# Cross-Search With Improved Multi-Dimensional Dichotomy-Based Joint Optimization for Distributed Parallel Training of DNN

Guangyao Zhou, Yiqin Fu, Haocheng Lan, Yuanlun Xie, Wenhong Tian, *Member, IEEE*, Rajkumar Buyya, *Fellow, IEEE*, Jianhong Qian, and Teng Su

***Abstract*—Distributed parallel training of large-scale deep neural networks (DNN) has attracted the attentions of both artificial intelligence and high-performance distributed computing. One of efficient approaches is the micro-batch-based pipeline parallelism (MBPP), e.g., GPipe and Terapipe. Based on the MBPP, we establish a time-cost model with the basic time function of layers, which considers computing time and communication time simultaneously as well as considers they are nonlinear with the amount of input data. Focusing on the jointly optimal solutions of network division and data partition, we propose a Cross-Search algorithm with Improved Multi-dimensional Dichotomy (CSIMD). Through theoretical derivation, we prove improved multi-dimensional dichotomy (IMD) has appreciable theoretical optimality and linear computational complexity significantly faster than the state-of-the-art methods including dynamic programming and recursive algorithm. Extensive experiments on both CNN-based and transformer-based neural networks demonstrate our proposed CSIMD can obtain optimal network division and data partition schemes under MBPP. On average, the training speeds of CSIMD in CNN- and transformer-based DNNs are respectively $(2.0, 2.5)\times$ and $(2.66, 5.48)\times$ of (MBPP-R, MBPP-E).**

***Index Terms*—Distributed parallelism, micro-batch pipeline, cross-search, improved multi-dimensional dichotomy, large DNN, transformer.**

## I. Introduction

Deep learning (DL) is showing a significant trend that the scale of model parameters continues to increase [1].

Received 3 December 2024; revised 12 June 2025; accepted 12 June 2025. Date of publication 16 June 2025; date of current version 26 June 2025. This work was supported in part by the Fundamental Research Funds for the Central Universities under Grant 2682025CX127, in part by Project of Key Research and Development Program of Sichuan Province under Grant 2021YFG0325, and in part by Technical Cooperation Project of Huawei under Grant ID H04W220751. Recommended for acceptance by R. Prodan. *(Corresponding author: Guangyao Zhou.)*

Guangyao Zhou is with the School of Computing and Artificial Intelligence, Southwest Jiaotong University, Chengdu 610032, China (e-mail: guangyao_zhou@swjtu.edu.cn).

Yiqin Fu is with China National Petroleum Corporation, Beijing 100007, China.

Haocheng Lan, Yuanlun Xie, and Wenhong Tian are with the School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 610056, China (e-mail: tian_wenhong@uestc.edu.cn).

Rajkumar Buyya is with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, University of Melbourne, Melbourne, VIC 3010, Australia (e-mail: rbuyya@unimelb.edu.au).

Jianhong Qian and Teng Su are with Huawei Technologies Company Ltd, Shenzhen 518129, China.

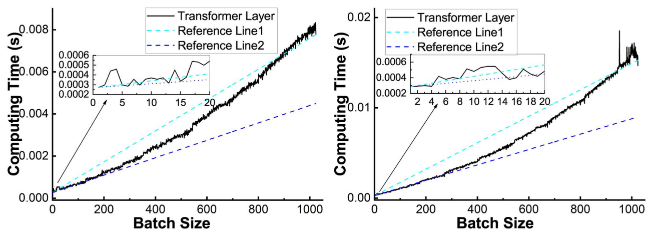
Digital Object Identifier 10.1109/TPDS.2025.3580098

The recent large deep neural networks (DNN), especially large language models (LLMs), have hundreds of billions of parameters, e.g., FLAN with 137B parameters [2], GPT-3 with 175B parameters [3], Gopher with 280B parameters [4], and Llama series with above 70B parameters [5], [6].

In practice, training large DNNs requires cooperative and parallel work of numerous distributed artificial intelligence (AI) devices (e.g., GPU cluster), which usually consume much time [7], [8]. How to improve the efficiency of parallel training becomes a hotspot in the fields of AI and distributed computing [7], [9]. The optimization of parallel training is a complex NP-Hard problem where the solution space (scheme space) is often an uncountable set [7], [10]. To obtain some approximate schemes, the community established data parallelism (DP), tensor model parallelism (TMP) and pipeline model parallelism (PMP) as three foundational policies [11], [12], [13], which directs current parallel training architectures. Currently, some well-performed parallel training architectures belong to micro-batch-based pipeline parallelism (MBPP), such as GPipe [14], Dapple [15], Hippie [11] and Terapipe [13].

Data partition and network division are two critical factors of parallel training especially for MBPP [16], [17], which directly affects the schedulable granularity of computing process and communication process [13]. Data partition contains two forms: dividing a mini-batch into multiple micro-batches and dividing a sequence into multiple micro-sequences (or called micro-tokens). Network division in pipeline parallelism means dividing multiple layers of a DNN into several continuous subsegments and deploying them on multiple devices (i.e., multiple stages). Improving schedulable granularity allows more overlap between the time of various processes and can improve feasible solutions of parallel schemes [15], [18], [19].

However, the finer granularity of data partition or network division may not lead to better training performance. It is because of the existence of nonlinear relationship between time consumption and data volume (or model parameter volume), which means excessive splitting of network layers or data may introduce additional time cost instead. Therefore, solving the optimization problems of data partition and network division based on a more realistic theoretical cost model showcases significance. To simplify the modeling and solving of optimal parallel schemes, the existing studies usually assumed computing time and communication time proportional to data



(a) GPT-2: 774M (seq length is 4) (b) GPT-2: 774M (seq length is 8)

Fig. 1. The computing time of one layer in LLMs with different data sizes.

volume [16], [20], [21], [22]. However, when using practical GPUs and communication networks to implement parallel training, the computing time or communication time is probably not proportional to their data volume [10], [23]. For example, in Fig. 1: although the computing time shows an approximately linear trend with the batch size (data volume) in some local ranges, there are some ranges in which the trend is nonlinear. The nonlinear relationship mainly stems from two reasons: when the data volume is reduced to a certain extent, processing small data also needs to complete some inherent operations; the devices generally have a certain parallel processing capability, which means the processing time of the data volume within the parallel capability has relatively constant trend, while the data beyond the parallel capability will enter the queue for serial execution. Therefore, improving parallel training requires finding the optimal schemes of network division and data partition according to the cost model considering the non-linear relationships.

To find the optimal schemes of data parallelism or model parallelism, some recent research mainly leveraged dynamic programming, linear programming, etc [15], [24]. However, in the optimization problems of MBPP with multiple parallel dimensions, the data partition and network division belong to heterogeneous decision variables, which indicates that a single algorithm is unable to simultaneously solve their optimal solutions. As the solution space generally increases exponentially with large numbers of model parameters and devices, the existing methods have massive computational complexity [20], [22], [25]. Additionally, the lack of accurate analysis models and formulas (especially lack of models considering nonlinear relationship) makes it difficult to evaluate the performance of parallel schemes during optimization. The above characteristics are challenging the optimization of MBPP.

Aiming at addressing the above challenges of MBPP, we derive and construct a theoretical time-cost model, which takes the schemes of network divisions and data partitions as variables, as well as considers the nonlinear relationship between time consumption and data volume. For the sake of solving the optimal schemes of MBPP, we formulate the optimization problem as a joint optimization problem with two subproblems including: solving the optimal network division schemes under the fixed data partitioning number, which can be formulated as a multi-dimensional array segmentation; solving the optimal data partitioning number under the fixed network division. Based on formulations of joint problem with multiple subproblems, the key to improving MBPP is to solve two subproblems and the joint problem respectively. To solve the subproblem of network division, we propose an improved multi-dimensional dichotomy (IMD) to obtain the optimal schemes in linear time complexity, by converting the subproblem into the ordered bin-packing problem of multi-dimensional arrays. For the subproblem of data partitions, we propose a fast optimal data partition algorithm (ODPA) based on matrix operations. Then, in order to solve the joint problem with heterogeneous decision variables, we propose CSIMD (a cross-search algorithm based on IMD and ODPA).

The main contributions are summarized as follows.

1) *Theoretical models:* to support the optimization of MBPP, we derive a theoretical cost model regarding heterogeneous decision variables of DP and PMP. Our cost model not only considers computing time and communication time simultaneously, but also considers the nonlinear relationship between time consumption and data size, which conforms to reality. With cost model, we convert the optimization problem of MBPP to a joint problem with multiple subproblems: solving the optimal network division under the fixed data partition, and solving the optimal data partition under the fixed network division.
2) *IMD and ODPA for subproblems:* to solve optimal network division (an ordered multi-dimensional array segmentation problem), we propose an improved multi-dimensional dichotomy (IMD). Through theoretical derivation, we prove that IMD has a theoretical approximation ratio close to 1 under linear time complexity. To solve optimal data partition, we propose a fast optimal data partition algorithm (ODPA) based on matrix operations.
3) *CSIMD for the joint problem:* we propose a cross-search framework (CS) to solve joint optimization problems containing heterogeneous decision variables by alternately solving subproblems. Combining joint optimization framework (cross-search) and sub optimization algorithms (IMD and ODPA), we construct an overall optimization algorithm (CSIMD) for simultaneously solving the multi-dimensional parallel schemes in MBPP.
4) Extensive experiments in real GPU cluster not only demonstrate the optimality and fastness of our IMD, but also demonstrate the optimality of CSIMD for both CNN- and transformer-based DNNs under MBPP.

The rest of this paper is organized as follows. We review the related work in Section II. The formulation and analysis of the time cost model are derived in Section III. We propose the methodology and present its theoretical analysis in Section IV. The evaluation results are presented in Section V. Finally, we conclude this paper in Section VI.

## II. Related Work

In this section, we mainly review three aspects that are related to our research in this paper: parallelism, mathematical cost model, and optimization algorithms.

Data parallelism and model parallelism are two basic modes of parallel training, which derive various new parallelism [11],

[24]. Data parallelism partitions training data into multiple pieces [26], [27] and model parallelism divides the DNN model into multiple parts [11], [20], [28]. Combining data parallelism and model parallelism, one important series is micro-batch-based pipeline parallelism (MBPP). GPipe [14] divided mini-batch data into multiple micro-batches, which allowed the computation and communication of different stages corresponding to different model nodes (on different devices) can overlap. The more temporal overlapping parts of various processes on different devices correspond to the less idle time (bubbles) of devices [24]. On the premise that the pipeline does not introduce redundant computing and communication costs, the total training time of GPipe is smaller than the original pipeline. Based on GPipe, PipeDream [29] added a strategy, i.e., shifting the gradient backward-propagation (BP) earlier to the moment immediately after its last part of forward-propagation (FP). Other well-performed methods or parallelism, including Dapple [15], Hippie [11], TeraPipe [13], NasPipe [30], et al. are all the variants of GPipe or PipeDream based on MBPP. Terapipe [13] with micro-tokens-based data parallelism followed GPipe and improved the granularity to reduce the pipeline bubbles of the transformer-based NLP model by proposing a new dimension, i.e., token dimension.

The mathematical cost model is also an important factor of parallel training, which is the basis for optimizing parallel training schemes. Next, we mainly review the time-cost model of MBPP in existing research. The complexity of parallel training, makes it difficult to obtain an accurate expression of the cost model. GPipe did not consider the corresponding time-cost model in [14]. In the subsequent studies [13], [16], [20], the cost model was considered as:

$$T_{PP} = (m-1)\cdot(\max(F_i) + \max(B_i)) + \sum (F_i + B_i) \tag{1}$$

where $T_{PP}$ means the total training time for one mini-batch in one iteration when using MBPP, $F_i$ and $B_i$ are the time respectively for FP and BP of the $i$-th stage, and $m$ is the number of micro-batches in one mini-batch. The cost model of PipeDream was considered as a recursive formula [29]. Narayanan et al. [31] proposed PipeDream-2BW and considered a time-cost model for pipelining as

$$T_{PP} = \max_i \left( \max \left( T_i^{cp} + \sum_j T_{j\to i}^{cm}, \frac{1}{m}\cdot T_i^{cm} \right) \right),$$

where $T_i^{cp}$ means the computing time of the $i$-th stage, $T_{j\to i}^{cm}$ means the communication time between stages $i$ and $j$, and $T_i^{cm}$ means the communication time of exchanging gradients. PipePar [32] considered a cost model as

$$T_{PP} = T_{cm} + T_{cp},$$

which directly adds communication and computing time. Narayanan et al. [33] proposed Megatron-LM and considered a cost model as

$$T_{PP} = C\cdot(F+B)),$$

where $C$ is a coefficient related to micro-batch size and data parallel size. Elango [22] proposed PaSE and considered the FLOP-to-bytes ratio in the cost model.

