xie, Y. Xiong, H. Xu, R. X. Xu, Y. Xu, D. Yang, Y. You, S. Yu, X. Yu, B. Zhang, H. Zhang, L. Zhang, L. Zhang, M. Zhang, M. Zhang, W. Zhang, Y. Zhang, C. Zhao, Y. Zhao, S. Zhou, S. Zhou, Q. Zhu, and Y. Zou, "DeepSeek LLM: Scaling Open-Source Language Models with Longtermism," Jan. 2024, arXiv:2401.02954 [cs]. [Online]. Available: http://arxiv.org/abs/2401.02954 X

[98] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. K. Li, Y. Wu, and D. Guo, "DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models," Apr. 2024, arXiv:2402.03300 [cs]. [Online]. Available: http://arxiv.org/abs/2402.03300 X

[99] H. Lu, W. Liu, B. Zhang, B. Wang, K. Dong, B. Liu, J. Sun, T. Ren, Z. Li, H. Yang, Y. Sun, C. Deng, H. Xu, Z. Xie, and C. Ruan, "DeepSeek-VL: Towards Real-World Vision-Language Understanding," Mar. 2024, arXiv:2403.05525 [cs]. [Online]. Available: http://arxiv.org/abs/2403.05525 X

[100] DeepSeek-AI, Q. Zhu, D. Guo, Z. Shao, D. Yang, P. Wang, R. Xu, Y. Wu, Y. Li, H. Gao, S. Ma, W. Zeng, X. Bi, Z. Gu, H. Xu, D. Dai, K. Dong, L. Zhang, Y. Piao, Z. Gou, Z. Xie, Z. Hao, B. Wang, J. Song, D. Chen, X. Xie, K. Guan, Y. You, A. Liu, Q. Du, W. Gao, X. Lu, Q. Chen, Y. Wang, C. Deng, J. Li, C. Zhao, C. Ruan, F. Luo, and W. Liang, "DeepSeek-Coder-V2: Breaking the Barrier of Closed-Source Models in Code Intelligence," Jun. 2024, arXiv:2406.11931 [cs]. [Online]. Available: http://arxiv.org/abs/2406.11931 X

[101] H. Xin, Z. Z. Ren, J. Song, Z. Shao, W. Zhao, H. Wang, B. Liu, L. Zhang, X. Lu, Q. Du, W. Gao, Q. Zhu, D. Yang, Z. Gou, Z. F. Wu, F. Luo, and C. Ruan, "DeepSeek-Prover-V1.5: Harnessing Proof Assistant Feedback for Reinforcement Learning and Monte-Carlo Tree Search," Aug. 2024, arXiv:2408.08152 [cs]. [Online]. Available: http://arxiv.org/abs/2408.08152 X

Appendix: Artifact Description/Artifact Evaluation

Artifact Description (AD)

I. Overview of Contributions and Artifacts

A. Paper's Main Contributions

- C_1 Our PCIe A100 architecture managed to attain approximately 80% of NVIDIA DGX A100 performance. TF32 GEMM performance: A100 PCIe 107TFLOPS vs DGX-A100 131TFLOPS. FP16 GEMM performance: A100 PCIe 220TFLOPS vs DGX-A100 263TFLOPS.
- C_2 Our PCIe A100 architecture costs 60% energy consumption of DGX-A100. 2500 Watts A100 PCIe Node vs 4200 Watts DGX-A100.
- C_3 Strong Scalability: HFReduce can achieve an inter-node bandwidth of 7-8GB/s when performing an allreduce test with 186MiB data, scaling from 16 to 1440 GPUs. In the same scenario, NCCL's inter-node bandwidth is only 2-4GB/s.
- C_4 Strong Scalability: Inter-node bandwidth of HFReduce with NVLink can achieve more than 10GB/s. In Cross Fat-Tree Zone operations, HFReduce runs very effectively with minimal performance decay.
- C_5 Weak Scalability: Training VGG16 with HFReduce takes only half the time compared to using the Torch Distributed Data Parallel (DDP) NCCL backend. It achieves nearly 88% parallel scalability when scaling from 32 GPUs to 512.
- C_6 Strong Scalability: When scaling up from 64 GPUs to 512 GPUs, the step time of training LLaMa-13B decreases from 64.118s to 9.717s, achieving a parallel efficiency of 91% at 256 GPUs.
- C_7 Weak Scalability: When training GPT2-medium and scaling from 16 to 128 GPUs, we achieve a parallel scalability of 95%.
- C_8 Our distributed file storage system has achieved a total read throughput of 8TB/s with 180 storage node, 2880 PCIe4.0x4 NVMe SSDs and 360 Mellanox InfiniBand CX6 200G HCAs.

