Du JS, Li DS, Wen YP *et al.* SAIH: A scalable evaluation methodology for understanding AI performance trend on HPC systems. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 39(2): 384-400 Mar. 2024. DOI: 10.1007/s11390-023-1840-y

SAIH: A Scalable Evaluation Methodology for Understanding AI Performance Trend on HPC Systems

Jiang-Su Du (杜江溯), Dong-Sheng Li (李东升), Ying-Peng Wen (文英鹏), Jia-Zhi Jiang (江嘉治)
Dan Huang (黄　聃), Xiang-Ke Liao (廖湘科), *Fellow, CCF*
and Yu-Tong Lu* (卢宇彤), *Distinguished Member, CCF*

School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, China

E-mail: dujs@mail2.sysu.edu.cn; lidsh25@mail2.sysu.edu.cn; wenyp6@mail2.sysu.edu.cn; jiangjzh6@mail2.sysu.edu.cn
huangd79@mail.sysu.edu.cn; xkliao@nudt.edu.cn; luyutong@mail.sysu.edu.cn

Received August 16, 2021; accepted June 7, 2023.

Abstract Novel artificial intelligence (AI) technology has expedited various scientific research, e.g., cosmology, physics, and bioinformatics, inevitably becoming a significant category of workload on high-performance computing (HPC) systems. Existing AI benchmarks tend to customize well-recognized AI applications, so as to evaluate the AI performance of HPC systems under the predefined problem size, in terms of datasets and AI models. However, driven by novel AI technology, most of AI applications are evolving fast on models and datasets to achieve higher accuracy and be applicable to more scenarios. Due to the lack of scalability on the problem size, static AI benchmarks might be under competent to help understand the performance trend of evolving AI applications on HPC systems, in particular, the scientific AI applications on large-scale systems. In this paper, we propose a scalable evaluation methodology (SAIH) for analyzing the AI performance trend of HPC systems with scaling the problem sizes of customized AI applications. To enable scalability, SAIH builds a set of novel mechanisms for augmenting problem sizes. As the data and model constantly scale, we can investigate the trend and range of AI performance on HPC systems, and further diagnose system bottlenecks. To verify our methodology, we augment a cosmological AI application to evaluate a real HPC system equipped with GPUs as a case study of SAIH. With data and model augment, SAIH can progressively evaluate the AI performance trend of HPC systems, e.g., increasing from 5.2% to 59.6% of the peak theoretical hardware performance. The evaluation results are analyzed and summarized into insight findings on performance issues. For instance, we find that the AI application constantly consumes the I/O bandwidth of the shared parallel file system during its iteratively training model. If I/O contention exists, the shared parallel file system might become a bottleneck.

Keywords high-performance computing (HPC), deep learning, parallel computing, AI framework

1 Introduction

In recent years, AI, especially its deep learning subset, has become one of the key trends in HPC (high-performance computing). Emerging novel AI applications have expedited various scientific discoveries, such as in cosmology[1, 2], physics[3, 4], and cancer diagnosis[5, 6]. Besides, scientists significantly improve domain results with AI technology on HPC systems over their traditional competitors[7].

To support AI applications, emerging HPC systems are designed towards exascale computing capability with better AI performance in consideration. For example, Fugaku in Japan exhibits 2.0x mixed-precision exaFLOPS with ARM-based many-core CPUs and Summit at ORNL has 1.4x mixed-preci-

Regular Paper

This work was supported by the National Natural Science Foundation of China under Grant No. U1811461, the Zhejiang Lab under Grant No. 2021KC0AB04, the Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant No. 2016ZT06D211, the Guangdong Provincial Natural Science Foundation of China under Grant No. 2018B030312002, and the Major Program of Guangdong Basic and Applied Research of China under Grant No. 2019B030302002.

*Corresponding Author

sion exaFLOPS with NVIDIA GPU. While the theoretical peak performances of these HPC systems are appealing, their practical performances on supporting AI training (HPC-AI performance) are still under investigation. In terms of traditional HPC benchmarks, e.g., HPL and HPL-AI[8], they parallelly solve a common task in the HPC domain, the linear equation $Ax = b$, which is rarely adopted in AI applications and thus lacks convincible metrics on AI performance. Compared with traditional HPC simulations, scientific AI applications are much more complex. HPC simulations typically consist of distinct execution phases, such as computation, I/O and collective communication. In contrast, the execution phases of AI applications are pipelined and encapsulated by higher abstractions and longer parallelism hierarchy, which include co-designed programming models, e.g., CUDA, and distributed AI frameworks, e.g., Tensorflow and PyTorch. Such abstraction complexity leads to doubt on the significance of profiling the AI performance by traditional HPC benchmarks.

Moving to existing AI benchmarks on HPC systems, e.g., CORAL-2[①] and MLPerf[9], they are endeavoring to evaluate either the accuracy or the performance (FLOPS) of AI models in the hotspot research domains by adopting representative AI applications with static data and models. This static methodology can reflect a fixed performance relation between a specific application and an HPC system, rather than a range of performance relations. Although MLPerf recently focuses on HPC systems, and adds scientific AI applications, e.g., CosmoFlow[2] and DeepCAM[7], into its benchmark suit, the main idea still follows its previous version, in which both models and datasets are static.

Driven by novel AI technologies, most of AI applications are evolving fast on their problem configurations including both models and datasets, for achieving higher accuracy and being applicable to more scenarios. As proposed in NVIDIA GTC 2021[②], AI model sizes are growing exponentially, on a pace of doubling every two and a half months. Due to the lack of scalability on the problem size, existing AI benchmarks might be incompetent to understand the performance trend of evolving AI applications on HPC systems, in particular, the scientific AI applications on large-scale systems. For instance, in our evaluation, we observe that the aggregate FLOPS of a parallel run of CosmoFlow (3D CNN) is significantly increased (from 5.2% to 59.6% of peak theoretical performance) with the scaling datasets and AI models. It reflects that a kind of AI workloads vary vastly in different configurations and the execution of a static application can only provide a very partial understanding.

To understand the AI performance trend on HPC systems in a more comprehensive way, we propose the Scalable Evaluation Methodology (SAIH). Like the successful HPL LINPACK benchmark on achieving the data and computation scalability by adjusting the size of $Ax = b$, SAIH builds a set of novel mechanisms to satisfy the requirements of data and computation scalability. Additionally, SAIH takes into account the scientific significance of AI workloads by selecting and building representative study cases from domain-specific scientific AI applications. Specifically, the contributions of SAIH are as follows.

- We propose SAIH with scientific significance to evaluate and understand the AI performance range of HPC systems, and it is with both data and model scalability to cover various problem sizes.
- We design a novel strategy for model scalability. By creatively incorporating network architecture search (NAS), the strategy extends the original AI model to more accurate and complex models with scaling computation demands.
- We implement a cosmological AI application as a prototype and rebuild it to an SAIH instance with data and computation scalability. We apply this instance to evaluate a real HPC system as a case study.
- We summarize the performance achievement and qualitative evaluation metrics to illustrate that the SAIH instance can profile the AI performance range of an HPC system on a specific scientific domain as well as revealing evaluation findings about the potential performance issues.

The rest of this paper is organized as follows. We present a survey on scientific AI applications and existing AI benchmarks along with related work in Section 2. A comprehensive methodology on building scalable AI evaluation is presented in Section 3. In Section 4, we present a case study on building an SAIH instance based on a cosmological AI application. Section 5 presents the evaluation and analysis of the cosmological SAIH instance on a real HPC system. Finally, we conclude this work in Section 6.

[①] https://asc.llnl.gov/coral-2-benchmarks, Mar. 2024.
[②] https://www.nvidia.com/gtc/keynote/, Mar. 2024.

2 Background and Related Work

2.1 Scientific AI Applications

In recent years, scientific AI applications are emerging in various domains. In particle physics, Kurth *et al.*[3] firstly attempted to deploy particle image classification on many-core HPC systems and achieved petaFLOPS performance. MENNDL[4] adopts neural network search to find an optimal model for improving the understanding of the electron-beam-matter interactions and real-time image-based feedback, which enables a huge step beyond human capacity towards nano-fabricating materials automatically. In the domain of astronomy, cosmologists[1, 2] take advantage of CNN models to estimate the universal states with higher accuracy than traditional methods. The universal parameters are key factors that determine the evolution of the whole universe and the classification and discovery of astrophysical objects. Medical imaging analysis is the science of solving clinical problems by analyzing images generated in clinical practice. Deep learning techniques are applied in computer aided diagnosis by analyzing the signal data from CT, MRI, DR, etc., including image segmentation[10], detection and classification of abnormality[5, 6]. In bioinformatics, RNN-based deep learning techniques are widely used to detect and recognize genomic patterns[11–13]. In climate changing and weather analysis, CNN and RNN models are used to detect the areas where the abnormal climate changes[7, 14].

2.2 AI Benchmarks

AI-based techniques are continuously driving various application scenarios intelligent, which can be deployed on diverse computing platforms, from large-scale HPC systems to tiny mobile devices. To satisfy the demands of evaluating the AI performance, scientists and engineers have released a number of AI benchmarks covering different application scenarios and platforms.