Because the cost model was generally non-analytical or recursive, dynamic programming is widely used to obtain the optimal partition of data parallelism or model parallelism [24]. Some examples include PipeDream [15], Dapple [15], Terapipe [13], EffTra [16], Alpa [20], PaSE [22], PipePar [32]. Linear programming is also a frequent method in parallel training [9], [27], [34]. Some examples include NetPlacer [34], HGP4CNN [9], DPDA [27]. Other partition methods include off-the-shelf graph partitioning algorithms [35], recurrence [22], multi-chromosome genetic algorithm [36], grouping genetic algorithm [37], minimum vertex-cut graph partitioning algorithm [38], near-optimal layer partition of local search [25]. As the state-of-the-art methods to obtain optimal network division schemes, dynamic programming and recursive algorithms suffered from their large computational complexity, which is a quadratic polynomial about the number of layers (denoted as $K$) and a linear polynomial about the number of stages (denoted as $N$), i.e., $\mathbf{O}(2NK^2)$.

From the literature review, the formulation of the cost model affected the choice of optimal algorithms. Some of the cost models only considered one of computation and communication processes; some only provided recursive formulas; and some with direct expressions relied on a lot of ideal assumptions which was far away from the real scenario. Referring to but distinguished from the existing research, this paper derives a time-cost model of MBPP considering computation and communication simultaneously. The model doesn't limit the features of devices, so it adapts to both homogenous and heterogeneous systems. The cost model also considers the nonlinear relationship between cost and data size, which is closer to reality. To solve the parallel training schemes, we proposed a novel method that is cross-search with improved multi-dimensional dichotomy (CSIMD). Compared with the state-of-the-art methods including dynamic programming and recursive algorithm, our proposed IMD is far faster in solving the network division. IMD's time complexity is linear with $K$ and decreases as $N$ increases.

## III. Theoretical Formulations of MBPP

For the sake of derivation and analysis for the cost model of MBPP, we list some notations in Table I.

In this paper, we mainly focus on the structures of MBPP (e.g., GPipe with micro-batch data and Terapipe with micro-tokens data) shown in Fig. 2. The characteristic of MBPP is that the BPs of micro-batches in each mini-batch need to start after the FP of the last micro-batch in the last layer ends. To simplify the structure of MBPP so that the analytical formulas of time-cost are obtainable, we consider that the receiving data and sending data of the same device do not affect each other, otherwise, the recursive formula will be more complex. This consideration is consistent with the full-duplex communication mode which is widely used in distributed systems [18]. Thus, Fig. 2 only needs to present the process of devices in sending data, without drawing the process of receiving data.

TABLE I
NOTATIONS AND DESCRIPTIONS

| Notation | Description |
|---|---|
| $N$ | Number of stages |
| $p$ | Number of partitions |
| $K$ | Number of layers of DNN |
| $F_i^P(p)$ | The forward computing time of one micro-batch in the $i$-th stage when partitioning the mini-batch into $p$ micro-batches |
| $F_i^M(p)$ | The forward communication time of one micro-batch |
| $B_i^P(p)$ | The backward computing time of one micro-batch |
| $B_i^M(p)$ | The backward communication time of one micro-batch |
| $S_{ij}^P(p)$ | The start computing time of the $j$-th partition in the $i$-th stage of FP |
| $S_{ij}^M(p)$ | The start communication time of the $j$-th partition in FP |
| $R_{ij}^M(p)$ | The start computing time of the $j$-th partition in BP |
| $R_{ij}^M(p)$ | The start communication time of the $j$-th partition in BP |
| $H_i(p)$ | The computing time of the $i$-th layer when partitioning the mini-batch into $p$ micro-batches |
| $J_i(p)$ | Time required to communicate the output data of the $i$-th layer |
| $L_i$ | The $i$-th layer |
| $C_i$ | The collection of layer on the $i$-th device |
| $\alpha_i$ | The maximum index of layers in $C_i$ |
| $\lambda$ | The collection of $\alpha_i$ as $\lambda = \langle \alpha_1, \dots, \alpha_N \rangle$ |

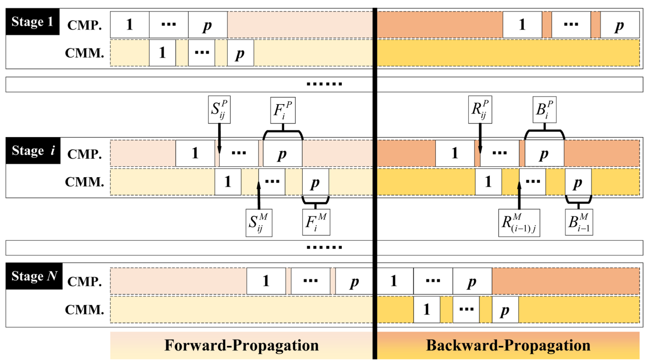


Fig. 2. The parallel structures of the MBPP: CMP. means computing process, and CMM. means communication process.

## A. Cost Model Considering Computation and Communication

For MBPP, the expression of time is as (1) without consideration of communication. While, it is not enough to support the cost model considering computation and communication simultaneously. In order to support subsequent research on data partitioning and network division, we need to formulate a more comprehensive cost model considering computation and communication simultaneously.

The computing time and communication time of one micro-batch (or micro-tokens) can be set as functions in terms of data partitions: $F_i^P(p)$ and $F_i^M(p)$ are respectively the computing time and communication for FP of one micro-batch (or one micro-token) in the $i$-th stage, where $p$ is the number of data partitions; $B_i^P(p)$ and $B_i^M(p)$ are that for BP. The processing time of different data partitions in the same stage is relatively stable. It can be set that the $S_{ij}^P(p)$ is the start computing time of the $j$-th partition in the $i$-th stage of FP and $S_{ij}^M(p)$ is the start communication time; $R_{ij}^P(p)$ and $R_{ij}^M(p)$ are for BP.

When the functions $F_i^P(p)$, $F_i^M(p)$, $B_i^P(p)$ and $B_i^M(p)$ are given, $S_{ij}^P(p)$, $S_{ij}^M(p)$, $R_{ij}^P(p)$ and $R_{ij}^M(p)$ can completely represent the whole process of parallel training in MBPP. Since these variables are functions of partition number $p$, we omit the $(p)$ in the subsequent expression. In MBPP, the BP in the same mini-batch needs to wait for all FPs to complete before starting. Then, the recursive formula can be written as (2) according to its characteristics.

$$\begin{cases} S_{ij}^P = \max\left(S_{(i-1)j}^M + F_{i-1}^M, S_{i(j-1)}^P + F_i^P\right) \\ S_{ij}^M = \max\left(S_{ij}^P + F_i^P, S_{i(j-1)}^M + F_i^M\right) \\ R_{ij}^P = \max\left(R_{ij}^M + B_i^M, R_{i(j-1)}^P + B_i^P\right) \\ R_{ij}^M = \max\left(R_{(i+1)j}^P + B_{i+1}^P, R_{i(j-1)}^M + B_i^M\right) \end{cases} \tag{2}$$

where $1 \le i \le N$ is the index of stage, $N$ is the number of stages, $1 \le j \le p$ is the index of micro-batch (or micro-token) in one mini-batch. The recursive formulas of (2) illustrate that the start time of each micro-batch data process (including micro-batch token process) depends on the end time of the two preamble processes: micro-batch process with the same index in the previous stage, and the previous micro-batch process in this stage. Taking computation process of FP as an example, the computation process of the $j$-th data partitions in the $i$-th stage (corresponding to the start time $S_{ij}^P$) needs to be simultaneously after the communication process of the $j$-th data partition in the $(i-1)$-th stage (corresponding to the end time $S_{(i-1)j}^M + F_{i-1}^M$) and the computation process of $(j-1)$-th data partitions in the $i$-th stage (corresponding to the end time $S_{i(j-1)}^P + F_i^P$). The recurrence relationship of other processes is similar to the above example. If $i \notin [1, N]$ or $j \notin [1, p]$, then $S_{ij}^P(p) = R_{ij}^P = -\infty$. And if $i \notin [1, N-1]$ or $j \notin [1, p]$, then $S_{ij}^M = R_{ij}^M = -\infty$. The initial conditions of (2) are:

$$S_{11}^P = 0, \quad R_{N1}^P = S_{Np}^P + F_N^P \tag{3}$$

Substituting (3) into (2) obtains the expressions of FP:

$$\begin{cases} S_{ij}^P = \sum_{k=1}^{i-1}\left(F_k^P + F_k^M\right) \\ \qquad + (j-1)\max\left(\max_{1\le k\le i-1}\left(F_k^P, F_k^M\right), F_i^P\right) \\ S_{ij}^M = \sum_{k=1}^{i}\left(F_k^P + F_k^M\right) - F_i^M \\ \qquad + (j-1)\max_{1\le k\le i}\left(F_k^P, F_k^M\right) \end{cases} \tag{4}$$

where $1 \le i \le N$ and $1 \le j \le p$. The expressions of BP are:

$$\begin{cases} R_{(N-i)j}^P = S_{Np}^P + F_N^P + \sum_{k=1}^{i-1}\left(B_{N-k}^P + B_{N-k}^M\right) + B_N^P \\ \qquad + (j-1)\max\left(\max_{1\le k\le i-1}\left(B_{N-k}^P, B_{N-k}^M\right), B_N^P\right) \\ R_{(N-i)j}^M = S_{Np}^P + F_N^P + \sum_{k=1}^{i}\left(B_{N-k+1}^P + B_{N-k}^M\right) \\ \qquad + (j-1)\max_{1\le k\le i}\left(B_{N-k+1}^P, B_{N-k}^M\right) \end{cases} \tag{5}$$

Then, the time-cost for one iteration of one mini-batch is:

$$\begin{aligned} T = R_{1p}^P + B_1^P &= \sum_{k=1}^{N}\left(F_k^P + F_k^M + B_k^P + B_k^M\right) \\ &+ (p-1)\max_{1\le k\le N}\left(\max\left(F_k^P, F_k^M\right)\right) \\ &+ (p-1)\max_{1\le k\le N}\left(\max\left(B_k^P, B_k^M\right)\right) \end{aligned} \tag{6}$$

where $F_N^M = B_N^M = 0$. From (6), the training time (makespan) of MBPP consists of three parts: the sum of the calculation time and communication time of one micro-batch in

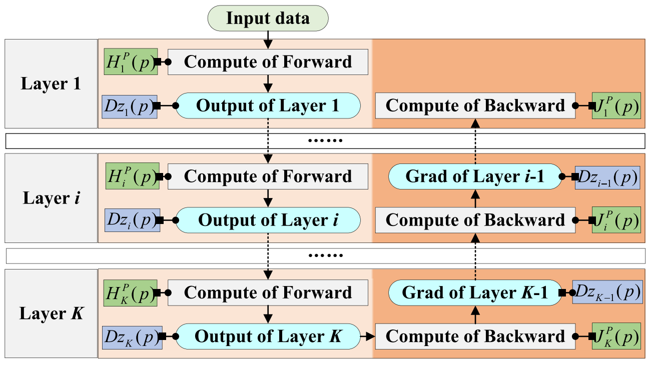


Fig. 3. The diagram of the whole computing process and the corresponding symbols for training one micro-batch of DNN.