B. Computational Artifacts

Please see files with DOI link:
https://doi.org/10.5281/zenodo.13327567

Download `sc_firefly2_ad_ae.tgz` and extract it. You will get eight folders named `A1` to `A8`, each corresponding to one artifact as follows:

- A_1 sc_firefly2_ad_ae/A1
- A_2 sc_firefly2_ad_ae/A2
- A_3 sc_firefly2_ad_ae/A3
- A_4 sc_firefly2_ad_ae/A4
- A_5 sc_firefly2_ad_ae/A5
- A_6 sc_firefly2_ad_ae/A6
- A_7 sc_firefly2_ad_ae/A7
- A_8 sc_firefly2_ad_ae/A8

Artifact ID	Contributions Supported	Related Paper Elements
A_1	C_1	Table II
A_2	C_2	Table II
A_3	C_3	Figure 7(a)
A_4	C_4	Figure 7(b)
A_5	C_5	Figure 8(a)
A_6	C_6	Figure 9(a)
A_7	C_7	Figure 8(b)
A_8	C_8	Section VI-B2

II. Artifact Identification

A. Computational Artifact A_1

Relation To Contributions

Tested GPU GEMM computation performance by running GEMM calculation code through Torch.

The GEMM of the PCIe card can achieve about 80% of the performance of DGX A100.

Expected Results

TF32 GEMM performance:
1) A100 PCIe: 107TFLOPS;
2) DGX A100: 131TFLOPS.
FP16 GEMM performance:
1) A100 PCIe: 220TFLOPS;
2) DGX A100: 263TFLOPS.

Expected Reproduction Time (in Minutes)

Artifact Setup: 5 minutes.
Artifact Execution: 5 minutes.
Artifact Analysis: 1 minute.

Artifact Setup (incl. Inputs)

Hardware: 1) DGX A100: epyc 7742 * 2, DDR4-3200*16, A100 SXM4 40GB * 8; 2) PCIe A100: epyc Rome 7502 *2, DDR4-3200*16, A100 PCIe 40GB * 8, IB HDR 200G * 1.

Software: Ubuntu20.04 / Python3.8 / PyTorch 1.12 / CUDA 11.3.

Datasets / Inputs: N/A.

Installation and Deployment: Used HaiPlatform ubuntu2004-cu113 image, source haienv 202111.

Artifact Execution

Refer to the files at `sc_firefly2_ad_ae/A1`.

Obtained results by executing "python bench_gemm.py" on A100 and DGX A100 separately.

Artifact Analysis (incl. Outputs)

Type: torch.float32 N=16384;
time_per_iter: 70399.85 usecs;
average performance: 124.94 TFLOPS.

B. Computational Artifact A_2

Relation To Contributions

As shown in table 1:
1) PCIe A100: 2500 Watts;
2) DGX A100: 4200 Watts.

Power consumption is measured by ResNet50 mdoel training workload.

A ResNet (Residual Neural Network) is a seminal deep learning model in which the weight layers learn residual functions with reference to the layer inputs. ResNet is widely used and very representative.

Expected Results

System power:
1) PCIe A100: about 2500 Watts;
2) DGX A100: about 4200 Watts.

Expected Reproduction Time (in Minutes)

Artifact Setup: 5 minutes.
Artifact Execution: 5 minutes.
Artifact Analysis: 1 minute.