MLPerf[9] is an AI benchmark suite targeting six AI application scenarios, including recommendation, speech recognition, reinforcement learning, image classification, object detection, and translation. DeepBench[③], released by Baidu, is a micro benchmark set that evaluates basic operations involved in training deep neural networks, including dense matrix multiplies, convolutions, recurrent layers, and all-reduce. While, this benchmark lacks the component-level and application-level evaluation cases. AI Matrix[15], released by Alibaba Group, aims to satisfy the needs of fully characterizing the deep learning workloads in Alibaba's e-commerce environment, including the tasks in computer vision, recommendation, and language processing. HPL-AI Mixed Precision Benchmark[16], released by University of Tennessee, is opting for low-precision (likely 16-bit) accuracy for LU (lower-upper) factorization, and a sophisticated iteration to recover the accuracy lost in factorization. Deep500[17] is a modular benchmark infrastructure for high-performance computing deep learning. It aims at evaluating different framework implementations and different levels of operators. However, it only evaluates a common image classification scenario on the ImageNet[18] dataset, rather than typical scientific scenarios. TBD Suite[19], developed by University of Toronto, is an end-to-end benchmark suite for neural work training. Typically, this work currently covers six major application domains and eight different state-of-the-art models, e.g., image classification, speech recognition. The above AI benchmarks provide evaluations either for classic application scenarios or for the computing operations in deep neural network (DNN) models on data centers and mobile devices. However, they have not covered the scientific AI applications and HPC systems, the configurations of which are significantly distinct from evaluation cases of existing AI benchmark in both datasets and DNN models.

Recently, a number of well-recognized scientific AI applications have been integrated into HPC benchmark suite for evaluating the performance of AI application on HPC systems. For example, CORAL-2 benchmarks[④] cover not only the traditional micro benchmarks, HPC simulations and analysis, but also emerging AI applications, which include the common operations in CNN and RNN as well as the CANDLE benchmark[6] for cancer diagnosis. In 2020, two scientific applications, CosmoFlow[2] and DeepCAM[7] with static models and datasets are included into MLPerf, named as MLPerf-HPC[20], for evaluating the AI performance of large HPC systems. In general, HPC-AI500[21] follows a similar idea to MLPerf-HPC. Moreover, in order to assure a fair ranking, it presents

[③]https://svail.github.io/DeepBench/, Mar. 2024.
[④]https://asc.llnl.gov/coral-2-benchmarks, Mar. 2024.

a new metric named Valid FLOPS, which imposes a penalty on failing to achieve a target training quality. While CORAL-2 benchmarks, MLPerf-HPC, and HPC-AI500 are capable of distributedly evaluating scientific AI applications with static scientific AI models and datasets on HPC systems, they lack the capability of scaling up problem sizes, which is critical to evaluating the potential AI performance of large-scale HPC systems.

3 Methodology

In this section, we introduce the methodology of SAIH. While scientific AI applications run on large-scale HPC systems in data-parallel fashion via sophisticate AI frameworks, such as MPI, Tensorflow and PyTorch, the problem sizes are typically limited by the static training datasets and AI models. Thus, we intend to scale up problem sizes as well as the demands on memory and computation resources by integrating the methods of data augment and AI model augment. Our augment methods take the scientific meaning into account, and higher resolution or higher accuracy can be achieved compared with original applications. As shown in Fig.1, SAIH provides a set of augment methods for transforming scientific AI application into candidate benchmarks with the scalability on problem sizes, and then evaluating the AI performance of HPC systems.

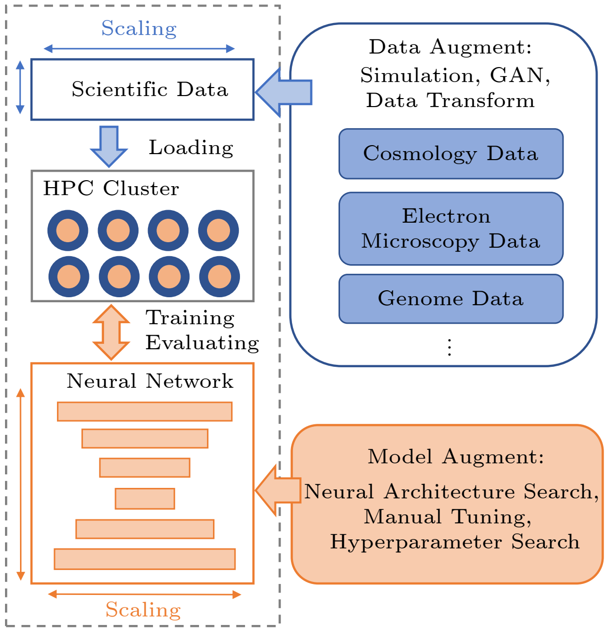


Fig.1. SAIH overview.

3.1 Data Augment

The data augment method in SAIH is designed for both achieving data scalability and keeping scientific meaning. Thus, simulation can be a potential method since it is naturally supported by many scientific applications. Besides, emerging generative adversarial network (GAN)[22] is also a potential method.

Simulation. Many scientific applications simulate natural phenomena and the runtime states of large research facilities. To discover the scientific insights, the outputs of these simulations are the raw data that needs to be further processed by analysis techniques, such as visualization tools and AI based analysis. To efficiently generate hundreds and thousands of data samples, we manage the initial parameters by a centralized parameter server to ensure parameters between different simulations randomly and uniquely. The detailed procedure is in Algorithm 1.

Algorithm 1. Concurrently Generate Data by Simulation

```
1:  Start MPI-based program with total $N$ processes, one for
    parameter server and each of the $(N-1)$ processes runs
    $M$ instances of the simulation.
2:  Initialize MPI.
3:  if $rank = 0$ then
4:    Initialize a centralized parameter server.
5:    Randomly generate initial parameters for simulations.
6:  else
7:    Wait for parameter server ready and prepare $M\times$
      $(N-1)$ parameters.
8:    while $m < M$ do
9:      Load parameters from the server.
10:     Initialize an instance of simulation.
11:     Perform simulation.
12:     Save generated data into the filesystem.
13:     $m = m + 1$
14:   end while
15: end if
```

Generative Adversarial Networks. GAN is another method of generating training data with demanded structures and properties for scientific applications. We can use GAN to explore the space of possible data, tuning the generated data to have specific target properties. In particular, GAN (CycleGAN) is used to augment CT images and improve generalizability in CT segmentation tasks[23]. GAN is also adopted to augment 3D MRI data for medical image segmentation[24]. Besides augmenting biomedical datasets, GAN can be also used to generate multi-sensor data for the aerial object detection and semantic segmentation on visual data, such as 3D Lidar reconstruction using the ISPRS and DOTA datasets[25]. As shown in Fig.2, GAN has a pair of components competing with each other, where the generator model is responsible for

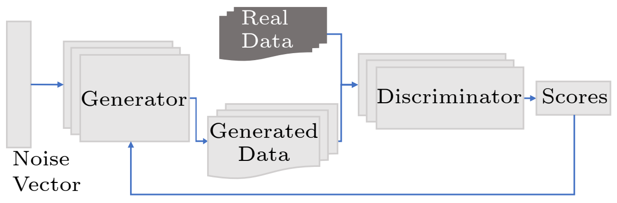


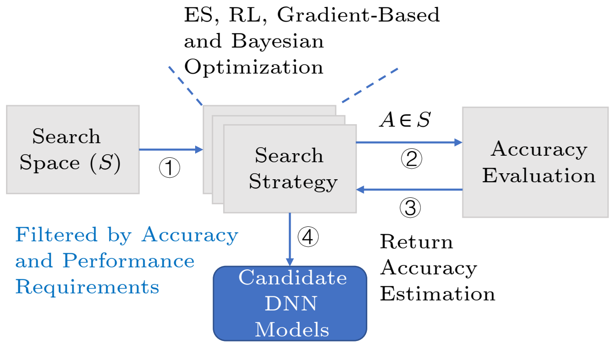
Fig.2. GAN for data augment.

generating new synthetic data (e.g., the DNA sequence in the Genome). The discriminator model calculates the similarity score between the generated data and the real data. Concretely, the GAN models are different case by case. As iteratively training the models, the accuracies of the two models are improved. With the trained generator by GAN, we can concurrently run it to generate a large amount of training data.

3.2 Model Augment

SAIH can scale its computation demand by model augment, which in deep learning is mainly either by AutoML[26] based techniques such as NAS, or manually adjusted by AI experts. With these two kinds of model augment methods, generated candidate models can satisfy both scalability and accuracy requirements. Although manually tuning is easier, it is time-consuming and requires a lot of expert knowledge. In SAIH, we adopt NAS as the fundamental of our model augment method and a new component is inserted into NAS for selecting out the qualified models that have customized computation demand.

As shown in Fig.3, NAS generally consists of three main components: the search space, search strategy, and accuracy evaluation. The search space contains the architectures that can be selected. Users can set hyperparameters to restrict the searching properties of architectures for reducing the size of the search space and simplify the search. The search strategy specifies how to traverse in the search space, and often has exponential complexity, which motivates experts to design novel search optimizations or heuristic algorithms to find the optimal or near-optimal DNN models. The widely investigated optimizations include evolution strategy (ES), reinforce learning (RL), Bayesian optimization, etc. The accuracy evaluation is estimating the performance of model architecture, e.g., by performing a standard training and validation of the architecture on data. When the estimation is evaluated and returned back, the search strategy will decide the next searching architecture, or accept the model based on the accuracy.


ES, RL, Gradient-Based and Bayesian Optimization
Search Space (S)
①
Search Strategy
$A \in S$
②
③
Accuracy Evaluation
Return Accuracy Estimation
④
Filtered by Accuracy and Performance Requirements
Candidate DNN Models
Fig.3. Network architecture search in SAIH.