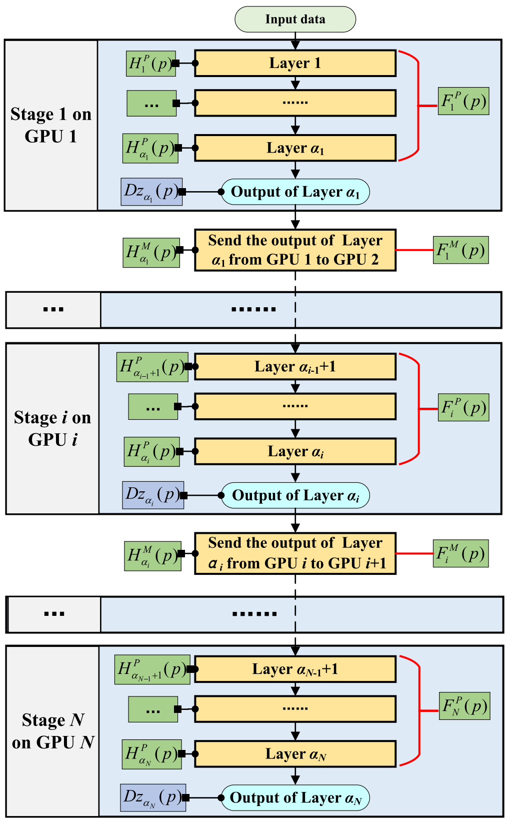

Input data
Stage 1 on GPU 1
Layer 1
Layer $\alpha_1$
Output of Layer $\alpha_1$
Send the output of Layer $\alpha_1$ from GPU 1 to GPU 2
Stage i on GPU i
Layer $\alpha_{i-1}$+1
Layer $\alpha_i$
Output of Layer $\alpha_i$
Send the output of Layer $\alpha_i$ from GPU i to GPU i+1
Stage N on GPU N
Layer $\alpha_{N-1}$+1
Layer $\alpha_N$
Output of Layer $\alpha_N$

Fig. 4. The diagram of network division ($K$ layers to $N$ stages) in FP.

all pipeline stages (i.e., $\sum_{k=1}^{N}(F_k^P + F_k^M + B_k^P + B_k^M)$); the computation time or communication time of the longest micro-batch data in FP multiplied by $(p-1)$ (i.e., $(p-1)\max_{1\le k\le N}(\max(F_k^P, F_k^M))$); the time of the longest micro-batch data in BP multiplied by $(p-1)$ (i.e., $(p-1)\max_{1\le k\le N}(\max(B_k^P, B_k^M))$). Compared to (1), (6) takes the computations and communications into consideration simultaneously. The derivation process of (6) only depends on the recursive relationship between the micro-batch processes in MBPP, and does not limit the form of the basic time functions and the device nodes. Therefore, (6) is applicable to scenarios considering both nonlinear time relationships and clusters combining various devices.

## B. Theoretical Analysis of Cost Model

In parallel training of MBPP, two aspects need to be optimized: dividing DNN into $N$ stages (network division) and finding the optimal number of data partitions.

If $F_i^P(p)$, $F_i^M(p)$, $B_i^P(p)$ and $B_i^M(p)$ are given, the optimal partition number can be obtained by the extreme values of (6). Thus, we first discuss network division which needs to be determined before solving the optimal data partition.

It can be set that the DNN has $K$ layers denoted as $L = \langle L_1, L_2, \ldots, L_K\rangle$ where $L_i$ corresponds to the $i$-th layer. In this paper, we mainly discuss the situation that $K \ge N$ and the network layers within the same device must be continuous, which means each device must at least contain one layer of network. For the sake of analysis, we also set that the computing time and communication time (time required to communicate its output data to the next layer) for FP of the $i$-th layer are respectively $H_i^P(p)$ and $H_i^M(p)$ where $1 \le i \le K$. That for the BP are set as $J_i^P(p)$ and $J_i^M(p)$. Thus, we can plot a diagram of the whole computation process and the corresponding symbols for one micro-batch as Fig. 3.

The division of the network layer is actually to determine on which device each layer is executed on. It can be set that the collection of layers on the $i$-th device is $C_i$, i.e., $L_j \in C_i$ means the $j$-th layer is executed on the $i$-th device. The MBPP structure divides the network layer in order, thus: if $L_{i_1} \in C_j$ and $L_{i_2} \in C_j$ where $i_1 \le i_2$, then $L_i \in C_j$ for $i_1 \le \forall i \le i_2$. Therefore, we can set the maximum index of layers in $C_i$ as $\alpha_i$ where $\alpha_{i-1} < \alpha_i$. This means if $\alpha_{i-1} < j \le \alpha_i$ then $L_j \in C_i$, else $L_j \notin C_i$. It also means $C_j = \langle L_{\alpha_{i-1}+1}, L_{\alpha_{i-1}+2}, \ldots, L_{\alpha_i}\rangle$. Then, we can use $H^P$, $H^M$, $J^P$ and $J^M$ to express the $F^P$, $F^M$, $B^P$ and $B^M$ as (7).

$$\begin{cases} F_i^P = \sum_{k=\alpha_{i-1}+1}^{\alpha_i} H_k^P, \; F_i^M = H_{\alpha_i}^M \\ B_i^P = \sum_{k=\alpha_{i-1}+1}^{\alpha_i} J_k^P, \; B_i^M = J_{\alpha_i+1}^M \end{cases} \tag{7}$$

where $F_i^P = \sum_{\alpha_{i-1}+1}^{\alpha_i} H_k^P$ means the computing time of $i$-th pipeline stage equals to the sum of computing time of all the layers in this stage; $F_i^M = H_{\alpha_i}^M$ means the communication time of $i$-th pipeline stage equals to the time to transmit the output data of the last layer (i.e., $L_{\alpha_i}$) in this stage.

For the sake of presentation of the relationship between pipeline stages and layers of DNN, we plot the diagram of network division with FP in Fig. 4. The process of BP is analogous to that of FP. Substituting (7) into (6) can obtain the expression of training time regarding decision variables of network division (i.e., $\alpha_i$) and data partition (i.e., $p$):

$$T = \sum_{i=1}^{K}\left(H_i^P + J_i^P\right) + \sum_{i=1}^{N-1}\left(H_{\alpha_i}^M + J_{\alpha_i+1}^M\right) + (p-1)\left(\rho_1(\lambda, p) + \rho_2(\lambda, p)\right) \tag{8}$$

where $\rho_1 = \max_{i=1}^{N}(\max(\sum_{k=\alpha_{i-1}+1}^{\alpha_i} J_k^P, J_{\alpha_i+1}^M))$ and $\rho_2 = \max_{i=1}^{N}(\max(\sum_{k=\alpha_{i-1}+1}^{\alpha_i} H_k^P, H_{\alpha_i}^M))$, respectively corresponding to the longest FP and BP pipeline stages.

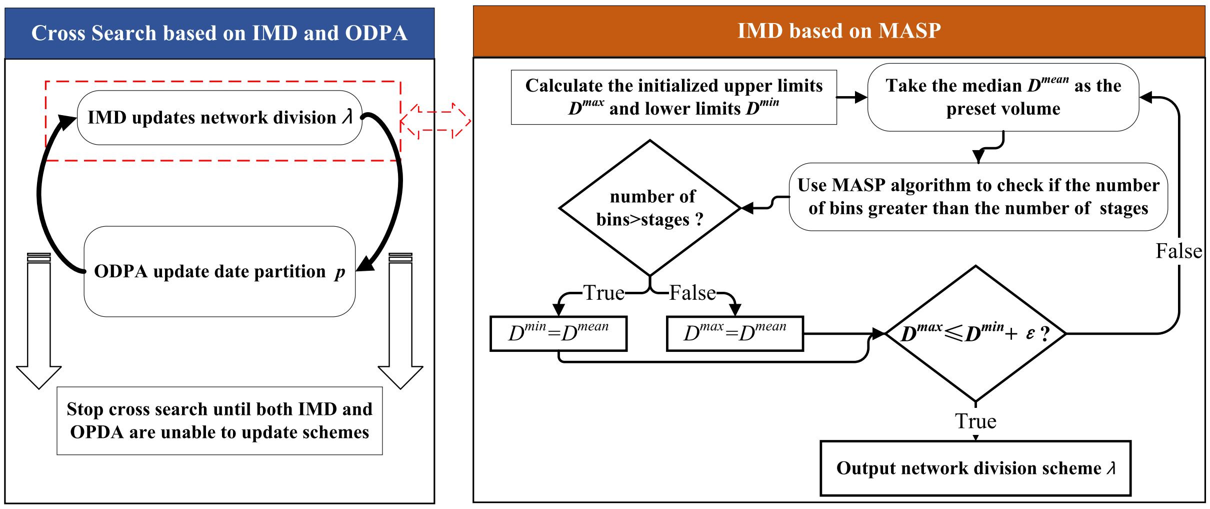

Cross Search based on IMD and ODPA
IMD updates network division λ
ODPA update date partition p
Stop cross search until both IMD and OPDA are unable to update schemes
IMD based on MASP
Calculate the initialized upper limits $D^{max}$ and lower limits $D^{min}$
Take the median $D^{mean}$ as the preset volume
Use MASP algorithm to check if the number of bins greater than the number of stages
number of bins>stages ?
True
False
$D^{min}=D^{mean}$
$D^{max}=D^{mean}$
$D^{max}\leqslant D^{min}+\varepsilon$ ?
False
True
Output network division scheme λ

Fig. 5. CSIMD framework of joint optimization for parallel training.

Therefore, the key to solving the problem of minimizing total training time is to find the optimal collection of $\lambda = \langle \alpha_1, \alpha_2, \ldots, \alpha_N \rangle$ and $p$. In the scenario of this paper, $\lambda$ and $p$ necessarily and sufficiently correspond to a unique parallel training scheme under MBPP. Thus, we can use the vector $\langle \lambda, p \rangle$ to represent the joint solution of a parallel training problem and use $T(\lambda, p)$ to represent its corresponding training time under MBPP for one mini-batch.

When $p$ is given, $\sum_{i=1}^{K} (H_i^P + J_i^P)$ is a constant. Thus, the problem can be transformed into the balancing division of layers (a multi-dimensional array segmentation problem). If only considering the computing of FP, the problem is converted to minimizing $\max_{i=1}^{N}(\sum_{k=\alpha_{i-1}+1}^{\alpha_i} H_k^P)$, which is segmenting $K$ numbers to $N$ groups without changing orders to minimize the maximum sum of these groups (one-dimensional array segmentation problem). However, considering communication and computation simultaneously or considering FP and BP simultaneously will increase the complexity significantly, because changing network division will also change the layers involved in communication. Additionally, it will change both $\max_{i=1}^{N}(\sum_{k=\alpha_{i-1}+1}^{\alpha_i} H_k^P)$ of FP and $\max_{i=1}^{N}(\sum_{k=\alpha_{i-1}+1}^{\alpha_i} J_k^P)$ of BP, hence requiring a method to simultaneously optimize data partition and network division (jointly optimizing the parallel schemes of DP and PMP).

## IV. Methodology

Based on the theoretical formulation and analysis in the above section, the problem of solving the optimal training scheme for MBPP can be transformed into three aspects:

1) $\omega_1$: solving optimal $\lambda = \langle \alpha_1, \alpha_2, \ldots, \alpha_N \rangle$ (network division) of PMP with given $p$ (partition number) of DP;
2) $\omega_2$: solving optimal $p$ (partition number) of DP with given $\lambda = \langle \alpha_1, \alpha_2, \ldots, \alpha_N \rangle$ (network division) of PMP;
3) Jointly solving problem $\omega_1$ and problem $\omega_2$ to obtain the optimal parallel training scheme $\langle \lambda, p \rangle$.

To achieve the solutions of the above three aspects, we proposed CSIMD (cross-search with improved multi-dimensional dichotomy algorithm), whose framework is presented in Fig. 5. As shown in Fig. 5, CSIMD contains three key components: improved multi-dimensional dichotomy (IMD) for $\omega_1$, optimal data partition algorithm (ODPA) for $\omega_2$, and cross-search for jointly solving $\omega_1$ and $\omega_2$, respectively corresponding to the above three aspects. In this section, we will detail the corresponding methods to solve these three aspects.

### A. *Cross-Search for Joint Solution of $\omega_1$ and $\omega_2$*

First, we discuss the systematic method, i.e., cross-search for a joint solution of $\omega_1$ and $\omega_2$. Supposing two algorithms, denoted as $A_1$ and $A_2$, can respectively obtain the theoretical optimal solution of problems $\omega_1$ and $\omega_2$, two avenues to get the joint solution of $\omega_1$ and $\omega_2$ are:

1) Traversing all feasible partition numbers to find the corresponding optimal network divisions, and then comparing their solutions to obtain the optimal scheme;
2) Traversing all network divisions to find their corresponding optimal partition number, and then comparing their solutions to obtain the optimal scheme.