Artifact Setup (incl. Inputs)

Hardware: 1) DGX A100: epyc 7742 * 2, DDR4-3200*16, A100 SXM4 40GB * 8; 2) PCIe A100: epyc Rome 7502 *2, DDR4-3200*16, A100 PCIe 40GB * 8, IB HDR 200G * 1.

Software: Ubuntu 20.04 / Python 3.8 / PyTorch 1.12 / CUDA 11.3.

Datasets / Inputs: ImageNet.

Installation and Deployment: Used HaiPlatform ubuntu2004-cu113 image, source haienv 202111.

Artifact Execution

Refer to the files at `sc_firefly2_ad_ae/A2`.
1) Allocate one node for testing.
2) Run "python resnet.py".
3) When the training speed is stable, use ipmitool to read the overall machine power consumption.
3.a) PCIe A100: Run "ipmitool sensor get Total_Power" and view the result.
3.b) DGX A100: Run "ipmitool sensor get PWR_SYSTEM" and view the result.

Artifact Analysis (incl. Outputs)

1) PCIe A100:
1.a) Sensor ID: Total_Power;
1.b) Sensor Reading: 2580 (+/- 0) Watts;
2) DGX A100:
2.a) Sensor ID: PWR_SYSTEM;
2.b) Sensor Reading: 4398 (+/- 0) Watts.

C. Computational Artifact A_3

Relation To Contributions

Strong Scalability: HFReduce can achieve an inter-node bandwidth of 7-8GB/s when performing an allreduce test with 186MiB data, scaling from 16 to 1440 GPUs. In the same scenario, NCCL's inter-node bandwidth is only 2-4GB/s.

Expected Results

GPU Count	HFReduce	NCCL
16	8.10	4.01
32	8.03	4.41
64	7.95	4.29
128	7.78	3.73
256	7.12	4.83
512	7.33	4.62
1024	6.31	1.63
1440	6.99	1.65

Expected Reproduction Time (in Minutes)

Artifact Setup: 5 minutes.
Artifact Execution: 20 minutes.
Artifact Analysis: 5 minutes.

Artifact Setup (incl. Inputs)

Hardware: PCIe A100: epyc Rome 7502 *2, DDR4-3200*16, A100 PCIe 40GB * 8, IB HDR 200G * 1.

Software: Ubuntu 20.04 / Python 3.8 / PyTorch 1.12 / CUDA 11.3.

Datasets / Inputs: N/A.

Installation and Deployment: Used HaiPlatform ubuntu2004-cu113 image, source haienv 202111.

Artifact Execution

Refer to the files at `sc_firefly2_ad_ae/A3`.
See bench_allreduce.py for communication bandwidth test code.

You could login with a platform account and run `submit.sh` script for submitting cluster tasks.

Execution: Run the task submission script, automatically submit tasks from 16 to 1440 GPUs; wait for the tasks to complete, and check the individual execution results.

Artifact Analysis (incl. Outputs)

View the code test results.

NCCL uses ringallreduce for allreduce communication, and the amount of data sent over the network is 2*(N-1)/N of the data that needs to be reduced.

To unify the calculation of network bandwidth, we use 2*(N-1)/N of the algorithm bandwidth as the network bandwidth.

D. Computational Artifact A_4

Relation To Contributions

Strong Scalability: Inter-node bandwidth of HFReduce with NVLink can achieve more than 10GB/s. In Cross Fat-Tree Zone operations, HFReduce runs very effectively with minimal performance decay.

Expected Results

GPU Count	HFReduce with NVLink	HFReduce cross Fat-Tree Zone
16	19.18	N/A
32	13.00	N/A
64	12.07	N/A
128	10.30	9.27
256	9.66	8.63
512	8.56	8.06
1024	7.99	7.75

Expected Reproduction Time (in Minutes)

Artifact Setup: 5 minutes.
Artifact Execution: 20 minutes.
Artifact Analysis: 5 minutes.

Artifact Setup (incl. Inputs)

Hardware: PCIe A100: epyc Rome 7502 *2, DDR4-3200*16, A100 PCIe 40GB * 8, IB HDR 200G * 1.
Software: Ubuntu 20.04 / Python 3.8 / PyTorch 1.12 / CUDA 11.3.
Datasets / Inputs: N/A.
Installation and Deployment: Used HaiPlatform ubuntu2004-cu113 image, source haienv 202111.