In order to generate models with a demanded computation cost, e.g., FLOPs, we design a novel component, model filter, to filter models with a demanded range of computation costs. The model filter is integrated as a supplement component of the search strategy, enabling that SAIH explores models with a specific range of computation costs. Due to the long time of NAS, e.g., searching 20 000 neural networks across 500 P100 GPUs over four days[27], in SAIH we tend to place the model filter before the searching strategy, preparing candidate DNN models with scaled computation costs before training and evaluating. The model filter estimates the theoretical computing cost of the candidate model as the filtering basis. We aggregate the number of multiply-accumulate operations to represent the computation cost. It is straightforward to count the number of multiply-accumulate operations in the forward pass, while it is not for the backward pass, which is derived by the forward pass. Here we adopt (1)[5] for estimating the computation cost of both the forward and backward pass,

$$Computation = numAddMul \times flopAddMul \times FB, \tag{1}$$

where $numAddMul$ denotes the total number of add-multiply operations in the forward pass of a model, $flopAddMul$ is the number of FLOPs per add-multiply, and FB is a constant of 3 for calculating both forward and backward pass.

3.3 Performance and Qualitative Evaluation Summary

AI applications have been emerging from various scientific domains. These applications have two main

[5]https://openai.com/research/ai-and-compute, Mar. 2024.

components: data and model. The data format and the model architecture vary largely in different AI applications. When we apply SAIH to a domain case, the evaluation can present not only the AI performance on HPC systems, but also the performance bottlenecks introduced by the characteristics of AI applications and frameworks, in terms of data movement, scalability, etc. Thus, as shown in Table 1, we need to set a standard or metrics to summarize the evaluation results of SAIH instances. The metrics table typically includes three main sections, 1) the characteristics of the SAIH study case, 2) AI performance in terms of FLOPS range, speedup, etc., and 3) other profiling results.

4 Case Study

In this section, we orchestrate a cosmological AI application, CosmoFlow[2], as a case study of SAIH, named as SAIH-cosmo. Then, we apply SAIH-cosmo to an HPC system for understanding the relative AI performance trend. We observe some findings stemming from the convergence of the training model, performance bottlenecks on the software stack and hardware configurations and tuning. Such insights can guide the HPC systems for tuning better AI performance in the future.

4.1 Background

4.1.1 CosmoFlow

In the domain of cosmology, it is of great significance to determine cosmological states, which can describe the evolution of the whole universe and the distribution features of matter and energy in a particular space. Enabled by AI technology, the CosmoFlow project aims at providing a faster, cheaper, and more accurate data processing workflow for cosmology research. It employs 3D CNN to learn the mapping between the distribution of matters within a defined space and three cosmological states, by training the model on a large amount of simulated data. With the trained model, cosmological states of a space can be inferred faster and more accurately, compared with traditional statistical methods.

4.1.2 HPC System Configuration

The system is a heterogeneous HPC cluster. Each node of the cluster is equipped with two Intel® Xeon® Gold 6132 CPUs (14 cores) operating at 256 GB memory and four Tesla V100 SXM2 GPUs. Each Tesla V100 GPU has 16 GB HBM2 and can provide up to 7.8 TFLOPS double-precision performance, 15.7 TFLOPS single-precision performance, and 125 TFLOPS half-precision performance. Tesla V100 GPUs are interconnected by NVLink within the node and the nodes of the cluster are interconnected by InfiniBand. The file system is Lustre for the persistence of the training data.

4.2 Data Augment and Preprocessing

We use the simulation method for SAIH-cosmo, which adopts a widely-accepted simulator of generating cosmological data, e.g., adopted by CosmoFlow[2]. Different from the static datasets in CosmoFlow of MLPerf HPC, we produce the dataset of SAIH-cosmo by modifying its simulation and make it scalable on the dataset size, avoiding non-trivial data collecting,

Table 1. Performance and Qualitative Evaluation Metrics for SAIH Case Study

Metric	Value
Domain	HPC scientific domains, e.g., particle physics, astronomy, biomedical, climate, bioinformatics
Data augment	Simulation, GAN, and data transformation
Model augment	Manual tuning, hyperparameter search, and NAS
DNN model	2D CNN, 3D CNN, RNN, Transformer, etc.
Data format	1D genome, 2D image, 3D particle, 3D tomography, etc.
AI framework	Tensorflow, PyTorch, Caffe, etc.
Dataset size	Dataset size range in weak scaling
HPC system	The hardware configurations of an HPC system on which SAIH is performed
FLOPs of AI model	Computation demand range for the candidate AI models
FLOPS (ratio of theoretical FLOPS)	Aggregate FLOPS achieved by performing SAIH instance on an HPC system
Singe GPU performance	Single GPU performance range on data/node scaling
Speedup on node scaling	Speedup range on strong scaling evaluation
Arithmetic intensity	Range of arithmetic intensity of AI models
Accuracy/loss	Range of accuracy/loss of AI models with data/node scaling

downloading, and preprocessing. The data is produced by N-body simulation, pyCOLA[28], of which the initial parameters are randomly generated by MUSIC[29].

In our case study, MUSIC randomly initializes matter distribution based on three parameters, consisting of Ω_M, σ_8 and N_s, before the simulation runs. There parameters can describe the cosmological state. The three parameters are also kept as the label for its corresponding simulation output. In detail, Ω_M is the proportion of matter in the universe. We assume a flat geometry for universe, i.e., the sum of the contributions of matter and dark energy to the energy density of the universe $\Omega_M + \Omega_\lambda = 1$. Ω_λ is the proportion of the dark energy. σ_8 is the amplitude of mass fluctuations in the universe at a distance scale of 8 Mpc^3/h. N_s is the scalar spectral index for the spatial curvature of a comoving slice of the space time. The simulation, pyCOLA, implements the comoving lagrangian acceleration (COLA) method in the temporal and spatial domains for the N-body simulations. It simulates the evolution of physical particles from the MUSIC initial conditions. The simulation result describes the position information of a predefined 3-dimensional space, such as 256^3, 512^3, and $1\,024^3$, which can be adjusted in the initial condition of MUSIC. Essentially, the three cosmological states are set in the specific ranges, in which MUSIC can randomly pick values based on the uniform distribution.

We set the simulation cube 512^3 relevant to 512 Mpc^3/h cubic space as the leftmost cube in Fig.4 and the three cosmological states are randomly generated from $0.25 < \Omega_M < 0.35$, $0.78 < \sigma_8 < 0.95$, and $0.9 < N_s < 1.0$. MUSIC and pyCOLA run on CPUs and consume massive resources, in particular memory. Based on our evaluation, the memory requirement for an instance of the simulation with the space size of 512^3 requires 17 GB memory for MUSIC and 20 GB for pyCOLA. For a single-node execution, the concurrency of MUSIC and pyCOLA is constrained by memory capacity. Since MUSIC and pyCOLA run in series rather than in parallel, we can at most concurrently perform 12 simulations on each node. Further, as in Algorithm 1, we employ multi-node execution to further accelerate data augment. As each simulation is independent, three cosmological states are randomly sampled within the predefined ranges via a third party math library, GNU Scientific Library[⑥].

After raw data is collected from pyCOLA simulation, the data preprocessing is then performed to transform the raw data to the training data that can be directly fed into the 3D CNN model. The data is transformed to volumetric form where a 3D histogram of d^3 voxels represents the normalized density of the dark matter for each cube. As shown in Fig.4, the resolution of voxel is $2 \times 2 \times 2$ Mpc^3/h, and the 512 Mpc^3/h cube is transformed to 256^3 voxel volumes. Then, each volume is split into eight sub-volumes labeled with the same cosmological states as in the rightmost cube of Fig.4. Each sub-volume is 16 MB and stored in the HDF5 data format. For this case study, we perform 12 632 simulations and create 101 088 sub-volumes as the total training dataset, taking around 1.6 TB storage. In SAIH-cosmo, we extract a portion of the dataset for weak and strong scaling evaluations.

4.3 3D CNN Model Augment

Here we take advantage of both the manual design method and automated method. For the manual design method, we adopt the 3D CNN model proposed by Mathuriya *et al.*[2] as the smallest model with the least computation cost. For the automated method, we customize NAS to prepare another two

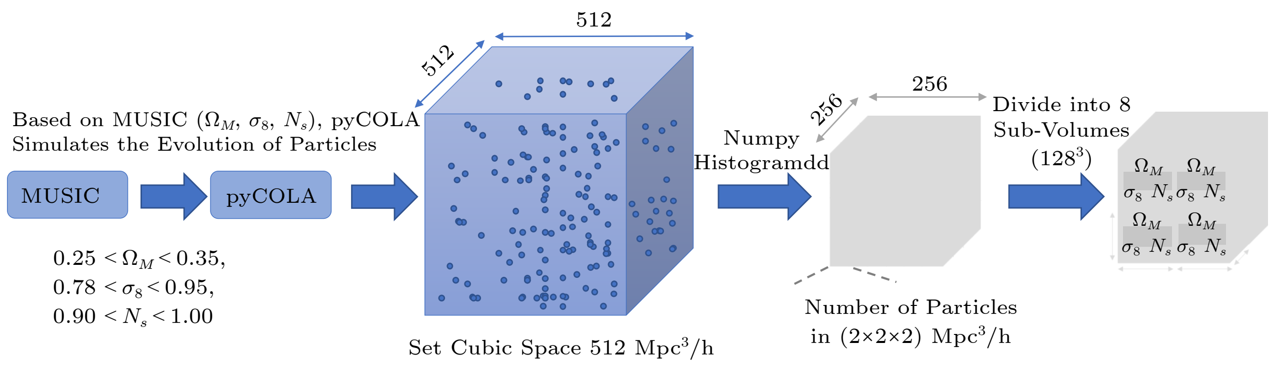

Based on MUSIC (Ω_M, σ_8, N_s), pyCOLA Simulates the Evolution of Particles
MUSIC
pyCOLA
$0.25 < \Omega_M < 0.35$,
$0.78 < \sigma_8 < 0.95$,
$0.90 < N_s < 1.00$
512
512
Set Cubic Space 512 Mpc^3/h
Numpy Histogramdd
256
256
Number of Particles in (2×2×2) Mpc^3/h
Divide into 8 Sub-Volumes (128^3)
Ω_M σ_8 N_s

Fig.4. Data augment and preprocessing of the cosmological data.