However, these two traversal avenues both need to consume a lot of computational complexity. To improve the speed for jointly solving $\omega_1$ and $\omega_2$ with maintaining the optimality, we propose a cross-search algorithm whose pseudo-code can be seen in Algorithm 1:

1) First, set an initial data partition $p$ (Line 1), and obtain the corresponding initial network divisions $\lambda$ (Line 2).
2) The loop from Line 3 to Line 10 is the cross-search to update the optimization schemes of data partitions and network divisions alternately, until there is no better network layer division and no better partition number.

**Algorithm 1:** Cross-Search Algorithm for Joint Solution of $\omega_1$ and $\omega_2$.

```
Input  : $K$, $N$, The functions of $H_i^P$, $H_i^M$, $J_i^P$, and $J_i^M$ for $\forall 1 \le i \le K$
Output: Solution $\langle \lambda, p \rangle$ of parallel training and its corresponding training time $T(\lambda, p)$
1  Set $p = 1$ or other initial number
2  Use algorithm $A_1$ to obtain the optimal network division $\lambda = \langle \alpha_1, \alpha_2, \ldots, \alpha_N \rangle$ under $p$ and record its corresponding training time $T(\lambda, p)$
3  while True do
4      Use algorithm $A_2$ to obtain the optimal data partition number $p'$ under $\lambda$ and obtain its corresponding training time as $T(\lambda, p')$
5      if $T(\lambda, p') = T(\lambda, p)$ then
6          Break
7      Use algorithm $A_1$ to obtain the optimal network division $\lambda'$ under $p'$ and obtain $T(\lambda', p')$
8      if $T(\lambda', p') = T(\lambda, p')$ then
9          Break
10     Update $p = p'$ and $\lambda = \lambda'$
```

By switching the sub optimization problems between $\omega_1$ and $\omega_2$, the cross-search of Algorithm 1 enables joint optimization of network division and data partition.

With the framework of the systematic method (cross-search), two algorithms $A_1$ and $A_2$ which can respectively solve problems $\omega_1$ and $\omega_2$ are significantly required. Next, we will discuss them successively.

**Algorithm 2:** Improved Dichotomy to Solve Multi-Dimensional Array Segmentation (IMD).

```
Input  : $K$, $N$, $p$, The functions of $H_i^P$, $H_i^M$, $J_i^P$, and $J_i^M$ for $\forall 1 \le i \le K$
Output: Solution $\lambda = \langle \alpha_1, \alpha_2, \ldots, \alpha_N \rangle$ and $T(\lambda, p)$
1  Set a small value $\epsilon$ as the convergence standard
2  Set $D^{min} = \max\left(\max\left(H^P, J^P\right), \min\left(H^M, J^M\right)\right)$,
   $D_1 = \sum_{i=1}^{K}\left(H_i^P + H_i^M\right)$, $D_2 = \sum_{i=1}^{K}\left(J_i^P + J_i^M\right)$,
   $D^{max} = D_1 + D_2$
3  Set the weight groups with $\eta + 1$ (or other given number) groups $W = \langle w_0, w_1, w_2, \ldots, w_\eta \rangle$ where
   $w_i = \langle w_i(1), w_i(2) \rangle = \left\langle \frac{i}{\eta}, 1 - \frac{i}{\eta} \right\rangle$
4  while $D^{max} - D^{min} \ge \epsilon$ do
5      Set $D^{mean} = \frac{D^{max} + D^{min}}{2}$, $check = False$
6      for $w$ in $W$ do
7          Set $v_1 = w(1) \cdot D^{mean}$, $v_2 = w(2) \cdot D^{mean}$
8          Substitute $v_1$ and $v_2$ into multi-dimensional array segment packing algorithm (Algorithm 3) to obtain $\tau$ and $\lambda^{mean}$
9          if $\tau \le N$ then
10             $check = True$, $\lambda = \lambda^{mean}$
11             Break the "for" loop
12     if $check$ then
13         $D^{max} = D^{mean}$
14     else
15         $D^{min} = D^{mean}$
16 Calculate the training time $T(\lambda, p)$
```

## B. IMD to Divide Network Layers

*1) Algorithm and Framework:* The problem $\omega_1$, that solving the optimal network division with a given partition number, is a variant of the array-balanced segmentation problem, i.e., multi-dimensional array segmentation problem. The typical array balanced segmentation problem only considers a one-dimensional array. For example, when only considering computing of BP, the objective of problem $\omega_1$ can be simplified to $\min(\max_{i=1}^{N}(\sum_{k=\alpha_{i-1}+1}^{\alpha_i} J_k^P))$ setting $p$ is given, which is a one-dimensional array segmentation problem. A well-performed method to solve the one-dimensional array segmentation problem is the dichotomy method, while it does not apply to multi-dimensional array segmentation. To solve the problem $\omega_1$ in real scenarios which requires considering $J^P$, $J^M$, $H^P$, and $H^M$ simultaneously, we introduce multiple weight vectors and propose improved multi-dimensional dichotomy method (IMD), whose pseudo-code can be seen in Algorithm 2 and algorithm framework is shown in Fig. 5.

Following the strategy of typical dichotomy, IMD converts the problem $\omega_1$ to the ordered bin-packing problem and transforms the solution target to find the minimum volume of bins. Line 1 to Line 3 in Algorithm 2 are respectively setting convergence standard (i.e., $\epsilon$), the initial range of packing volume ($D^{\min}$ and $D^{\max}$), and the weight groups ($W$). The operations in Line 2 can ensure that the initial range must contain the optimal packing volume. The loop from Line 4 to Line 15 is the search process of the multi-dimensional dichotomy. As Line 5, IMD utilizes the mean value ($D^{\text{mean}}$) of the maximum value ($D^{\max}$) and the minimum value ($D^{\min}$) as the preset packing volume. The volume is measured by $\rho_1 + \rho_2$. Because the proportion between $\rho_1$ (corresponding to the longest FP pipeline stage) and $\rho_2$ (corresponding to the longest BP pipeline stage) varies with pipeline network divisions, IMD introduces multi-weight vectors to distribute the preset volume ($D^{\text{mean}}$) according to different proportions (as Line 7) to check: whether there is a packing scheme satisfying the current preset packing volume after proportional distribution (i.e., $v_1$ and $v_2$) so that the total number of packing bins ($\tau$) is not greater than the number of pipeline stages ($N$). The loop from Line 6 to Line 11 is to traverse all vectors in the weight groups. Shown as Lines 12 to 15, IMD updates the search range by adjusting the upper and lower boundaries. If there is a weight vector satisfying the number of bins is not greater than the number of pipeline stages (i.e., $\tau \le N$), the preset volume can be further reduced (i.e., the value $\rho_1 + \rho_2$ corresponding to the theoretical optimal solution of $\lambda$ is less than $D^{\text{mean}}$), otherwise, to enlarge the preset volume.

**Algorithm 3:** Multi-Dimensional Array Segment Packing Algorithm (MASP).

```
Input  : $K$, $N$, $p$, upper limit $v_1$ and $v_2$, the functions
         of $H_i^P$, $H_i^M$, $J_i^P$, and $J_i^M$ for $\forall 1 \le i \le K$
Output: The number of bins $\tau$, the solution of array
         segmentation $\lambda = \langle \alpha_1, \alpha_2, \ldots, \alpha_\tau \rangle$
1 Set $\psi_1 = \psi_2 = 0$, $\lambda = \emptyset$, $\tau = 0$, $i = 0$
2 while $i \le K$ do
3     $i++$, $\psi_1 += H_i^P$, $\psi_2 += J_i^P$
4     if $\max(\psi_1, H_i^M) \le v_1 \wedge \max(\psi_2, J_i^M) \le v_2$ then
5         $\alpha = i$
6     if $\max(\psi_1, H_i^M) > v_1 \wedge \max(\psi_2, J_i^M) > v_2$ then
7         $\tau++$, $\lambda += \{\alpha\}$, $i = \alpha$, $\psi_1 = \psi_2 = 0$
8 $\tau++$, $\lambda += \{K\}$
```

As the IMD transforms the problem $\omega_1$ to an array segmentation packing problem, it needs to call an improved multi-dimensional array segmentation packing algorithm (MASP), shown as Algorithm 3. When $v_1$ and $v_2$ are given, the theoretical optimal solution of the problem of partitioning ordered arrays can be obtained through the local greedy algorithm.

*2) Analysis to Convergent Solutions of IMD:* The improved dichotomy to solve multi-dimensional array segmentation (IMD, Algorithm 2) aims at solving the problem that

$$\min \omega^{(1)}(\lambda) = \rho_1(\lambda, p) + \rho_2(\lambda, p) \tag{9}$$

Denoting the convergent solution of Algorithm 2 as $\lambda^{ID}$, the feasible solution set of the problem as $\Lambda$, and the theoretical optimal solution is $\lambda^O$, then the solution has the following property according to the process of Algorithm 2.

*Property 1:* If $\epsilon \to 0$ and the weight groups $W$ has adequate weights $\eta \to +\infty$ which can ergodic all possible proportions between $\rho_1$ and $\rho_2$, then for $\forall \omega^{(1)} > 0$ the following formula must be tenable.

$$\begin{cases} \tau(\omega) \le N, \text{ if } \omega \ge \omega^{(1)}\left(\lambda^{(ID)}\right) \\ \tau(\omega) > N, \text{ if } \omega < \omega^{(1)}\left(\lambda^{(ID)}\right) \end{cases} \tag{10}$$

where $\tau(\omega)$ is the minimum required number of bins when setting the preset volume of bins as $\omega$.

The proof of Property 1 can be seen as follows considering $\tau(\omega)$ is a non-increasing function.

*Proof 1:* Because $\tau(\omega^{(1)}(\lambda^{(ID)})) = N$, therefore $\tau(\omega) \le N$ when $\omega \ge \omega^{(1)}(\lambda^{(ID)})$. If $\exists \omega < \omega^{(1)}(\lambda^{(ID)})$ $\mathbf{s.t.}$ $\tau(\omega) \le N$, then the value $D^{\min} < D^{\text{mean}}$. Then, the Algorithm 2 needs to be continued, which is in contradiction with $\lambda^{(ID)}$ is a convergent solution. Therefore, for $\forall \omega < \omega^{(1)}(\lambda^{(ID)})$, $\tau(\omega) > N$. Thus, Property 1 is proved.

On the basis of Property 1, we can obtain Property 2.

*Property 2:* For $\forall \lambda \in \Lambda$, $\omega^{(1)}(\lambda) \ge \omega^{(1)}(\lambda^{(ID)}) = \omega^{(1)}(\lambda^{(O)})$ under the conditions of Property 1, i.e., $\lambda^{(ID)}$ is one theoretical optimal solution of problem $\min \omega^{(1)}$.

The proof of Property 2 is as follows by contradiction.

*Proof 2:* If $\exists \lambda \in \Lambda$ $\mathbf{s.t.}$ $\omega^{(1)}(\lambda) < \omega^{(1)}(\lambda^{(ID)})$, then $\tau(\omega^{(1)}(\lambda)) > N$ according to the second formula of (10). Because $\lambda \in \Lambda$ is a feasible solution of problem $\min \omega^{(1)}$, therefore $\tau(\omega^{(1)}(\lambda)) \le N$. These two inequalities are contradictory. Thus, for $\forall \lambda \in \Lambda$, $\omega^{(1)}(\lambda) \ge \omega^{(1)}(\lambda^{(ID)})$, i.e., $\lambda^{(ID)}$ is a theoretical optimal solution.

In fact, Property 1 and Property 2 are equivalent to reveal that the convergent solution of Algorithm 2 is the theoretically optimal solution. However, there are two indispensable conditions for Property 1 and Property 2 to be tenable, i.e., $\epsilon \to 0$, and $W$ has adequate weights ($\eta \to +\infty$). In real computer programs, these two conditions are generally unable to be achieved, since the computer cannot generate infinitely small numbers. In practical implementation of Algorithm 2 to solve the problem $\min \omega^{(1)}$, the error between $\omega^{(1)}(\lambda^{(ID)})$ and $\omega^{(1)}(\lambda^{(O)})$ is related to both $\epsilon$ and $\eta$, which is revealed by Property 3 through deduction.