Artifact Execution

Refer to the files at `sc_firefly2_ad_ae/A4`.
See submit.sh for running communication bandwidth test in Fire-Flyer 2 Platform.
You could login with a platform account and run `submit.sh` script for submitting cluster tasks.
Execution: Run the task submission script, automatically submit tasks from 16 to 1024 GPUs; wait for the tasks to complete, and check individual execution results.

Artifact Analysis (incl. Outputs)

View the code test results.
Since the allreduce algorithm first performs a reduce, and then broadcasts the reduced result to all nodes, the amount of data sent over the network is twice the amount of data that needs to be reduced.
To unify the calculation of network bandwidth, we use the same formula as in the previous step: 2*(N-1)/N of the algorithm bandwidth is taken as the network bandwidth.

E. Computational Artifact A_5

Relation To Contributions

Weak Scalability: Training VGG16 with HFReduce takes only half the time compared to using the Torch Distributed Data Parallel (DDP) NCCL backend. It achieves nearly 88% parallel scalability when scaling from 32 GPUs to 512.
The VGG model is based on the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" and VGG16 is a classical model for image classification tasks.

Expected Results

GPU Count	HFReduce	NCCL
32	0.13	0.27
64	0.133	0.271
128	0.139	0.279
256	0.149	0.287
512	0.15	0.288

Expected Reproduction Time (in Minutes)

Artifact Setup: 5 minutes.
Artifact Execution: 20 minutes.
Artifact Analysis: 5 minutes.

Artifact Setup (incl. Inputs)

Hardware: PCIe A100: epyc Rome 7502 *2, DDR4-3200 *16, A100 PCIe 40GB * 8, IB HDR 200G * 1.
Software: Ubuntu 20.04 / Python 3.8 / PyTorch 1.12 / CUDA 11.3
Datasets / Inputs: ImageNet.
Installation and Deployment: Used HaiPlatform ubuntu2004-cu113 image, source haienv 202207.

Artifact Execution

Refer to the files at `sc_firefly2_ad_ae/A5`.
See train_vgg16.py for the test code.
You could login with a platform account and run `submit.sh` script for submitting cluster tasks.
Execution: Run the task submission script, automatically submit tasks from 32 to 512 GPUs; wait for the tasks to complete, and check individual execution results.

Artifact Analysis (incl. Outputs)

View the execution results, it will output second/step.

F. Computational Artifact A_6

Relation To Contributions

Strong Scalability: When scaling up from 64 GPUs to 512 GPUs, the step time of training LLaMA 13B decreases from 64.118s to 9.717s, achieving a parallel efficiency of 91% at 256 GPUs. LLaMA (Large Language Model Meta AI) is a family of autoregressive large language models (LLMs), released and open source by Meta AI.

Expected Results

GPU Count	ZBPP + PPO
64	64.12
128	32.51
256	17.45
512	9.72

Expected Reproduction Time (in Minutes)

Artifact Setup: 5 minutes.
Artifact Execution: 20 minutes.
Artifact Analysis: 5 minutes.

Artifact Setup (incl. Inputs)

Hardware: PCIe A100: epyc Milan 7513 *2, DDR4-3200*16, A100 PCIe 40GB * 8, IB HDR 200G * 1.
Software: Ubuntu 20.04 / Python 3.8 / PyTorch 2.01 / CUDA 11.7.
Datasets / Inputs: Wikitext-103.
Installation and Deployment: Used Hai-Platform ubuntu2004-cu113 image and our hai-llm framework.

Artifact Execution

Refer to the files at `sc_firefly2_ad_ae/A6`.
The hai-llm framework is not yet open source.
You may need login with a platform account and run `submit.sh` script for submitting cluster tasks.
Execution: Run the task submission script, automatically submit tasks from 64 to 512 GPUs; wait for the tasks to complete, and check individual execution results.

Artifact Analysis (incl. Outputs)

View the execution results, it will output second/step.