⑥https://www.gnu.org/software/gsl/, Mar. 2024.

CNN models for SAIH-cosmo with incremental computation costs and model accuracies. In details, we generate them by our proposed filter-based model augment method, which in this case adopts DARTS[30] as its internal NAS strategy. DARTS is a cell-based NAS technique with the gradient descent search strategy. The gradient descent search strategy is the non-differentiable approach and it is orders of magnitude faster than other non-differentiable approaches. With the V100 GPU, DARTS only takes about two GPU days for exploring the medium model in our experiments, while other techniques require thousands of GPU days theoretically. DARTS is a cell-based NAS technique, which searches the architecture of a cell and connects multiple identical cells as the entire model. As shown in Fig.5(a), DARTS initially assigns a number of feature maps (nodes) in a cell, while edges between nodes are unknown. Then, users define candidate operations for each edge based on observations from previous models. Since each edge normally has only one operation, DARTS introduces the architecture possibility for each operation in each edge and determines the operation in each edge by gradient descent. Training DARTS on a small dataset is adequate, which further accelerates the searching process. As shown in Fig.5(c), in each iteration, DARTS jointly learns the potential architectures and the network weights, which have the same target that improves the prediction accuracy. In this way, the model constantly converges to the architecture with a higher accuracy. Finally, DARTS selects the most likely operation of each edge to determine the final architecture.

Our 3D convolutional cell consists of $N=7$ nodes, and operation candidates include 3×3 and 5×5 3D separable convolutions, 3×3 and 5×5 3D dilated separable convolutions, 3×3 3D max pooling, 3×3 average pooling, identity, and zero. For each convolution operation, a batch-norm layer and the Leakey Relu activation function are followed behind based on observations from the smallest model. Since a single sample in our dataset is larger than experiments in DARTS, we set our model with 16 cells. Reduction cells (stride=2) are the 1st, 5th, and 13th cells, and three FC layers are composed as in the small model. For the other hyperparameters, we adopt the default setting in the cifar-10 example of DARTS[30].

The performance filter is placed before the original searching strategy in DARTS. The channel number of convolution layers is a hyperparameter in DARTS and it largely influences the computation cost. With a predefined computation cost, the performance filter can estimate the channel number of convolution layers for the formal searching process. The performance filter evaluates the forward pass of models by using PyTorch-OpCounter[⑦] and calculating the overall training cost by the aforementioned formula in (1). We set the candidate models with 4 TFLOPs (medium) or 16 TFLOPs (large) in the performance filter. Eventually, the medium model has 101.6×10^6 parameters and the training process of a single sample requires about 4.15 TFLOPs, which largely exceeds that of the small model. As for the large model, it has 374.2×10^6 parameters and takes

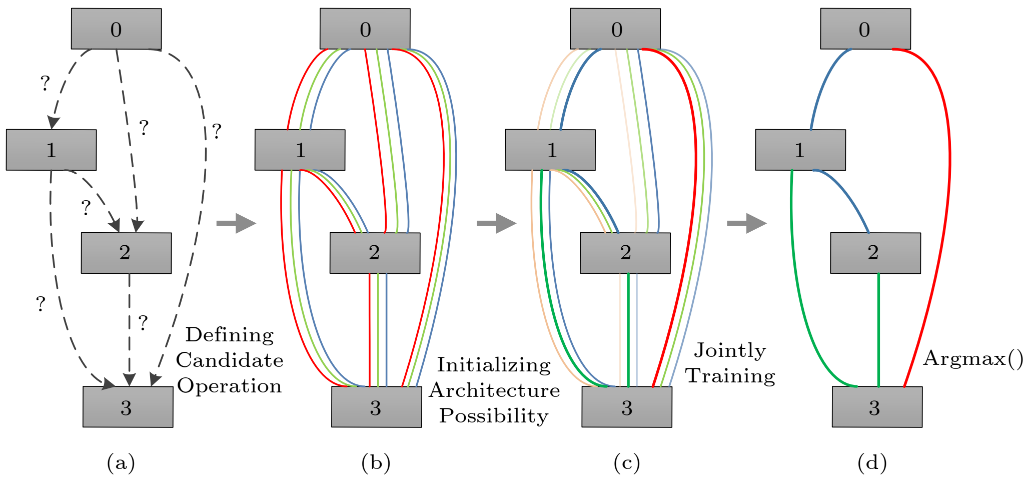

0
1
2
3
?
Defining Candidate Operation
Initializing Architecture Possibility
Jointly Training
Argmax()
(a)
(b)
(c)
(d)

Fig.5. DARTS in SAIH-cosmo. (a) Assigning feature maps. (b) Defining candidate operations for each edge. (c) Learning the potential architectures and the network weights jointly. (d) Selecting the final architecture.

[⑦]https://github.com/Lyken17/pytorch-OpCounter, Mar. 2024.

16.2 TFLOPs for training with a single sample. Notably, we can store model architectures searched for using in other HPC systems.

4.4 Parallelized Training

Training a CNN model generally requires iterative traversals over the dataset to improve the accuracy of the model and make it finally converge. We adopt the commonly-used data parallelism for the training process of SAIH-cosmo on the target HPC system. Data parallelism allocates a training task on a minibatch of samples to an available computational devices (GPU or CPU core). Each device keeps a copy of the target model and performs the training process independently on its own samples. After finishing the forward and backward propagations, the weights are updated by performing the allreduce operation on all devices. We adopt NVIDIA NCCL as the collective data movement backend, which transmits data through NVLink and Infiniband.

Batch size, which determines the number of samples parallelly trained in a GPU, is critical for data parallelism. The larger the batch size, the less the number of allreduce operations required for each epoch. This results in less communication overheads. However, the batch size is limited by two factors, i.e., memory capacity and model convergence. In terms of memory capacity, increasing the batch size leads to a proportional growth of memory capacity. In order to maximize the performance of an HPC system and investigate the bottleneck when it is fully loaded on the GPU device, we tune the batch size for each model. Thus, the batch sizes of the small model, medium model and large model in each GPU are set to 10, 4 and 1, respectively. As for the convergence problem, it is mainly solved by optimizing machine learning algorithms, such as gradient-based optimization, e.g., momentum, Adagrad and Adam[31]. Here we adopt Adam as our gradient optimizer. Adam designs the adaptive learning rate for accelerating stochastic optimization. We set hyperparameters of Adam with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\hat{\epsilon} = 10^{-8}$. Then we further combine Adam with LARS[32]. LARS adaptively adjusts the learning rate for each layer and it enables better convergence in an extremely large batch size.

5 Evaluation

5.1 Experimental Setup

SAIH-cosmo is implemented in PyTorch v1.5.0 and the *DistributedDataParallel* class of PyTorch is used for data parallelism. We perform both node scaling and data scaling executions on SAIH-cosmo with different models to understand 3D CNN training on the HPC system. Meanwhile, we further investigate their arithmetic intensity by the NVIDIA profiling tool (*nvprof*) and mixed precision training by *Nvidia apex*. Overall, as shown in Table 2, for each model, the node scaling training uses 1/32 of the complete dataset (101 088 samples) and runs on 1, 2, 4, 8, 16, and 32 nodes of the cluster ($n = 1, 2, 4, 8, 16, 32$). The data scaling training uses four nodes and the training dataset size scales from 1/64 to 1/1 of the complete dataset ($d = 64, 32, 16, 8, 4, 2, 1$). To illustrate the model convergence, we execute each training with 60 epochs. Notably, it is optional for SAIH users to ignore the convergence and only focus on performance trends, which can largely reduce the evaluation costs, from days to a few hours.

Table 2. Experimental Settings

	Node Scaling		Data Scaling	
	Data	Node	Data	Node
Small model	1/32	n	$1/d$	4
Medium model	1/32	n	$1/d$	4
Large model	1/32	n	$1/d$	4

5.2 Model Accuracy

Training an accurate DNN model is the fundamental requirement for an AI application. The losses of the three models show a rapid decrease within the first epoch and then show different trends. We demonstrate the losses after each epoch of the three models in Fig.6, which shows that a larger model can generally provide better accuracy.

Besides, we additionally find that a larger model in SAIH-cosmo has more stable convergence in multinode training. For the small model, the losses of three cosmological states decrease with significant fluctuations, and it fails to converge within 60 epochs. In contrast, the medium and large models converge much more stable. At the beginning of the training phase, the losses of the medium model initially go up and then decrease rapidly, while those of the large model converges constantly and steadily. Also, in the final epochs, the large model achieves with the best losses.