*Property 3:* When $\epsilon > 0$ and $\eta < +\infty$, the error $\xi$ between the convergent solution $\omega^{(1)}(\lambda^{(ID)})$ and the theoretical optimal solution $\omega^{(1)}(\lambda^{(O)})$ is:

$$0 \le \xi = \omega^{(1)}\left(\lambda^{(ID)}\right) - \omega^{(1)}\left(\lambda^{(O)}\right) \le \epsilon + \frac{\omega^{(1)}\left(\lambda^{(O)}\right)}{\eta - 1} \tag{11}$$

According to the process of Algorithm 2, we can present the proof of Property 3 as follows.

*Proof 3:* When the Algorithm 2 reaches convergence, the following relationships are tenable:

1) $\exists i$ $\mathbf{s.t.}$ $D^{\max}\frac{i}{\eta} \ge \varrho_1(\lambda^O)$ and $D^{\max}\frac{\eta-i}{\eta} \ge \varrho_2(\lambda^O)$;
2) $\forall i \in [0, \eta]$, $D^{\min}\frac{i}{\eta} \le \varrho_1(\lambda^O)$ or $D^{\min}\frac{\eta-i}{\eta} \le \varrho_2(\lambda^O)$;
3) $0 \le D^{\max} - D^{\min} \le \epsilon$.

If $0 \le \exists i \le \eta$ $\mathbf{s.t.}$ $D^{\min}\frac{i}{\eta} \le \varrho_1(\lambda^O)$ and $D^{\min}\frac{\eta-i}{\eta} \le \varrho_2(\lambda^O)$, then $D^{\min} \le \varrho_1(\lambda^O) + \varrho_2(\lambda^O) = \omega^{(1)}(\lambda^{(O)})$. Because, $\omega^{(1)}(\lambda^{(ID)}) \le D^{\max} \le D^{\min} + \epsilon$, thus $\omega^{(1)}(\lambda^{(ID)}) \le \omega^{(1)}(\lambda^{(O)}) + \epsilon$.

If for $0 \le \forall i \le \eta$, $(D^{\min}\frac{i}{\eta} \le \varrho_1(\lambda^O)) \wedge (D^{\min}\frac{\eta-i}{\eta} \le \varrho_2(\lambda^O)) = False$, then there must $0 \le \exists i < \eta$ $\mathbf{s.t.}$ $D^{\min}\frac{i}{\eta} \le \varrho_1(\lambda^O)$, $D^{\min}\frac{\eta-i}{\eta} \ge \varrho_2(\lambda^O)$, $D^{\min}\frac{i+1}{\eta} \ge \varrho_1(\lambda^O)$ and $D^{\min}\frac{\eta-i-1}{\eta} \le \varrho_2(\lambda^O)$. Therefore, $D^{\min}\frac{\eta-1}{\eta} \le \varrho_1(\lambda^O) + \varrho_2(\lambda^O)$. Thus, $\omega^{(1)}(\lambda^{(ID)}) \le \frac{\eta}{\eta-1}\omega^{(1)}(\lambda^{(O)}) + \epsilon$.

Thus, Property 3 is proved.

Property 3 reveals the theoretical optimality of IMD. From (11) of Property 3, we can also obtain that:

$$\lim_{\epsilon \to 0, \eta \to +\infty} \left(\omega^{(1)}\left(\lambda^{(ID)}\right) - \omega^{(1)}\left(\lambda^{(O)}\right)\right) = 0 \tag{12}$$

*3) Analysis of Computational Complexity and Selection of Parameters:* To further discuss the performance of Algorithm 2, we analyze its computational complexity $\mathfrak{C}_{IMD}$.

Since the dichotomy will reduce the search space by half each time, it needs $\log_2(\frac{D}{\epsilon})$ times to reach convergence where $D = D_1 + D_2 - \max(\max(H^P, J^P), \min(H^M, J^M))$. The complexity of each time is determined by $\eta$ and Algorithm 3. It can be obtained as $\mathbf{O}(\eta(2K))$, where the complexity of Algorithm 3 is $\mathbf{O}(2K)$. Thus, the complexity of Algorithm 2 can be derived as

$$\mathfrak{C}_{IMD} = \mathbf{O}\left(2K\eta \log_2\left(\frac{D}{\epsilon}\right)\right) \tag{13}$$

If the maximum allowable error is given as $\xi$ where $\xi = \epsilon + \frac{1}{\eta-1}\omega^{(1)}(\lambda^{(O)})$ according to (11) of Property 3, we can obtain the complexity of Algorithm 2 with respect to $\epsilon$ as:

$$\mathfrak{C}_{IMD} = \mathbf{O}\left(2K\frac{\omega^{(1)}\left(\lambda^{(O)}\right)}{\xi-\epsilon}\log_2\left(\frac{D}{\epsilon}\right)\right) \tag{14}$$

Minimizing $\mathfrak{C}_{IMD}$ is equivalent to minimizing $\frac{1}{\xi-\epsilon}\ln(\frac{D}{\epsilon})$ where $\xi$ and $D$ are given. It can be derived that when

$$\epsilon\left(\ln D - \ln\epsilon + 1\right) = \xi, \tag{15}$$

$\mathfrak{C}_{IMD}$ achieves the minimum. Equation (15) presents a way to select the appropriate parameters of $\epsilon$ and $\eta$ to reduce the computational complexity under the given maximum error $\xi$. Equation (15) can be solved through various numeric methods such as Newton iteration method and secant method.

Contemporary methods to solve the pipeline networks division mainly include dynamic programming and recursive algorithm [15], [22], [24], whose computational complexities are both $\mathbf{O}(2NK^2)$. One advantage of IMD is that its computational complexity is not affected by the number of stages (i.e., $N$), and the other is that it has a linear relationship with the number of networks (i.e., $K$). Thus, IMD is significantly faster than the state-of-the-art methods.

### C. Optimal Data Partition Algorithm (ODPA)

To obtain the optimal solution of parallel training, we also need to find the optimal partition number $p$ of data parallelism (i.e., problem $\omega_2$) under a given network division scheme $\lambda$.

When, $H^P$, $H^M$, $J^P$ and $J^M$ are known, it is easy to calculate the training time $T(\lambda, p)$ for $\forall p$. Then, it only needs to choose the partition number corresponding to the minimum running time. As the network division scheme $\lambda$ is given, matrix operation can be used to accelerate the calculation of training time for all possible partition numbers. The algorithm to obtain the number of the optimal partitions under given $\lambda$ is as Algorithm 4, where $X \times Y$ means matrix multiplication, $\max(X, dim=1)$ means taking the maximum value of each row in $X$, $\max(X, Y)$ means taking the maximum value of the homologous elements of two matrices, $X \cdot Y$ means multiplying the homologous elements of two matrices. In Algorithm 4, Line 1 and Line 2 are to prepare the matrix for subsequent calculation. Line 3 is to calculate the training time of each data partition schemes, where $U_1 = \langle \ldots, (p-1)\rho_1(\lambda, p), \ldots\rangle$, $U_2 = \langle \ldots, (p-1)\rho_2(\lambda, p), \ldots\rangle$, and $Z = \langle \ldots, T(\lambda, p), \ldots\rangle$.

With cross-search for joint optimization problem, improved multi-dimensional dichotomy method (IMD) for the subproblem $\omega^{(1)}$ and optimal data partition algorithm (ODPA) for the subproblem $\omega^{(2)}$, we can obtain the systematic method for solving optimal parallel schemes of MBPP, i.e., cross-search based on IMD and ODPA (CSIMD) whose framework is shown as Fig. 5. Subsequently, we will evaluate the performance in the real GPU cluster.

**Algorithm 4:** Optimal Data Partition Algorithm via Basic Time Function (ODPA).

```
Input: $K$, $N$, $\lambda$, the functions of $H_i^P$, $H_i^M$, $J_i^P$, and $J_i^M$
      for $\forall 1 \le i \le K$
Output: The optimal partition number $p$
1 Get four matrices $Q^{(1)}$, $Q^{(2)}$, $Q^{(3)}$ and $Q^{(4)}$ to
  respectively represent $H_i^P$, $H_i^M$, $J_i^P$, and $J_i^M$. For
  example, the element $Q_{ji}^{(1)}$ in $Q^{(1)}$ equals to $H_i^P(j)$
2 Get two $K \times N$ matrices ($P^{(1)}$ and $P^{(2)}$) and a
  one-dimensional matrix $P^{(3)}$ where
  $$\begin{cases} P_{ij}^{(1)} = \begin{cases} 1, & \text{if } \alpha_{i-1}+1 \le j \le \alpha_i \\ 0, & \text{others} \end{cases}, \\ P_{ij}^{(2)} = \begin{cases} 1, & \text{if } j = \alpha_i \\ 0, & \text{others} \end{cases}, \quad P_i^{(3)} = i-1 \end{cases}$$
3 Calculate $Z$ (the array composed of respective training
  time under each data partition schemes):
  $$\begin{cases} U_1 = \max\begin{pmatrix} \max\left(Q^{(1)} \times P^{(1)}, dim=1\right) \\ \max\left(Q^{(2)} \times P^{(2)}, dim=1\right) \end{pmatrix} \cdot P^{(3)} \\ U_2 = \max\begin{pmatrix} \max\left(Q^{(3)} \times P^{(1)}, dim=1\right) \\ \max\left(Q^{(4)} \times P^{(2)}, dim=1\right) \end{pmatrix} \cdot P^{(3)} \\ Z = \text{sum}\left(\sum_{i=1}^{4}\left(Q^{(i)} \times P^{(2-(i \bmod 2))}\right), dim=1\right) \\ \quad +U_1 + U_2 \end{cases}$$
4 Obtain the index corresponding to the minimum value of
  matrix $Z$ as the optimal partition number that
  $$p = \arg\min(Z)$$
```

## V. Experiment Evaluation

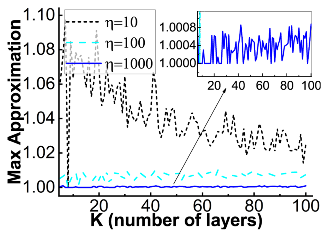
For the sake of the comprehensive evaluations of our proposed methodologies, we carry out three groups of experiments from various aspects including:

1) $EX_1$: evaluation of IMD to solve the ordered multi-dimensional array segmentation problem;
2) $EX_2$: evaluation of CSIMD in the CV-related networks with CNN layers to obtain jointly optimal schemes of network division and data partition;
3) $EX_3$: evaluation of CSIMD in the NLP-related networks (typical LLMs) with transformer layers to obtain jointly optimal schemes of network division and data partition.

In the actual parallel training, there is also a restriction that the peak memory of the workload in each GPU cannot exceed that of the corresponding GPU. Then, the experiments are launched on a real cluster with multi-servers. The configurations of the realistic cluster environment are as follows.

- Communication Network: 10 Gigabit, full duplex;
- Program version: Python 3.7 + Pytorch 1.13.1;
- Servers:
  - CPU: Intel i9 10850 K @ 3.6 GHz, 10 cores;
  - RAM: LPX 64 GB DDR4 3200;
  - GPU: NVIDIA TESLA V100 @ 32 GB;

The datasets used to execute training in experiments mainly include Mnist (with 60000 training images resized to $100 \times 100$


(a) $K \in [5, 100], N = 4$

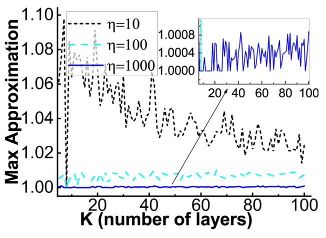
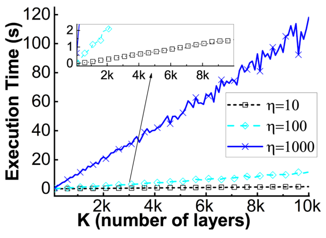

(b) $K \in [5, 100], N = 5$

Fig. 6. The maximum approximations for different numbers of weight groups when using IMD to solve multi-dimensional array segmentation where each combination of $(K, N)$ has 100 instances.