G. Computational Artifact A_7

Relation To Contributions

Weak Scalability: When training GPT2-medium and scaling from 16 to 128 GPUs, we achieve a parallel scalability of 95%.
GPT-2, developed by OpenAI, is a large-scale unsupervised language model that uses transformer-based deep learning techniques to generate human-like text. It's a classic transformer-based model in NLP.

Expected Results

GPU Count	HaiScale	Torch FSDP
16	0.57	0.86
32	0.58	0.985
64	0.598	0.875
128	0.595	0.99

Expected Reproduction Time (in Minutes)

Artifact Setup: 5 minutes.
Artifact Execution: 20 minutes.
Artifact Analysis: 5 minutes.

Artifact Setup (incl. Inputs)

Hardware: PCIe A100: epyc Rome 7502 *2, DDR4-3200*16, A100 PCIe 40GB * 8, IB HDR 200G * 1.
Software: Ubuntu 20.04 / Python 3.8 / PyTorch 1.12 / CUDA 11.3.
Datasets / Inputs: Wikitext-103.
Installation and Deployment: Used HaiPlatform ubuntu2004-cu113 image, source haienv 202207.

Artifact Execution

Refer to the files at `sc_firefly2_ad_ae/A7`.
See train_haiscale.py and train_torch_fsdp.py for the test code. You could login with a platform account and run `submit.sh` script for submitting cluster tasks.
Execution: Run the task submission script, automatically submit tasks from 16 to 128 GPUs; wait for the tasks to complete, and check individual execution results.

Artifact Analysis (incl. Outputs)

View the execution results, it will output second/step.

H. Computational Artifact A_8

Relation To Contributions

Our distributed file storage systemhas achieved a total read throughput of 8TB/s with 180 storage node, 2880 PCIe4.0x4 NVMe SSDs and 360 InfiniBand CX6 200G HCAs.

Expected Results

Run fio, the total throughput consistently surpasses 8TB/s.

Expected Reproduction Time (in Minutes)

Artifact Setup: 5 minutes.
Artifact Execution: 10 minutes.
Artifact Analysis: 5 minutes.

Artifact Setup (incl. Inputs)

Hardware: 1) Compute Node: PCIe A100: epyc Rome 7502 *2, DDR4-3200*16, A100 PCIe 40GB * 8, IB HDR 200G * 1; 2) Storage Node: EPYC 7742,DDR4-3200*8, IB HDR 200G * 2, 15.36 NVMe SSD * 16.
Software: Ubuntu 20.04 / Customized version of fio with 3fs custom API / 3FS version 231023-rel-0-70409-8ea13fe7.
Datasets / Inputs: N/A.
Installation and Deployment: The cluster has 180 3FS storage nodes.

Artifact Execution

Refer to the files at `sc_firefly2_ad_ae/A8`.
You may need login with a platform account and run submit.sh script for submitting cluster tasks.
Execution: Run customized version of `fio` with 3FS custom API on 400 3FS client nodes; wait for 10 minutes.

Artifact Analysis (incl. Outputs)

View the 3FS monitoring dashboard.

Artifact Evaluation (AE)

Evaluation scripts and codes are availble at https://doi.org/10.5281/zenodo.13327567.

We provide the necessary compute resources and nodes for your evaluations. If you would like to evaluate on our Fire-Flyer 2 AIHPC Platform, please contact us via email at ly.zhang@high-flyer.cn to request a platform account and further support.

Please see files with DOI link:

https://doi.org/10.5281/zenodo.13327567

Download `sc_firefly2_ad_ae.tgz` and extract it. You will get eight folders named `A1` to `A8`, each corresponding to one artifact as follows:

A_1	sc_firefly2_ad_ae/A1
A_2	sc_firefly2_ad_ae/A2
A_3	sc_firefly2_ad_ae/A3
A_4	sc_firefly2_ad_ae/A4
A_5	sc_firefly2_ad_ae/A5
A_6	sc_firefly2_ad_ae/A6
A_7	sc_firefly2_ad_ae/A7
A_8	sc_firefly2_ad_ae/A8

A. Computational Artifact A_1

Artifact Setup (incl. Inputs)

Hardware:

- DGX A100: 2x EPYC 7742 CPUs, 16x DDR4-3200, 8x NVIDIA A100 SXM4 40GB GPUs, 9x IB HDR 200G
- PCIe A100: 2x EPYC 7502 CPUs, 16x DDR4-3200, 8x NVIDIA A100 PCIe 40GB GPUs, 1x IB HDR 200G

Software: Ubuntu 20.04 / Python 3.8 / PyTorch 1.12 / CUDA 11.3

Datasets / Inputs: N/A.