Although we apply the LARS algorithm[32], increasing the number of GPUs to 16 (4 nodes) influ-

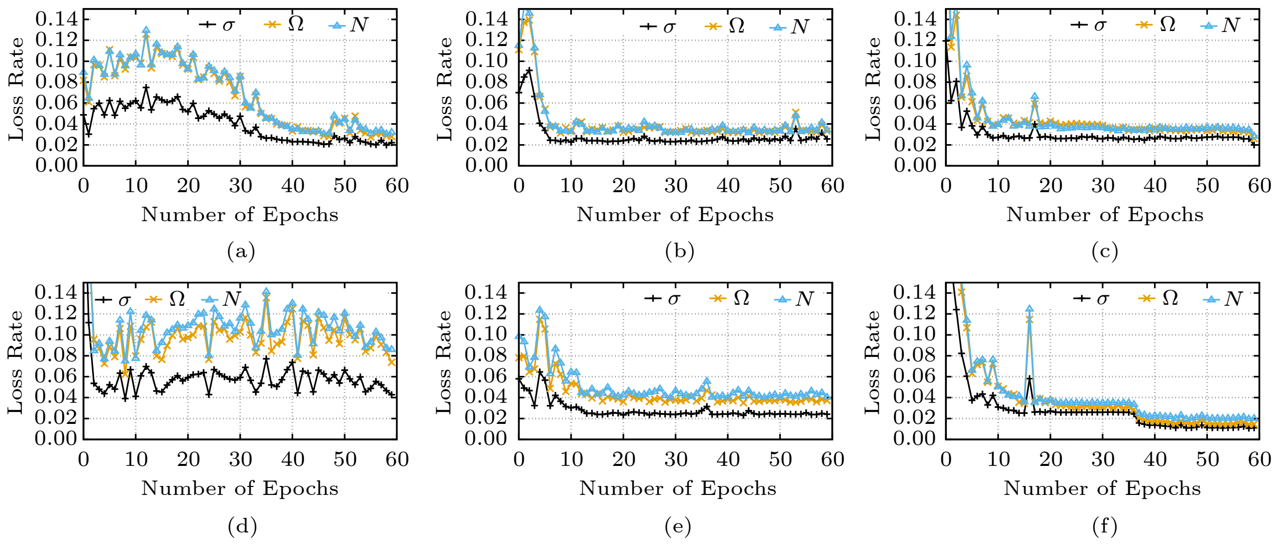


Fig.6. Loss change in 1/32 dataset. (a) Small model in 1 node. (b) Medium model in 1 node. (c) Large model in 1 node. (d) Small model in 4 nodes. (e) Medium model in 4 nodes. (f) Large model in 4 nodes.

ences the convergence of the AI models. As shown in Fig.6(d), the small model displays wider fluctuations in losses, severely damaging the convergence, which is also consistent with the results of CosmoFlow[2]. In terms of the medium model, it also shows more rough converging process compared with the 1-node execution, and slightly impairs the final accuracy. On the contrary, distributed training contributes to positive impact on the large model. Increasing the number of nodes makes the large model converges to a better result.

While we adopt the simplified NAS in SAIH-cosmo with a narrow searching space, our automatically generated models achieve significant improvement as the computation demand increases, which validates its applicability as well as the preservation of scientific meaning.

Finding 1. Overall, the models augmented by NAS lead to better accuracy and better convergence. Besides, with progressively increasing the sizes of models, we find that the convergences of larger models are more steady in multi-node training.

5.3 Evaluation with GPU Node Scaling

The evaluation with GPU node scaling follows the definition of strong scaling, which concerns the acceleration for a fixed problem size with respect to the number of GPU devices, and is limited by the fraction of the serial part in a program that is not amenable to parallelization. When fixing the data size (1/32 dataset) and scaling up the number of GPUs from 4 to 128, it displays different performance scalability for these models as shown in Figs.7(a)–7(c). The average FLOPS of a single GPU are shown in Fig.8(a).

We first discuss on the average performance of each GPU, which varies largely in different model sizes. With four GPUs, the medium model shows the best performance at 9.21 TFLOPS for each GPU, achieving 58.66% peak performance of Tesla V100. In comparison, the large model executes with lower average FLOPS. The small model has only achieved about 800 GFLOPS on each GPU, which is much lower than the other two models.

As shown in Table 3, the arithmetic intensity of the small model is much lower than those of the other two models. Here "Read Times" and "Write Times" represent the number of memory read accesses and memory write accesses respectively. Also, we can notice that the computation demand of the small model is several orders of magnitude lower than those of the other two models. To profile the memory usage of each model, we find that the gap of the memory footprint between the small model and other two models is not so large as the computation demand. In details, the weights and intermediate results of the medium and large models are only 3.79x and 7.46x more than those of the small model, respectively. In comparison, the computation demands are 67.25x and 257.97x higher than those of the small model, respectively. When we adjust the batch sizes of the three models to saturate the GPU memory, the small model cannot train with an extremely large batch size to make full use of all computing units, resulting in a much lower arithmetic intensity than the other mod-

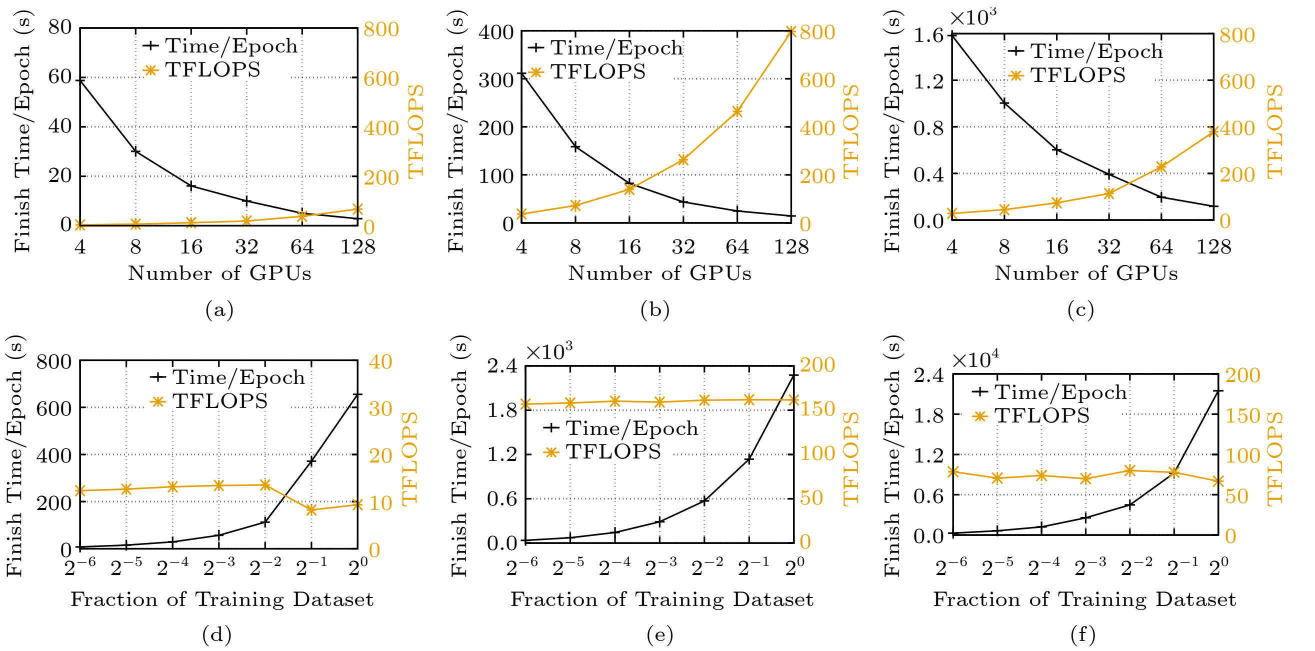


Fig.7. Performance evaluation with node and data scaling. (a) Small model in node scaling. (b) Medium model in node scaling. (c) Large model in node scaling. (d) Small model in data scaling. (e) Medium model in data scaling. (f) Large model in data scaling.

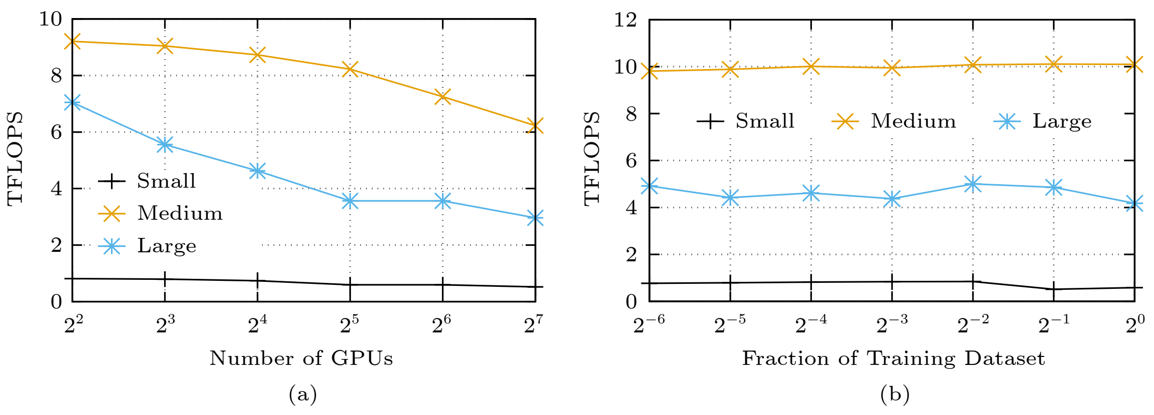


Fig.8. Average TFLOPS per GPU. (a) GPU node scaling. (b) Data scaling.