(a) $K \in [100, 10000], N = 50$

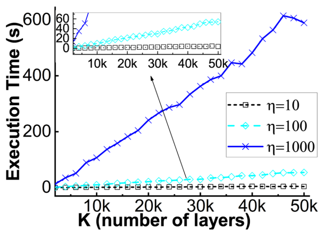
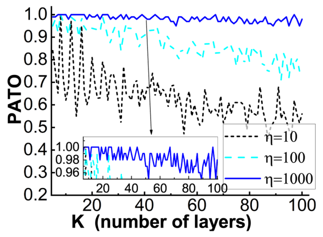

(b) $K \in [2000, 50000], N = 1000$

Fig. 8. The average execution time (computational complexity) for different sizes of weight groups when using IMD to solve multi-dimensional array segmentation where each combination of $(K, N)$ has 20 instances.


(a) $K \in [5, 100], N = 4$

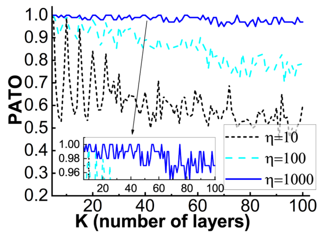

(b) $K \in [5, 100], N = 5$

Fig. 7. The probabilities achieving the theoretical optimization (PATO) for different numbers of weight groups when using IMD to solve multi-dimensional array segmentation where each combination of $(K, N)$ has 100 instances corresponding to Fig. 6.

TABLE II
THE FITTED SLOPE (FS) AND GOODNESS-OF-FIT TO LINEARLY FIT THE EXECUTION TIME OF IMD CORRESPONDING TO FIG. 8

| Scenario | $\eta$ | FS | $R^2$ |
|---|---|---|---|
| $K \in [1000, 10000], N = 50$ | 10 | 0.00015 | 0.9977 |
| | 100 | 0.00112 | 0.9857 |
| | 1000 | 0.01116 | 0.9887 |
| $K \in [2000, 50000], N = 1000$ | 10 | 0.00009 | 0.9459 |
| | 100 | 0.00112 | 0.9951 |
| | 1000 | 0.01285 | 0.9938 |

), ImageNet (with more than 14M images uniformly resized to $100 \times 100$ ) and WikiText-2 (with 2M tokens divided by the seq length). To observe the algorithm performance in the scenarios with more GPUs, we also use process concatenation to simulate the parallel operation of neural networks appropriately.

## *A. $EX_1$: Evaluating the Theoretical Performance of IMD*

To evaluate the optimality of the improved dichotomy algorithm (IMD) in solving multi-dimensional array segmentation, we carry out experiments in two groups of scenarios with the small scale that $(K \in [5, 100], N = 4)$ and $(K \in [5, 100], N = 5)$ to observe the approximation and probabilities to achieve the theoretical optimization (PATO). In each combination of $(K, N)$, we execute 100 instances by randomly generating the values of the array in a uniform distribution, i.e., $H_i^P, H_i^M, J_i^P, J_i^M \sim$ U$[50, 100]$; use an enumerative algorithm to obtain theoretical optimization solution; and record the maximum approximation and PATO of each $(K, N)$ using IMD respectively with the number of weight group as 11, 101, and 1001. Then, we plot the approximation in Fig. 6 and the corresponding PATO in Fig. 7.

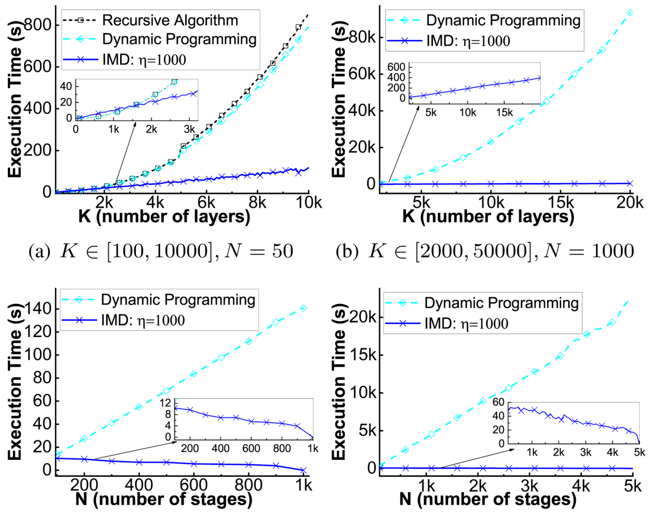
From the results of Fig. 6, the maximum approximations of $\eta = 1000$ (within the range of [1,1.001]) are lowest, followed by that of $\eta = 100$ (within [1,1.01]) and $\eta = 10$ (within [1,1.1]). As $\eta$ increases, the maximum approximation decreases, indicating that the solution of the algorithm is closer to the theoretically optimal solution. The fluctuation range of the maximum approximations is approximately inversely proportional to $\eta$. Additionally, as $K$ increases, the maximum approximation ratio does not show a significant upward trend, which indicates that the approximation (or relative error) is mainly influenced by $\eta$ and has no explicit positive or negative correlation with $(K, N)$. These observations are consistent with the theoretical conclusion of Property 3, which can serve as supplementary proof of theory verifying the theoretical error of the IMD algorithm proposed in this paper.

From Fig. 7, the probability of the algorithm reaching the theoretical optimal solution increases with the $\eta$. The PATO is within the range of (0.45,1]when $\eta = 10$, (0.70,1] when $\eta = 100$, and (0.95,1] when $\eta = 1000$. This is also consistent with the conclusion of Property 3.

As concluded by Property 3, Figs. 6 and 7, when $\eta$ approaches infinity, the approximation and PATO will both tend to 1. However, the actual selection of $\eta$ needs to be based on the requirements of computational complexity and optimality. To further observe the time complexity required for IMD to achieve the convergent solution, we execute experiments in two groups of scenarios with the huge scale that $(K \in [100, 10000], N = 50)$ and $(K \in [2000, 50000], N = 1000)$. Each combination of $(K, N)$ has 20 instances. Fig. 8 plots the average execution time of IMD.

As shown in Fig. 8, the average execution time of each $\eta$ increases approximately linearly with $K$. As $\eta$ increases, the gradient of the curve significantly increases. To further observe the relationship between slope and $\eta$, we linearly fit the curves in Fig. 8, and then obtained their fitted slope and goodness-of-fit (R-square) as shown in Table II.

The R-squares of all rows in Table II are larger than 0.9, indicating that the execution time can be statistically acceptable as proportional to $K$. The fitted slope in Table II is approximately proportional to $\eta$, which equals to that the execution time is proportional to $\eta$. These conclusions on computational complexity



(a) $K \in [100, 10000], N = 50$ (b) $K \in [2000, 50000], N = 1000$

(c) $K = 1000, N \in [100, 1000]$ (d) $K = 5000, N \in [100, 5000]$

Fig. 9. The average execution time (computational complexity) of IMD and baselines (dynamic programming) to solve multi-dimensional array segmentation where each combination of $(K, N)$ has 20 instances.

are consistent with (14) in Section IV-B3, which means that in the statistical sense, IMD is a linear time algorithm with controllable errors in solving the multi-dimensional array segmentation.

To demonstrate the rapidity of IMD, we also carry out experiments to compare it with contemporary methods including dynamic programming and recursive algorithm. Based on the simulation dataset, incremental experiments, for $K$ and $N$ respectively, were conducted on IMD, dynamic programming and recursive algorithms. Due to that multi-layer recursions will affect the stability of the program and system, some experiments in ultra-large scale scenarios only present the results of IMD and dynamic programming, without losing representativeness. While, in large-scale scenarios, it can be seen that the computational complexity of recursive algorithms is similar to that of dynamic programming, i.e., both satisfying $\mathrm{O}(2NK^2)$. In experiments, each combination of $(K, N)$ contains 20 instances. IMD method chooses 1000 weights (i.e., $\eta = 1000$). Then, the results of the average execution time of these three algorithms to solve the multi-dimensional array segmentation (problem $\omega^{(1)}$) are plotted in Fig. 9.

From Fig. 9, the computational complexity of the state-of-the-art algorithms (dynamic programming and recursive algorithm) is slightly lower than that of IMD for small-scale cluster, while IMD has a significantly lower computational complexity with increasing scale, which may be because small-scale corresponds to a search space that can be traversed in a short period of time, in which the advantages of IMD (using $\eta = 1000$) have not yet been manifested. IMD is an algorithm for the linear polynomial computational complexity of $K$ (the number of network layers); dynamic programming and recursive algorithm are algorithms for the quadratic polynomial computational complexity of $K$. For $N$ (the number of stages), dynamic programming and recursive algorithm are algorithms with positive linear polynomial computational complexity; while the complexity of IMD decreases with increasing $N$. This is because, in IMD, $N$ mainly affects the calculation process of the total occupancy in each bin after transforming multi-dimensional segmentation problems into orderly packing. As $N$ increases, the average number of layers in each stage decreases, resulting in a decrease in overall computational complexity. Thus, the computational complexity of IMD can be further updated as:

$$\mathfrak{C}_{IMD} = \mathbf{O}\left(2\left(K - \phi(N)\right)\eta \log_2\left(\frac{D}{\epsilon}\right)\right) \tag{16}$$

where $\phi(N)$ is an increasing positive function related to $N$, satisfying $\phi(N) \leq K$ and $\phi(K) = K$.

The experiments on approximation, PATO, and computational complexity in this subsection not only verify the theoretical derivation of IMD algorithm performance in Section IV-B, but also again demonstrate its optimality and rapidity. We set $\eta = 1000$ considering the number of DNN's layers in the subsequent experiment is much smaller than the order of magnitude of $K$ in the experiment of this subsection. The optimization algorithm can give a parallel scheme in advance without affecting the actual process of parallel training.

TABLE III
DETAIL OF SELF-DESIGNED CNNS

| Layer | Types | Input Channel | Output Channel | Kernel |
|---|---|---|---|---|
| $L_1$ | CNN | $\mathrm{In}_1 = 1$ | $\mathrm{Out}_1 = \mathrm{U}(20, 50)$ | |
| $L_x$ | CNN | $\mathrm{In}_x = \mathrm{Out}_{x-1}$ | $\mathrm{Out}_x = \mathrm{U}(20, 50)$ | $3 \times 3$ |
| $L_K$ | FC | $\mathrm{In}_K = \mathrm{Out}_{K-1}$ | $\mathrm{Out}_K = 10$ | |

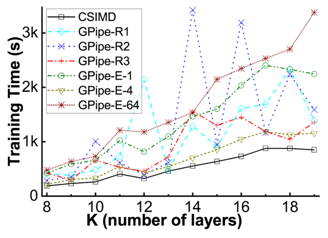
## B. $EX_2$: Evaluating CSIMD in the CV-Related Networks

To evaluate the CSIMD in solving the joint optimization problem to obtain the network division and data partition, we first carry out the experiments in CV-related networks which mainly consist of convolutional layers and fully connected layers. The compared strategies are selected as:

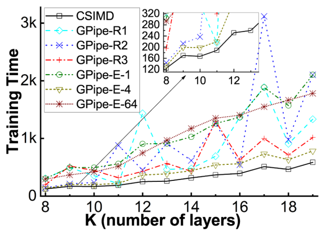
1) GPipe-R: GPipe based on random network divisions and data partitions;
2) GPipe-E: GPipe based on the evenly distributed network divisions and a fixed number of data partitions.

In self-designed CNNs, as shown in Table III, we continuously increase the number of CNN layers, where the number of output channels of each CNN layer is a random value generated by uniform distribution $\mathrm{U}(20, 50)$. Three fixed partitions of GPipe-E are set as 1, 4 and mini-batch size (BS). The mini-batch size is set as 64, the input figures are resized to $100 \times 100$. Then, we record the time to train 6400 images in Mnist dataset for one iteration respectively in each instance of two groups of scenarios that $(K \in [8, 19], N = 4)$ and $(K \in [8, 19], N = 8)$ and plot results in Fig. 10.