Installation and Deployment: Utilized the Fire-Flyer 2 HaiPlatform ubuntu2004-cu113 image, and source haienv 202111

Artifact Execution

Refer to the files at `sc_firefly2_ad_ae/A1`.

Results were obtained by executing `python bench_gemm.py` on both the A100 and DGX A100 systems separately.

Artifact Analysis (incl. Outputs)

The `bench_gemm.py` script produced the following outputs:

- DGX A100:
  - Type: `torch.float32`, N=8192, Average Performance: 130.17 TFLOPS
  - Type: `torch.bfloat16`, N=8192, Average Performance: 262.87 TFLOPS
- PCIe A100:
  - Type: `torch.float32`, N=8192, Average Performance: 106.37 TFLOPS
  - Type: `torch.bfloat16`, N=8192, Average Performance: 226.52 TFLOPS

The PCIe A100 system achieved over 80% of the performance of the DGX A100 system.

B. Computational Artifact A_2

Artifact Setup (incl. Inputs)

Hardware:

- DGX A100: 2x EPYC Rome 7742 CPUs, 16x DDR4-3200 RAM, 8x NVIDIA A100 SXM4 40GB GPUs, 9x IB HDR 200G HCAs;
- PCIe A100: 2x EPYC Rome 7502 CPUs, 16x DDR4-3200 RAM, 8x NVIDIA A100 PCIe 40GB GPUs, 1x IB HDR 200G HCA.

Software: Ubuntu 20.04 / Python 3.8 / PyTorch 1.12 / CUDA 11.3

Datasets / Inputs: ImageNet https://www.image-net.org/download.php

Installation and Deployment: Utilized the Fire-Flyer 2 HaiPlatform ubuntu2004-cu113 image, and source haienv 202111

Artifact Execution

Refer to the files at `sc_firefly2_ad_ae/A2`.

1) Allocate one node for running the test script `python train.py`, or simply run `hfai_run.sh`.
2) When the training speed stabilizes, use `ipmitool` to measure the overall machine power consumption.
   a) PCIe A100: Run `ipmitool sensor get Total_Power` and view the result.
   b) DGX A100: Run `ipmitool sensor get PWR_SYSTEM` and view the result.

Artifact Analysis (incl. Outputs)

1) Allocate one node for running the test script `python train.py`, or simply run `hfai_run.sh`.
2) When the training speed stabilizes, use `ipmitool` to measure the overall machine power consumption.
   a) PCIe A100:
      - Sensor ID: `Total_Power`
      - Sensor Reading: 2580 ($\pm$ 0) Watts
   b) DGX A100:
      - Sensor ID: `PWR_SYSTEM`
      - Sensor Reading: 4398 ($\pm$ 0) Watts

C. Computational Artifact A_3

Artifact Setup (incl. Inputs)

Hardware: PCIe A100: 2x EPYC Rome 7502 CPUs, 16x DDR4-3200 RAM, 8x NVIDIA A100 PCIe 40GB GPUs, 1x IB HDR 200G HCA.

Software: Ubuntu 20.04 / Python 3.8 / PyTorch 1.12 / CUDA 11.3

Datasets / Inputs: N/A.

Installation and Deployment: Utilized the Fire-Flyer 2 HaiPlatform ubuntu2004-cu113 image, and source haienv 202111

Artifact Execution

Refer to the files at `sc_firefly2_ad_ae/A3`.

See bench_allreduce.py for communication bandwidth test code.

You could login with a platform account and run submit.sh script for submitting cluster tasks.

Execution: Run the task submission script, automatically submit tasks from 16 to 1440 GPUs; wait for the tasks to complete, and check the individual execution results.