Table 3. Arithmetic Intensity Analysis with 4 GPUs and 1/32 Dataset

Model Size	Batch Size	FLOPs	Read Times	Write Times	Intensity
Small	10	6.90×10^{10}	1.00×10^{7}	7.53×10^{7}	808
Medium	4	4.64×10^{12}	1.15×10^{9}	7.09×10^{8}	2 500
Large	1	1.78×10^{13}	7.13×10^{9}	5.21×10^{9}	1 442

els with more computation demands. In this way, the small model is limited by GPU memory capacity and the small batch size cannot saturate the computing unit. Besides, we also observe that the memory footprint mainly comes from intermediate results rather than weights in 3D CNN, where the memory footprint of intermediate result is about 30x larger than those of the weights in these three models.

As for the large model, although it has the largest GPU memory footprint and computation demand, the counter-intuitive observation is that it has lower arithmetic intensity than the medium model. Table 4 and Table 5 list the arithmetic intensity of top-10 kernels sorted by the number of memory access. %MemAcc is the percentage distribution of memory access of kernels. "ID" is a unique ID for each kernel in the CNN model, which is executed on GPU. First, the types and percentages of kernels in the large model are quite different from those in the medium model. In the large model, the kernel with 66.53% memory accesses holds only 516 arithmetic intensity, while the top-3 kernels in the medium model have 1.8x higher weighted aggregate arithmetic intensity than those in the large model, i.e., 2 183 compared with 1 210. This

Table 4. Profiling Results of CNN Kernels

Medium Model			Large Model		
ID	%MemAcc	Intensity	ID	%MemAcc	Intensity
0	26.76	**922**	0	66.52	**516**
1	26.31	**5 201**	1	13.57	**6 391**
2	15.06	**3 777**	5	2.18	**5**
3	5.68	19	10	1.57	19
4	3.41	1	11	1.46	4
5	3.40	36	3	1.46	4
6	2.90	2	7	1.46	4
7	2.90	8	6	1.10	8
8	1.75	0	12	1.02	1
9	1.15	12	13	0.94	36

Note: These dominant values are marked in bold.

Table 5. Kernels in Cosmological CNN Model

ID	Kernel
0	$scudnn\_128x64\_stridedB\_splitK\_small\_nn\_v1$
1	$scudnn\_128x64\_stridedB\_splitK\_medium\_nn\_v1$
2	$scudnn\_128x128\_stridedB\_splitK\_small\_nn\_v1$
3	$implicit\_convolveNd\_sgemm(int1024)$
4	$convolveNd\_wgrad\_engine$
5	$implicit\_convolveNd\_sgemm(int512)$
6	$bn\_bw\_1C11\_kernel_new$
7	$bn\_fw\_tr\_1C11\_kernel\_NCHW$
8	$setOutputKernel$
9	$vectorized\_elementwise\_kernelGLOBAL\_\_N\_\_57\_tmpxft$
10	$convolveNd\_wgrad\_engine(int = 8)$
11	$convolveNd\_wgrad\_engine(int = 7)$
12	$vectorized\_elementwise\_kerneladd\_kernel\_cuda$
13	$vectorized\_elementwise\_kernelgpu\_kernel\_with\_scalars$

is due to the higher internal thread-level parallelism in the medium model than that in the large model. In addition, the fact that the 3D feature map of the large model is bigger than that of the medium model results in more strided memory access, further decreasing the arithmetic intensity.

Finding 2. For 3D CNN models, the ratio of memory footprint and computation amount is not stable but changes in a wide range. Besides, the models with more layers and weights are not always leading to higher GPU performance and TFLOPS, which are instead significantly determined by the arithmetic intensity of AI models.

Then we analyze the performance scalability of the distributed training. When scaling the training to 128 Tesla V100 GPUs, the small model achieves 112.3 aggregate TFLOPS; the medium model achieves up to 797.1 TFLOPS; and the large model achieves 379.2 TFLOPS. From 4 GPUs to 128 GPUs, the speedups of these three models are 25.6x, 22.2x, and 13.45x respectively. Since the training procedure contains computation and communication in each iteration, the larger model has weaker scalability due to both the larger communication overhead of each iteration and the smaller batch size limited by the GPU memory capacity. On the one hand, communications among GPUs adopt allreduce to collectively aggregate gradients with $O(parameters + nodes)$ time complexity. The hyperparameter, *batch size*, does not change the total number of gradients in the model to be communicated, since the gradients in a minibatch of samples will be aggregated locally within a GPU before performing inter-node communications. On the other hand, the larger batch size can reduce the number of communication operations in one epoch, leading to relatively less communication overheads. The 16 GB HBM of a single Tesla V100 GPU, which is larger than most of other GPUs' memory capacity, can only contain one input sample trained on the large model, which occupies more than 13 GB memory. Thus, the larger model has a smaller batch size, resulting in weaker parallelism. A large batch size plays a key role in increasing parallel efficiency, which is also stressed in You *et al.*[32].

Finding 3. Besides saturating computation units by high parallelism, a large batch size can further improve the performance scalability by reducing I/O frequency and overhead.

5.4 Evaluation with Data Scaling

In this subsection, we further investigate the data scaling, which fixes the number of nodes to 4 and scale the training dataset from the fraction 1/64 to 1/1 of the complete dataset and the performance is illustrated in Figs.7(d)–7(f). The average TFLOPS per GPU is in Fig.8(b).

Overall, in data scaling, there are two major factors that affect the aggregate performance, data loading in memory hierarchy and communication for allreduce. For communication, since the number of communication-intensive operation, allreduce, changes proportionally to the computation, it almost has no impact on the ration of communication time and the overall evaluation time. Thus, we here mainly focus on data loading. In terms of the medium model and the large model, their performances fluctuate in a limited range, and that of the large model is wider. As for the small model, when the data volume is 1/64, 1/32, 1/16, 1/8, and 1/4, the average FLOPS is very stable at around 800 GFLOPS, while there is a decrease to about 520 GFLOPS with the 1/2 and 1/1 datasets, and this observation is more obvious in Fig.7(d). Investigating the breakdown, for all of the three models, we observe that there is a long data loading time at the beginning of each epoch, and unexpected data loading time when training with the 1/1 or 1/2 datasets. As the small model has the shorter computation time and larger fraction of data loading time, we can observe that the overall performance decrease is more obvious.

The data loader in PyTorch endeavors to load all training data into GPU node memory. At the beginning of each epoch, the execution of data loading results in a large amount of idle time for computing units, as well as heavily stressing the shared file system. Zooming into the unexpected data loading time, the data loader in PyTorch has a prefetch strategy from node memory to GPU memory, while it lacks an efficient data prefetch strategy from the file system to node memory. The entire dataset takes 1.6 TB storage, while each node of the cluster has only 256 GB memory capacity. In this way, four nodes fail to load the entire 1/1 and 1/2 dataset into memory. It has to wait for data in the swap area to be loaded before continuing the training procedure, leading to unexpected data swapping overhead. For example, the percentage of data loading time in the small model increases from 2.83% to about 20% as the dataset scales from 1/4 to 1/1, resulting in 31.2% reduction in overall FLOPS.

Finding 4. Different from traditional HPC applications, AI training demands I/O throughout during the whole training procedure for accessing input training data and intermediate data. Thus, a local storage system on each compute node is recommended for staging the training and intermediate data.

Finding 5. A large dataset with frequent data accesses can make memory hierarchy become a bottleneck. Thus, with scaling scientific datasets, SAIH-cosmo probes the performance capability of memory hierarchy, which is critical to the evaluation of the overall AI performance on HPC systems.

5.5 Mixed Precision Results

Theoretically, Tesla V100 claims 15.7 TFLOPS of single-precision (FP32) performance and 125 TFLOPS half-precision (FP16) performance by utilizing its novel tensor core architecture. We evaluate the performance of SAIH-cosmo under the mixed precision setting by employing NVIDIA apex with optimization level O1. However, as shown in Table 6, the mixed precision evaluations, i.e., Apex[O1], can only achieve up to 1.76x speedup for the three models compared with the FP32 execution, i.e., Apex[O0]. This is significantly lower than the theoretical peak performance.

Table 6. Execution Time in 1/32 Dataset with Different Precision Levels

Precision Level	Description	Execution Time (s)		
		Small	Medium	Large
Apex[O0]	Pure FP32	18.71	364.06	2 497.2
Apex[O1]	Mixed precision	19.46	205.38	1 867.9

We adopt *nvprof* with turning on the *tensor_precision_fu_utilization* metrics to diagnose the counter-intuitive performance gap. While according to the NVIDIA official document, only cudnn[⑧] newer than version 8.0.2 can be compatible with PyTorch and enable it to utilize tensor core architecture for training 3D convolution with mixed precision. Our results show that the tensor core is under a very low utilization only contributing around 4.4% of the entire execution, which results in an extremely limited performance gain on SAIH-cosmo. Also, through profiling on individual kernels, we find that cudnn is inefficient in some outlier mixed-precision 3D convolutions, e.g., the situation with a small number of channels requires high memory access overhead, and it is easy to improve. To further take advantage of the tensor core architecture, the instruction-level restrictions under high-level AI frameworks and convolution libraries, e.g., the input size and memory layout[33], have to be resolved, in particular for performing 3D convolutions, which are not optimized so well as the 2D convolutions commonly used in hotspot AI research areas.