As shown in Fig. 10, the curve of CSIMD remains the lowest, followed by GPipe-E-4. The results of Fig. 10 demonstrate using CSIMD to search the network division and mini-batch partitions can further improve the performance of MBPP. This also indicates that setting fixed network division and mini-batch partitions cannot adapt to all scenarios. The observation that GPipe-E-4 is better than GPipe-E-1 indicates performing certain

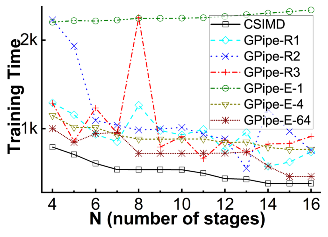


(a) $K \in [8, 19], N = 4$

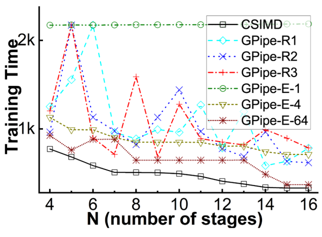


(b) $K \in [8, 19], N = 8$

Fig. 10. The training time with respect of $K$ (number of layers) for self-designed CNN in Table III to train 6400 images of Mnist under different network division and batch partition strategies where mini-batch size is 64, the resize of image is $100 \times 100$.



(a) 1 GB Bandwidth



(b) 10 GB Bandwidth

Fig. 11. The training time regarding $N$ (number of stages) for VGG16 to train 6400 images of ImageNet under different network division and batch partition strategies where mini-batch size is 64, the resize of image is $224 \times 224$.

mini-batch partitioning can improve training speed, which is consistent with the goal proposed by GPipe. However, as the number of partitions increases to 64, the training time actually becomes larger on the contrary. This indicates that there are one or more network layers whose time cost (computing time or communication time) is non-linear to the data volume (batch size), i.e., $H_i^P, H_i^M, J_i^P, J_i^M$ may not be inversely proportional to $p$.

Additionally, to verify the performance of CSIMD in parallel training of existing common CV-related neural networks, we execute experiments in VGG16 on the ImageNet dataset. We collect training performance data in two network bandwidth environments that are under the Gigabit bandwidth and 10 Gigabit bandwidth. As the layers of VGG16 are given, we select the number of stages ($N$) as abscissa. Then, we plot their training time for training 6400 images in Fig. 11.

In Fig. 11, the curve of CSIMD also remains the lowest showing a decreasing trend with the increase of $N$. The curve of GPipe-E-1 increases with $N$, which is because adding the stages actually increases additional communication time. In addition, with the change in the number of stages, the experimental results of GPipe-E have a certain fluctuation, which may be mainly because the division of network layers is not strictly balanced under different numbers of pipeline stages, and the real device operation status also has a certain fluctuation. This also reflects the necessity of optimizing the network division of pipeline stages. Comparing Fig. 11(a) to (b), as bandwidth increases, the training time of CSIMD and GPipe-E decreases, which indicates that improving communication speed can accelerate

TABLE IV
THE (MINIMUM, AVERAGE, MAXIMUM) RATIOS OF TRAINING TIME BETWEEN THE COMPARISON STRATEGIES AND CSIMD UNDER CNNS

| Compared Strategies | Self CNNs | VGG16 | Average |
|---|---|---|---|
| GPipe-R1 | $(1.04, 2.29, 6.60)$ | $(1.51, 2.01, 3.71)$ | 2.15 |
| GPipe-R2 | $(1.16, 2.24, 6.18)$ | $(1.25, 2.12, 3.19)$ | 2.18 |
| GPipe-R3 | $(1.14, 1.76, 3.53)$ | $(1.21, 2.10, 4.20)$ | 1.93 |
| GPipe-E-1 | $(1.66, 2.40, 3.69)$ | $(2.78, 4.68, 6.55)$ | 3.54 |
| GPipe-E-4 | $(1.07, 1.24, 1.48)$ | $(1.43, 1.80, 2.18)$ | 1.52 |
| GPipe-E-BS | $(2.16, 2.93, 3.97)$ | $(1.11, 1.40, 1.78)$ | 2.17 |

TABLE V
DETAILS OF EXPERIMENTS AND CONFIGURES OF LLMS

| DP Dimension | LLMs | Transformer | D_model | Heads | $(M, S)$ |
|---|---|---|---|---|---|
| Batch Size | Llama 3.2: 3B | 28 Layers | 3072 | 32 | (256, 32) |
| | Llama 2: 7B | 32 Layers | 4096 | 32 | (256, 32) |
| | GPT-2 Large: 774M | 36 Layers | 1280 | 20 | (512, 64) |
| | GPT-2 XL: 1.56B | 48 Layers | 1600 | 25 | (512, 64) |
| Token Size | Llama 2: 70B | 80 Layers | 8192 | 64 | (8, 1024) |
| | GPT-3: 175B | 96 Layers | 12288 | 96 | (4, 1024) |

parallel training. However, the acceleration effect by improving communication for GPipe-E-1 with the longest training time is the most significant, and that for CSIMD is the smallest. This is because CSIMD minimizes the impact of communication time on the overall training speed by optimizing network division and mini-batch partition, which once again validates the advantage of cross-search when considering both nonlinear computation time and communication time. To our knowledge, there is currently no algorithm that can achieve this.

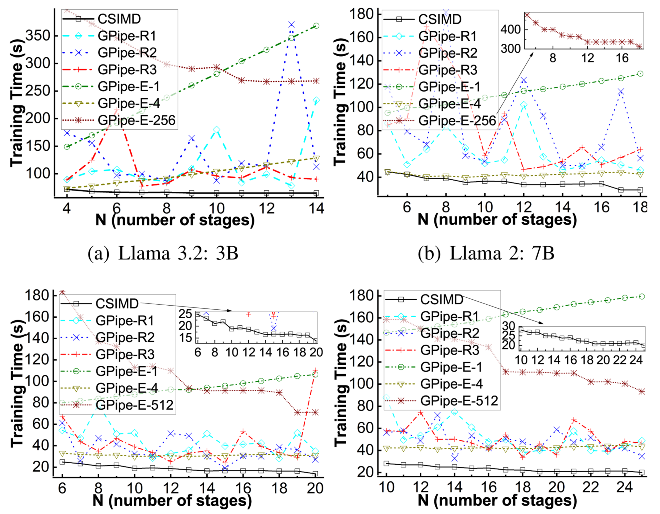
To quantitatively evaluate the improvement effect of CSIMD, we compile various experimental results on CNNs in this subsection and record the (minimum, average, maximum) ratios of training time between the comparison strategies and CSIMD under CNNs. Then, we list the ratios in Table IV.

As shown in Table IV, CSIMD can achieve $2.0\times$, $3.5\times$, $1.5\times$ and $2.1\times$ speeds on average respectively over GPipe-R, GPipe-E-1, GPipe-E-4 and GPipe-E-BS in CNNs of the experiments. On average, the training speeds of CSIMD in CNNs are $(2.0, 2.5)\times$ of (GPipe-R, GPipe-E).

## *C. $EX_3$: Evaluating CSIMD in the NLP-Related Networks*

To evaluate CSIMD in the NLP-related networks, we carry out experiments on GPT series and Llama series which are some representative LLMs. Different from CNNs, transformer-based NLP models can not only be partitioned in data batches but also in tokens, i.e., Terapipe [13]. Since these two aspects are actually both data parallelism obeying the same cost model of (6), our CSIMD is applicable to both of them. Therefore, in experiments of transformer-based DNNs, we present the verification for the partition of both batch size and token size, comparing the heuristic strategies under GPipe and Terapipe respectively. Table V lists the configures and details of LLMs in experiments, where D_model is embedding dimension, $M$ means mini-batch size and $S$ means seq length.

For the perspective of micro-batch-based data parallelism, we carry out experiments on real clusters, training LLMs on WikiText-2, comparing CSIMD with the heuristic strategies



(a) Llama 3.2: 3B (b) Llama 2: 7B

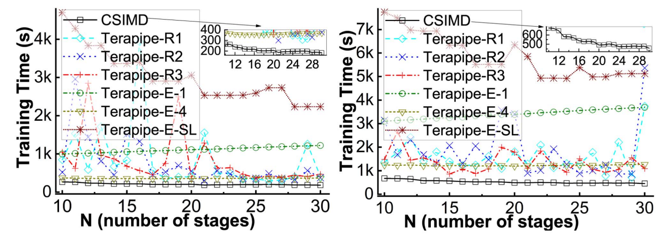
(c) GPT-2 Large: 774M (d) GPT-2 XL: 1.56B

Fig. 12. The training time with respect of number of stages for typical LLMs to train 10240 sequences of WikiText-2, comparing CSIMD with heuristic strategies from the perspective of micro-batch-based data parallelism.

under GPipe. Applying the number of stages as abscissa, we plot the time to train 10240 sequences in Fig. 12.

In Fig. 12, the curves of CSIMD are significantly lower than compared strategies, which verifies the feasibility and superiority of CSIMD in optimizing parallel training of typical LLMs. The trends of GPipe-E-1, GPipe-E-4 and GPipe-E-BS indicate there are layers with nonlinear time cost (computing time or communication time) with respect to data volume in transformer-based NLP networks, otherwise, GPipe-E-BS (GPipe-E-256 or GPipe-E-512) should be better than GPipe-E-4 and GPipe-E-1, because if following the linear assumptions of time functions, more partitions of mini-batch will cause a smaller training time usually. In addition, as the number of layers continues to increase, the advantages of CSIMD become increasingly apparent, which is because an increase in the number of network layers will bring more possibilities for network division, and CSIMD can find optimal solutions among these possibilities due to the optimality of IMD. In experiments, when $N$ is greater than a certain number, the training time achieved by CSIMD remains almost unchanged and does not decrease as $N$ increases. This is because using a certain number of GPUs in CSIMD search results is optimal, while more GPUs may actually introduce additional communication time. Therefore, when the graphics card has enough memory, the more stages of the pipeline, the shorter the training time may not be.

To further verify the practicability of CSIMD for the general MBPP parallel training architecture, we carry out experiments on real clusters to observe performance of CSIMD for micro-tokens-based data parallelism, by training LLMs on WikiText-2 (significantly, the training performance mainly corresponds to the training time of the language text under the specific sequence length. In fact, it has little relevance to the dataset itself). To adapt to micro-tokens-based data parallelism, we leverage the heuristic strategies under Terapipe as the baselines. Then, the results of



(a) Llama 2: 70B (b) GPT-3: 175B

Fig. 13. The training time with respect of number of stages for typical LLMs to train 1024 sequences (with 1024 × 1024 tokens) of WikiText-2, comparing CSIMD with heuristic strategies from the perspective of micro-tokens-based data parallelism.

TABLE VI
THE (MINIMUM, AVERAGE, MAXIMUM) RATIOS OF TRAINING TIME BETWEEN THE COMPARISON STRATEGIES AND CSIMD UNDER LLMS

| Compared Strategies | Llama Series | GPT Series | Average |
|---|---|---|---|
| MBPP-R1 | $(1.20, 2.73, 18.07)$ | $(1.48, 2.62, 8.19)$ | 2.68 |
| MBPP-R2 | $(1.31, 2.76, 11.24)$ | $(1.11, 2.66, 11.74)$ | 2.71 |
| MBPP-R3 | $(1.18, 2.80, 12.27)$ | $(1.34, 2.40, 7.98)$ | 2.60 |
| MBPP-E-1 | $(2.08, 4.42, 6.66)$ | $(3.22, 6.35, 8.96)$ | 5.38 |
| MBPP-E-4 | $(1.01, 1.53, 2.00)$ | $(1.33, 2.01, 2.78)$ | 1.77 |
| MBPP-E-BS/SL | $(4.09, 10.79, 17.73)$ | $(4.38, 7.75, 12.93)$ | 9.27 |

the time to training 1024 sequences (with 1024 × 1024 tokens) are plotted in Fig. 13 where "SL" means the fixed partitions of tokens are seq length (i.e., partitioning one sequence into 1024 micro-sequences in experiments).

Overall, CSIMD remains lowest in Fig. 13, which demonstrates the superiority of CSIMD in MBPP with micro-tokens. In fact, the partitions from tokens and mini-batches both correspond to a certain dimension of the input data matrix, not changing the formulas of the cost model. The theoretical cost model and optimization algorithm of CSIMD are targeted at the pipeline parallel training with the general data parallelism, so they are not only suitable for micro-batch-based GPipe, but also suitable for micro-tokens-based Terapipe.