Artifact Analysis (incl. Outputs)

Each task submitted by `submit.sh` would output results similar to the following:

```
[RANK]: 0 , [hfreduce] size: 1.84e+02 MiB, algoBW 4.634GB/s, Network Bandwidth 8.688367340653281GB/s
```

This **Network Bandwidth** is the expected result. All test cases should get network bandwidth results as follows:

GPU Count	HFReduce	NCCL
16	8.10	4.01
32	8.03	4.41
64	7.95	4.29
128	7.78	3.73
256	7.12	4.83
512	7.33	4.62
1024	6.31	1.63
1440	6.99	1.65

Algorithm Bandwidth and Network Bandwidth:

NCCL (NVIDIA Collective Communications Library) uses the ring-allreduce algorithm for allreduce communication. The amount of data sent over the network is $\frac{2(N-1)}{N}$ times the data that needs to be reduced.

To unify the calculation of network bandwidth, we use $\frac{2(N-1)}{N}$ of the algorithm bandwidth as the network bandwidth.

D. Computational Artifact A_4

Artifact Setup (incl. Inputs)

Hardware: PCIe A100: 2x EPYC Rome 7502 CPUs, 16x DDR4-3200 RAM, 8x NVIDIA A100 PCIe 40GB GPUs, 1x IB HDR 200G HCA.

Software: Ubuntu 20.04 / Python 3.8 / PyTorch 1.12 / CUDA 11.3

Datasets / Inputs: N/A.

Installation and Deployment: Utilized the Fire-Flyer 2 Hai-Platform ubuntu2004-cu113 image, and source haienv 202111

Artifact Execution

Refer to the files at `sc_firefly2_ad_ae/A4` .

See `submit.sh` for running communication bandwidth test in Fire-Flyer 2 Platform.

You could login with a platform account and run `submit.sh` script for submitting cluster tasks.

Execution: Run the task submission script, automatically submit tasks from 16 to 1024 GPUs; wait for the tasks to complete, and check individual execution results.

Artifact Analysis (incl. Outputs)

Each task submitted by `submit.sh` would output results similar to the following:

```
Algorithm bandwidth: 8.956921GiB/s, Network Bandwidth 16.794226875GB/s
```

This **Network Bandwidth** is the expected result. All test cases should get network bandwidth results as follows:

GPU Count	HFReduce with NVLink	HFReduce cross Fat-Tree Zone
16	19.18	N/A
32	13.00	N/A
64	12.07	N/A
128	10.30	9.27
256	9.66	8.63
512	8.56	8.06
1024	7.99	7.75

E. Computational Artifact A_5

Artifact Setup (incl. Inputs)

Hardware: PCIe A100: 2x EPYC Rome 7502 CPUs, 16x DDR4-3200 RAM, 8x NVIDIA A100 PCIe 40GB GPUs, 1x IB HDR 200G HCA.

Software: Ubuntu 20.04 / Python 3.8 / PyTorch 1.12 / CUDA 11.3

Datasets / Inputs: ImageNet https://www.image-net.org/download.php

Installation and Deployment: Utilized the Fire-Flyer 2 Hai-Platform ubuntu2004-cu113 image, and source haienv 202207

Artifact Execution

Refer to the files at `sc_firefly2_ad_ae/A5`.

See `train_vgg16.py` for the test code.

You could login with a platform account and run `submit.sh` script for submitting cluster tasks.

Execution: Run the task submission script, automatically submit tasks from 32 to 512 GPUs; wait for the tasks to complete, and check individual execution results.

Artifact Analysis (incl. Outputs)

Each task submitted by `submit.sh` would output results similar to the following:

```
Step 20: 0.1321681 second/step.
```

This **second/step** is the expected result. All test cases should get **second/step** results as follows:

GPU Count	HFReduce	NCCL
32	0.13	0.27
64	0.133	0.271
128	0.139	0.279
256	0.149	0.287
512	0.15	0.288

F. Computational Artifact A_6

Artifact Setup (incl. Inputs)

Hardware: PCIe A100: 2x EPYC Milan 7513 CPUs, 16x DDR4-3200 RAM, 8x NVIDIA A100 PCIe 40GB GPUs, 1x IB HDR 200G HCA.