Finding 6. AI related libraries, like cudnn and cublas, are released with the capability of supporting low-precision and mixed-precision training operations on novel architectures, e.g., tensor core. However, without customized by AI system experts, the mixed precision is not well prepared in a plug-and-play manner for supporting accelerating common AI operations on AI frameworks, e.g., conv3d in PyTorch.

5.6 Comparison and Summary

This subsection is going to compare SAIH with other methods and illustrate our contributions. Table 7 presents features of representative benchmarks. No-

[⑧]https://developer.nvidia.com/cudnn, Mar. 2024.

Table 7. SAIH and Representative Benchmarks

Name	Positioning	Metric	Application Domain	Target
HPL-AI[8]	Benchmark	FLOPS, FLOPS/Watt	Linear equations	Measure mixed-precision performance of HPC systems
Deep500[17]	Benchmark	Throughput, time to solution	Any AI applications in theory	Help evaluate framework implementations
MLPerf-HPC[20]	Benchmark	Time to solution	Cosmology and weather analysis	Speed up the training time
HPC-AI500[21]	Benchmark	Valid FLOPS	ImageNet and weather analysis	Speed up the training time
SAIH	Evaluation method (mainly vertical comparison)	FLOPS as scaling, scalability	Cosmology	Understand AI performance trend and guide the design for emerging HPC systems

tably, SAIH is an evaluation methodology rather than an AI benchmark with a defined AI model and dataset, e.g., HPL-AI and HPC-AI500. It is different from existing benchmarking methods on evaluation target (AI performance trend via both strong scaling and weak scaling evaluations), evaluation settings (varying model sizes and dataset sizes) and evaluation testcases (cosmological AI applications). Specifically, in our testcase, we adopt unique evaluation settings such as both strong scaling and weak scaling evaluations with varying model sizes and dataset sizes. Thus, it could not directly compare our method with existing benchmarks, and we make a qualitative comparison between SAIH and four representative benchmarks, i.e., HPL-AI[16], Deep500[17], MLPerf-HPC[20] and HPC-AI500[21], to illustrate that our SAIH is effective and can bring new insights for next-generation HPC systems.

At first, SAIH has a different positioning from the benchmark projects. The benchmark projects mainly focus on the AI performance of the same job in different infrastructures and rank different infrastructures, while SAIH aims at demonstrating the performance changes when AI applications scale up and understanding the requirements of AI applications for subsystems, e.g., I/O, compute unit, and communicate. In detail, HPL-AI can represent the traditional HPC benchmarks. It measures and compares the mixed-precision performance of HPC systems by solving mixed-precision linear equations. Notably, HPL-AI provides the scalability as our SAIH and can derive the maximum performance of HPC systems. However, it mainly investigates the computation of an HPC system and can hardly represent the practical AI performance of an HPC system. In comparison, the data and model augment methods in our SAIH keep the scientific meaning when scaling AI applications, making our evaluation with practical significance.

Deep500 which is a modular benchmark infrastructure can be integrated into most evaluation methods. It characterizes many fine-grained metrics, e.g., utilization of computing devices, collective communication, and IO, into modules and can be used to help evaluate different framework implementations and multiple levels of operators. In fact, the implementation of Deep500 is difficult to be integrated into large-scale AI applications and our SAIH includes many metrics defined in Deep500.

MLPerf-HPC is an emerging benchmark focusing on the intersection of scientific AI applications and HPC systems. MLPerf-HPC rewrites the CosmoFlow (as our SAIH) and a weather analysis AI application. Compared with SAIH, it mainly adopts the static dataset and model architecture and takes the traditional time-to-solution metric as the main criterion for ranking HPC systems, lacking scalability. Compared with SAIH, it lacks the weak scaling evaluation, which is important in the HPC filed. Thus, it cannot investigate the performance changes dynamically and help understand the AI performance trends as our SAIH.

At last, for HPC-AI 500, in order to further assure equivalence, it presents a new metric named valid FLOPS, which imposes a penalty on failing to achieve a target training quality. It selects the image classification and extreme weather analysis as tasks. Besides, its core idea is similar to MLPerf-HPC. The findings of our SAIH are different from those of MLPerf-HPC and HPC-AI500, and can illustrate the AI capability and performance trend of HPC systems in both strong and weak scaling evaluation.

Through the evaluation and comparison, we notice that the main idea of SAIH does not overlap with those of existing methods. SAIH provides a feasible solution to make AI application evaluation scalable and can investigate many novel findings. Particularly, with the SAIH-cosmo, we evaluate the AI performance for an important type of models, 3D CNN, on a heterogeneous HPC cluster. The evaluation with scaling problem size helps us diagnose some emerging system performance bottlenecks and investigate the potential AI performance of the HPC system. Moreover, we illustrate the performance details of SAIH-cosmo and quantify its evaluation on the HPC system in Table 8.

Table 8. Performance and Qualitative Evaluation Summary of SAIH-cosmo

Metric	Value
Domain	Cosmology
Data augment	Simulation
Model augment	NAS
DNN model	3D CNN
Data format	3D particle
AI framework	PyTorch
Dataset size (TB)	[0.025, 1.600]
HPC system	GPU cluster with 32 nodes, 4 Tesla V100 GPUs/node
FLOPs of AI model	[69G, 16.2T]
FLOPS (ratio of theoretical FLOPS)	[69.6 TFLOPS (5.2%), 797.1 TFLOPS (59.6%)]
Singe GPU performance (TFLOPS)	[0.51, 9.21]
Speedup on strong scaling	[13.45x, 25.60x]
Arithmetic intensity	[838, 2 501]
Accuracy/loss	[0.024 75, 0.067 54]

6 Conclusions

In this paper, a novel scalable methodology, SAIH, was proposed for better understanding AI performance trend of HPC systems. Based on the methodology, we implemented a testcase SAIH-cosmo. SAIH-cosmo is transformed from a cosmological AI application and it is with data and computation scalability. Through evaluating SAIH-cosmo on a real HPC system, we successfully diagnosed many new insight findings as data and computation scaling large. For example, we found that the convergences of larger models are more steady in multi-node training, which cannot be noticed if there is not a range of incremental models. In this way, with SAIH, we can not only diagnose existing system performance bottlenecks, but also guide emerging HPC systems towards better AI support. In future work, we plan to enhance SAIH in the following aspects: 1) building new representative SAIH instances in various scientific domains, e.g., RNN application in genome analysis, and 2) creating fine-grained profiling components so as to identify performance bottlenecks on HPC systems.

Conflict of Interest The authors declare that they have no conflict of interest.

References

[1] Ravanbakhsh S, Oliva J B, Fromenteau S, Price L, Ho S, Schneider J G, Póczos B. Estimating cosmological parameters from the dark matter distribution. In *Proc. the 33rd International Conference on Machine Learning*, Jun. 2016, pp.2407–2416.

[2] Mathuriya A, Bard D, Mendygral P, Meadows L, Arnemann J, Shao L, He S Y, Kärnä T, Moise D, Pennycook S J, Maschhoff K, Sewall J, Kumar N, Ho S, Ringenburg M F, Prabhat P, Lee V. CosmoFlow: Using deep learning to learn the universe at scale. In *Proc. the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2018, pp.819–829. DOI: 10.1109/SC.2018.00068.

[3] Kurth T, Zhang J, Satish N, Racah E, Mitliagkas I, Patwary M M A, Malas T, Sundaram N, Bhimji W, Smorkalov M, Deslippe J, Shiryaev M, Sridharan S, Prabhat, Dubey P. Deep learning at 15PF: Supervised and semi-supervised classification for scientific data. In *Proc. the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2017, Article No. 7. DOI: 10.1145/3126908.3126916.

[4] Patton R M, Johnston J T, Young S R, Schuman C D, March D D, Potok T E, Rose D C, Lim S H, Karnowski T P, Ziatdinov M A, Kalinin S V. 167-PFlops deep learning for electron microscopy: From learning physics to atomic manipulation. In *Proc. the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2018, pp.638–648. DOI: 10.1109/SC.2018.00053.

[5] Balaprakash P, Egele R, Salim M, Wild S, Vishwanath V, Xia F F, Brettin T, Stevens R. Scalable reinforcement-learning-based neural architecture search for cancer deep learning research. In *Proc. the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2019, Article No. 37. DOI: 10.1145/3295500.3356202.

[6] Wozniak J M, Jain R, Balaprakash P, Ozik J, Collier N T, Bauer J, Xia F F, Brettin T, Stevens R, Mohd-Yusof J, Cardona C G, Van Essen B, Baughman M. CANDLE/Supervisor: A workflow framework for machine learning applied to cancer research. *BMC Bioinformatics*, 2018, 19(18): Article No. 491. DOI: 10.1186/s12859-018-2508-4.

[7] Kurth T, Treichler S, Romero J, Mudigonda M, Luehr N, Phillips E, Mahesh A, Matheson M, Deslippe J, Fatica M, Prabhat P, Houston M. Exascale deep learning for climate analytics. In *Proc. the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2018, pp.649–660. DOI: 10.1109/SC.2018.00054.

[8] Dongarra J J. The LINPACK benchmark: An explanation. In *Proc. the 1st International Conference on Supercomputing*, Jun. 1987, pp.456–474. DOI: 10.1007/3-540-18991-2_27.