Software: Ubuntu 20.04 / Python 3.8 / PyTorch 2.01 / CUDA 11.7

Datasets / Inputs: Wikitext-103 https://paperswithcode.com/dataset/wikitext-103

Installation and Deployment: Utilized the Fire-Flyer 2 HAI-Platform ubuntu2004-cu113 image, and source haienv 202207. It is necessary to use our hai-llm framework.

Artifact Execution

Refer to the files at `sc_firefly2_ad_ae/A6`.

The hai-llm framework is not yet open source. You may need login with a platform account and run `submit.sh` script for submitting cluster tasks.

Execution: Run the task submission script, automatically submit tasks from 64 to 512 GPUs; wait for the tasks to complete, and check individual execution results.

Artifact Analysis (incl. Outputs)

Each task submitted by `submit.sh` would output results similar to the following:

```
RANK 0, step 2, consumed_tokens 0.025
B, loss 0.00000, time 64.089, global_bs
4096, bs 256, 159.695 TFLOPS, util 51.18%
, 63.911123 samples/s
```

This **time 64.089** is the expected result means 64.089 second/s. All test cases should get **second/step** results as follows:

GPU Count	ZBPP + PPO
64	64.12
128	32.51
256	17.45
512	9.72

G. Computational Artifact A_7

Artifact Setup (incl. Inputs)

Hardware: PCIe A100: 2x EPYC Rome 7502 CPUs, 16x DDR4-3200 RAM, 8x NVIDIA A100 PCIe 40GB GPUs, 1x IB HDR 200G.

Software: Ubuntu 20.04 / Python 3.8 / PyTorch 1.12 / CUDA 11.3

Datasets / Inputs: Wikitext-103 https://paperswithcode.com/dataset/wikitext-103

Installation and Deployment: Utilized the Fire-Flyer 2 HAI-Platform ubuntu2004-cu113 image, and source haienv 202207.

Artifact Execution

Refer to the files at `sc_firefly2_ad_ae/A7`.

See code files `train_haiscale.py` and `train_torch_fsdp.py` for the running test.

You could login with a platform account and run `submit.sh` script for submitting cluster tasks.

Execution: Run the task submission script, automatically submit tasks for from 16 to 128 GPUs; wait for the tasks to complete, and check individual execution results.

Artifact Analysis (incl. Outputs)

Each task submitted by `submit.sh` would output results similar to the following:

```
RANK 0, epoch 1, step 60/929, lr
0.000032 loss 8.383, Speed: 0.87123
seconds/step
```

This **second/step** is the expected result. All test cases should get **second/step** results as follows:

GPU Count	HaiScale	Torch FSDP
16	0.57	0.86
32	0.58	0.985
64	0.598	0.875
128	0.595	0.99

H. Computational Artifact A_8

Artifact Setup (incl. Inputs)

Hardware:

- Compute Node: PCIe A100: 2x EPYC Rome 7502 CPUs, 16x DDR4-3200 RAM, 8x A100 PCIe 40GB GPUs, 1x IB HDR 200G HCA;
- Storage Node: EPYC 7742 CPU, 8x DDR4-3200 RAM, 2x IB HDR 200G HCAs, 16x 15.36TB NVMe SSDs.

Software: Ubuntu 20.04 / Customized version of fio with 3FS custom API / 3FS version 231023-rel-0-70409-8ea13fe7

Datasets / Inputs: N/A.

Installation and Deployment: The cluster has 180 3FS storage nodes. Submit 400 node tasks as clients to perform the fio read benchmark.

Artifact Execution

Refer to the files at `sc_firefly2_ad_ae/A8`.

You may need to log in with a platform account and run the `submit.sh` script to submit cluster tasks. Execute the customized version of fio with the 3FS custom API on 400 3FS client nodes and wait for 10 minutes.

Artifact Analysis (incl. Outputs)

View the 3FS monitoring dashboard to evaluate the total read bandwidth.