[9] Mattson P, Cheng C, Diamos G F, Coleman C, Micikevicius P, Patterson D A, Tang H L, Wei G Y, Bailis P, Bittorf V, Brooks D, Chen D H, Dutta D, Gupta U, Hazelwood K M, Hock A, Huang X Y, Kang D, Kanter D, Kumar N, Liao J, Narayanan D, Oguntebi T, Pekhimenko G, Pentecost L, Reddi V J, Robie T, John T S, Wu J, Xu

L J, Young C, Zaharia M. MLPerf training benchmark. In *Proc. Machine Learning and Systems*, Mar. 2020, pp.336–349. DOI: 10.48550/arXiv.1910.01500.

[10] Zhang L, Ji Q. A bayesian network model for automatic and interactive image segmentation. *IEEE Trans. Image Processing*, 2011, 20(9): 2582–2593. DOI: 10.1109/TIP.2011.2121080.

[11] Shen Z, Bao W Z, Huang D S. Recurrent neural network for predicting transcription factor binding sites. *Scientific Reports*, 2018, 8(1): Article No. 15270. DOI: 10.1038/s41598-018-33321-1.

[12] Trabelsi A, Chaabane M, Ben-Hur A. Comprehensive evaluation of deep learning architectures for prediction of DNA/RNA sequence binding specificities. *Bioinformatics*, 2019, 35(14): i269–i277. DOI: 10.1093/bioinformatics/btz339.

[13] Lyu C, Chen B, Ren Y F, Ji D H. Long short-term memory RNN for biomedical named entity recognition. *BMC Bioinformatics*, 2017, 18(1): Article No. 462. DOI: 10.1186/s12859-017-1868-5.

[14] Karpatne A, Kumar V. Big data in climate: Opportunities and challenges for machine learning. In *Proc. the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 2017, pp.21–22. DOI: 10.1145/3097983.3105810.

[15] Zhang W, Wei W, Xu L J, Jin L L, Li C. AI matrix: A deep learning benchmark for Alibaba data centers. arXiv: 1909.10562, 2019. http://arxiv.org/abs/1909.1056, Mar. 2024.

[16] Haidar A, Tomov S, Dongarra J, Higham N J. Harnessing GPU tensor cores for fast FP16 arithmetic to speed up mixed-precision iterative refinement solvers. In *Proc. the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2018, pp.603–613. DOI: 10.1109/SC.2018.00050.

[17] Ben-Nun T, Besta M, Huber S, Ziogas A N, Peter D, Hoefler T. A modular benchmarking infrastructure for high-performance and reproducible deep learning. In *Proc. the 2019 IEEE International Parallel and Distributed Processing Symposium*, May 2019, pp.66–77. DOI: 10.1109/IPDPS.2019.00018.

[18] Deng J, Dong W, Socher R, Li L J, Li K, Fei-Fei L. ImageNet: A large-scale hierarchical image database. In *Proc. the 2009 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp.248–255. DOI: 10.1109/CVPR.2009.5206848.

[19] Zhu H Y, Akrout M, Zheng B J, Pelegris A, Phanishayee A, Schroeder B, Pekhimenko G. TBD: Benchmarking and analyzing deep neural network training. arXiv: 1803.06905, 2018. http://arxiv.org/abs/1803.06905, Mar. 2024.

[20] Farrell S, Emani M, Balma J, Drescher L, Drozd A, Fink A, Fox G C, Kanter D, Kurth T, Mattson P, Mu D W, Ruhela A, Sato K, Shirahata K, Tabaru T, Tsaris A, Balewski J, Cumming B, Danjo T, Domke J, Fukai T, Fukumoto N, Fukushi T, Gerofi B, Honda T, Imamura T, Kasagi A, Kawakami K, Kudo S, Kuroda A, Martinasso M, Matsuoka S, Mendonça H, Minami K, Ram P, Sawada T, Shankar M, John T S, Tabuchi A, Vishwanath V, Wahib M, Yamazaki M, Yin J Q. MLPerf HPC: A holistic benchmark suite for scientific machine learning on HPC systems. arXiv: 2110.11466, 2021. https://arxiv.org/abs/2110.11466, Mar. 2024.

[21] Jiang Z H, Gao W L, Tang F, Wang L, Xiong X W, Luo C J, Lan C X, Li H X, Zhan J F. HPC AI500 v2.0: The methodology, tools, and metrics for benchmarking HPC AI systems. In *Proc. the 2021 IEEE International Conference on Cluster Computing*, Sept. 2021, pp.47–58. DOI: 10.1109/Cluster48925.2021.00022.

[22] Goodfellow I, Pouget-Abadie J, Mirza M *et al*. Generative adversarial nets. In *Proc. Advances in Neural Information Processing Systems*, Dec. 2014.

[23] Sandfort V, Yan K, Pickhardt P J, Summers R M. Data augmentation using generative adversarial networks (CycleGAN) to improve generalizability in CT segmentation tasks. *Scientific Reports*, 2019, 9(1): Article No. 16884. DOI: 10.1038/s41598-019-52737-x.

[24] Sun Y, Yuan P S, Sun Y M. MM-GAN: 3D MRI data augmentation for medical image segmentation via generative adversarial networks. In *Proc. the 2020 IEEE International Conference on Knowledge Graph*, Aug. 2020, pp.227–234. DOI: 10.1109/ICBK50248.2020.00041.

[25] Milz S, Rüdiger T, Süss S. Aerial GANeration: Towards realistic data augmentation using conditional GANs. In *Proc. the European Conference on Computer Vision*, Sept. 2018, pp.59–72. DOI: 10.1007/978-3-030-11012-3_5.

[26] Jin H F, Song Q Q, Hu X. Auto-keras: An efficient neural architecture search system. In *Proc. the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Jul. 2019, pp.1946–1956. DOI: 10.1145/3292500.3330648.

[27] Liu C X, Zoph B, Neumann M, Shlens J, Hua W, Li L J, Li F F, Yuille A, Huang J, Murphy K. Progressive neural architecture search. In *Proc. the 15th European Conference on Computer Vision*, Sept. 2018, pp.19–35. DOI: 10.1007/978-3-030-01246-5_2.

[28] Tassev S, Eisenstein D J, Wandelt B D, Zaldarriaga M. sCOLA: The N-body COLA method extended to the spatial domain. arXiv: 1502.07751, 2015. https://arxiv.org/abs/1502.07751, Mar. 2024.

[29] Hahn O, Abel T. Multi-scale initial conditions for cosmological simulations. *Monthly Notices of the Royal Astronomical Society*, 2011, 415(3): 2101–2121. DOI: 10.1111/j.1365-2966.2011.18820.x.

[30] Liu H X, Simonyan K, Yang Y M. DARTS: Differentiable architecture search. arXiv: 1806.09055, 2018. https://doi.org/10.48550/arXiv.1806.09055, Mar. 2024.

[31] Kingma D P, Ba J. Adam: A method for stochastic optimization. arXiv: 1412.6980, 2015. https://arxiv.org/abs/1412.6980, Mar. 2024.

[32] You Y, Zhang Z, Hsieh C J, Demmel J, Keutzer K. ImageNet training in minutes. In *Proc. the 47th International Conference on Parallel Processing*, Aug. 2018, Article No. 1. DOI: 10.1145/3225058.3225069.

[33] Yan D, Wang W, Chu X W. Demystifying tensor cores to optimize half-precision matrix multiply. In *Proc. the 2020 IEEE International Parallel and Distributed Processing Symposium*, May 2020, pp.634–643. DOI: 10.1109/IPDPS47924.2020.00071.

Jiang-Su Du received his B.S. degree in spatial information and digital technology from Wuhan University, Wuhan, in 2016, and his M.S. degree in high-performance computing from the University of Edinburgh, Edinburgh, in 2017. He is currently a Ph.D. candidate at the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou. His research interests focus on the intersection of the high performance computing and the artificial intelligence.

Dong-Sheng Li received his B.S. degree in computer science and technology from Wenzhou University, Wenzhou, in 2019, and he is now a Master student at the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou. His research interests include high performance computing and machine learning.

Ying-Peng Wen is a Ph.D. student at the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou. Before that, he received his B.S. degree in mathematical sciences from Xiamen University, Xiamen, in 2016. His research interests include computer vision, neural architecture search, distributed computing, and reinforcement learning.

Jia-Zhi Jiang received his B.S. and M.S. degrees in computer science and technology from the South China University of Technology, Guangzhou, in 2013 and 2016 respectively. He is currently working toward his Ph.D. degree with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou. His research interests focus on parallel and distributed computing of deep learning models.

Dan Huang currently is an associate professor in the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou. He received his Ph.D. degree in computer engineering at University of Central Florida, Orlando, in 2018. Before that, he received his B.S. degree from Jilin University, Changchun, in 2007, and his M.S. degree from Southeast University, Nanjing, 2010, both in computer science and technology. His research interests are the I/O of distributed system, scientific data management, parallel programming model, distributed storage systems, in-memory computing, and virtualization technology.

Xiang-Ke Liao received his B.S. degree from Tsinghua University, Beijing, in 1985, and his M.S. degree from National University of Defense Technology, Changsha, in 1988, both in computer science. Currently he is a full professor at the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou. His research interests include high-performance computing systems, operating systems, and parallel and distributed computing. Prof. Liao is a fellow of CCF and an academician of Chinese Academy of Engineering.

Yu-Tong Lu received her B.S. M.S. and Ph.D. degrees all in computer science from the National University of Defense Technology, Changsha. She is now a professor at the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou. Her research interests include parallel system management, high-speed communication, distributed file systems, and advanced programming environments with MPI (message passing interface).