Similarly, in order to quantitatively evaluate the improvement effect of CSIMD in NLP networks, we compile results in this subsection and list the (minimum, average, maximum) ratios of training time between the comparison strategies and CSIMD in Table VI. On average, the training speeds of CSIMD in transformer networks are $(2.66, 5.48)\times$ of (MBPP-R, MBPP-E) (MBPP can represent various pipeline parallel training architecture based on data time-sharing parallelism such as GPipe and Terapipe).

## VI. CONCLUSION

Distributed parallel training is now a hotspot in the development of artificial intelligence models. Micro-batch-based pipeline parallelism (MBPP, e.g., GPipe and Terapipe) is one of the popular strategies to improve the performance of parallel training. In MBPP, two key factors are the division of the network layers and the partition of mini-batches. Additionally, due to the complexity of the parallel training process, some cost models make a lot of assumptions, which makes them deviate from the real scenarios.

In view of these, we consider computing time and communication time simultaneously and consider them nonlinear to data size. Then focusing on the scenario where the number of network layers is greater than the number of devices, we derive a time-cost model with respect of network division and data partitions. Based on the time cost model, we formulate the problem of solving network division and data partitions as a joint optimization problem, where solving network division can be regarded as an ordered multi-dimensional array segmentation problem. To solve the joint optimization problem, we propose CSIMD, in which we propose an improved multi-dimensional dichotomy (IMD) to solve multi-dimensional array segmentation. Through theoretical derivation, we present and prove the theoretical performance of IMD, indicating that IMD can approximately achieve theoretical optimum with linear calculational complexity. Finally, experiments have verified the theoretical optimality and linear computational complexity $\mathbf{O}(2K\eta\log_2(\frac{D}{\epsilon}))$ of IMD, far faster that the state-of-the-art methods, i.e., dynamic programming and recursive algorithm with quadratic polynomials calculational complexity as $\mathbf{O}(2NK^2)$. We also carry out extensive experiments in CNN-based networks and transformer-based networks. Experimental results demonstrate our proposed CSIMD can obtain optimal network division and data partition schemes under MBPP. On average, training speeds of CSIMD in CNN- and transformer-related deep neural networks are respectively $(2.0, 2.5)\times$ and $(2.66, 5.48)\times$ of (MBPP-R, MBPP-E).

This paper not only provides an algorithm to obtain optimal parallel schemes but also provides a complete idea of solving high-performance parallel training schemes from the perspective of solving joint optimization problems, which can also be applied to scenarios adding tensor model parallelism. In the future, we consider combining 3D or 4D parallelism to explore time-cost models and optimization algorithms for more complex scenarios.

## References

[1] H. Wang et al., “A comprehensive survey on training acceleration for large machine learning models in IoT,” *IEEE Internet Things J.*, vol. 9, no. 2, pp. 939–963, Jan. 2022.

[2] J. Wei et al., “Finetuned language models are zero-shot learners,” in *Proc. 10th Int. Conf. Learn. Representations*, 2022, pp. 1–46.

[3] T. B. Brown et al., “Language models are few-shot learners,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, Art. no. 159.

[4] J. W. Rae et al., “Scaling language models: Methods, analysis & insights from training gopher,” 2021, arXiv: *2112.11446*.

[5] H. Touvron et al., “Llama 2: Open foundation and fine-tuned chat models,” 2023, arXiv: *2307.09288*.

[6] A. Dubey et al., “The Llama 3 herd of models,” 2024, arXiv: *2407.21783*.

[7] Z. Li et al., “Optimizing makespan and resource utilization for multi-DNN training in GPU cluster,” *Future Gener. Comput. Syst.*, vol. 125, pp. 206–220, 2021.

[8] P. Sun, Y. Wen, R. Han, W. Feng, and S. Yan, “GradientFlow: Optimizing network performance for large-scale distributed DNN training,” *IEEE Trans. Big Data*, vol. 8, no. 2, pp. 495–507, Apr. 2022.

[9] H. Fu et al., “HGP4CNN: An efficient parallelization framework for training convolutional neural networks on modern GPUs,” *J. Supercomputing*, vol. 77, no. 11, pp. 12741–12770, 2021.

[10] J. Xu et al., “Effective scheduler for distributed DNN training based on MapReduce and GPU cluster,” *J. Grid Comput.*, vol. 19, no. 1, 2021, Art. no. 8.

[11] D. Li et al., “A memory-efficient hybrid parallel framework for deep neural network training,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 35, no. 4, pp. 577–591, Apr. 2024.

[12] J. Romero et al., “Accelerating collective communication in data parallel training across deep learning frameworks,” in *Proc. 19th USENIX Symp. Netw. Syst. Des. Implementation*, 2022, pp. 1027–1040.

[13] Z. Li et al., “TeraPipe: Token-level pipeline parallelism for training large-scale language models,” in *Proc. 38th Int. Conf. Mach. Learn.*, 2021, pp. 6543–6552.

[14] Y. Huang et al., “GPipe: Efficient training of giant neural networks using pipeline parallelism,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 103–112.

[15] S. Fan et al., “DAPPLE: A pipelined data parallel approach for training large models,” in *Proc. 26th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, 2021, pp. 431–445.

[16] Z. Zeng, C. Liu, Z. Tang, W. Chang, and K. Li, “Training acceleration for deep neural networks: A hybrid parallelization strategy,” in *Proc. 58th ACM/IEEE Des. Automat. Conf.*, 2021, pp. 1165–1170.

[17] N. Dryden et al., “Channel and filter parallelism for large-scale CNN training,” in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2019, pp. 10:1–10:20.

[18] J. He et al., “FasterMoE: Modeling and optimizing training of large-scale dynamic pre-trained models,” in *Proc. 27th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, 2022, pp. 120–134.

[19] A. Jain et al., “GEMS: GPU-enabled memory-aware model-parallelism system for distributed DNN training,” in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2020, Art. no. 45.

[20] L. Zheng et al., “Alpa: Automating inter- and intra-operator parallelism for distributed deep learning,” in *Proc. 16th USENIX Symp. Operating Syst. Des. Implementation*, 2022, pp. 559–578.

[21] Y. Bai et al., “Gradient compression supercharged high-performance data parallel DNN training,” in *Proc. ACM SIGOPS 28th Symp. Operating Syst. Princ.*, 2021, pp. 359–375.

[22] V. Elango, “Pase: Parallelization strategies for efficient DNN training,” in *Proc. 35th IEEE Int. Parallel Distrib. Process. Symp.*, 2021, pp. 1025–1034.

[23] J. Dong et al., “EFLOPS: Algorithm and system co-design for a high performance distributed training platform,” in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2020, pp. 610–622.

[24] S. Ouyang, D. Dong, Y. Xu, and L. Xiao, “Communication optimization strategies for distributed deep neural network training: A survey,” *J. Parallel Distrib. Comput.*, vol. 149, pp. 52–65, 2021.

[25] S. Zhao et al., “vPipe: A virtualized acceleration system for achieving efficient and scalable pipeline parallel DNN training,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 3, pp. 489–506, Mar. 2022.

[26] Z. Han, G. Qu, B. Liu, and F. Zhang, “Exploit the data level parallelism and schedule dependent tasks on the multi-core processors,” *Inf. Sci.*, vol. 585, pp. 382–394, 2022.

[27] Y. Li, Z. Zeng, J. Li, B. Yan, Y. Zhao, and J. Zhang, “Distributed model training based on data parallelism in edge computing-enabled elastic optical networks,” *IEEE Commun. Lett.*, vol. 25, no. 4, pp. 1241–1244, Apr. 2021.

[28] O. Beaumont, L. Eyraud-Dubois, and A. Shilova, “MadPipe: Memory aware dynamic programming algorithm for pipelined model parallelism,” in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2022, pp. 1063–1073.

[29] D. Narayanan et al., “PipeDream: Generalized pipeline parallelism for DNN training,” in *Proc. 27th ACM Symp. Operating Syst. Princ.*, 2019, pp. 1–15.

[30] S. Zhao et al., “NASPipe: High performance and reproducible pipeline parallel supernet training via causal synchronous parallelism,” in *Proc. 27th ACM Int. Conf. Architect. Support Program. Lang. Operating Syst.*, 2022, pp. 374–387.

[31] D. Narayanan et al., “Memory-efficient pipeline-parallel DNN training,” in *Proc. 38th Int. Conf. Mach. Learn.*, 2021, pp. 7937–7947.

[32] J. Li, Y. Wang, J. Zhang, J. Jin, F. Dong, and L. Qian, “PipePar: A pipelined hybrid parallel approach for accelerating distributed DNN training,” in *Proc. 24th IEEE Int. Conf. Comput. Supported Cooperative Work Des.*, 2021, pp. 470–475.

[33] D. Narayanan et al., “Efficient large-scale language model training on GPU clusters using Megatron-LM,” in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2021, pp. 58:1–58:15.

[34] Z. Zhang, J. Chen, and B. Hu, “The optimization of model parallelization strategies for multi-GPU training,” in *Proc. IEEE Glob. Commun. Conf.*, 2021, pp. 1–6.

[35] Y. Lee, J. Chung, and M. Rhu, "SmartSAGE: Training large-scale graph neural networks using in-storage processing architectures," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, 2022, pp. 932–945.
[36] G. Zhou, W. Tian, R. Buyya, and K. Wu, "UMPIPE: Unequal microbatches-based pipeline parallelism for deep neural network training," *IEEE Trans. Parallel Distrib. Syst.*, vol. 36, no. 2, pp. 293–307, Feb. 2025.
[37] R. Gu et al., "Liquid: Intelligent resource estimation and network-efficient scheduling for deep learning jobs on distributed GPU clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2808–2820, Nov. 2022.
[38] V. Md et al., "DistGNN: Scalable distributed training for large-scale graph neural networks," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2021, pp. 76:1–76:14.

**Guangyao Zhou** received the bachelor's and master's degrees from Tianjin University, China, and the PhD degree from the University of Electronic Science and Technology of China, China. He received the excellent doctoral dissertation award of the University of Electronic Science and Technology of China. He is now an assistant professor with the School of Computing and Artificial Intelligence, Southwest Jiaotong University, China. His research interests include scheduling algorithms in cloud computing or edge computing, image recognition especially facial expression recognition, parallel training and reasoning of large-scale model and evolution algorithms.

**Yiqin Fu** received the bachelor's degree from Tianjin University, China, the master's degrees from the Chinese Academy of Sciences, Beijing, China, and the doctor degree from Tianjin University, China. He is now a senior engineer of China National Petroleum Corporation. His research interests include analysis of complex dynamical systems, optimization algorithm of artificial intelligence, and industrial large-scale models.

**Haocheng Lan** received the bachelor's and master's degrees from the School of Information and Software Engineering, University of Electronic Science and Technology of China. His main research interests include NLP, machine learning, evolution algorithms, and BigData processing.

**Yuanlun Xie** received the PhD degree from the School of Information and Software Engineering, University of Electronic Science and Technology of China. His main research interests include algorithmic theory of machine learning, facial expression recognition by deep learning, evolution algorithms, attention mechanism, and BigData processing.

**Wenhong Tian** (Member, IEEE) received the PhD degree from the Department of Computer Science, North Carolina State University, Raleigh, NC, USA. He is now a professor with the University of Electronic Science and Technology of China (UESTC). His research interests include scheduling in cloud computing and BigData platforms, image recognition by deep learning, algorithmic theory of machine learning, parallel training of large-scale model.

**Rajkumar Buyya** (Fellow, IEEE) is a Redmond Barry distinguished professor and director with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in cloud computing. He served as a future fellow of the Australian Research Council during 2012-2016. He has authored more than 750 publications and seven text books. He is one of the highly cited authors in computer science and software engineering worldwide (h-index=172, gindex=384, 159800+ citations). He is recognized as a "Web of Science Highly Cited Researcher" for six consecutive years since 2016, and Scopus researcher of the Year 2017 with Excellence in Innovative Research Award by Elsevier for his outstanding contributions to cloud computing and distributed systems.

**Jianhong Qian** received the PhD degree from Zhejiang University, China. He is now working with Huawei Technologies Co. Ltd, China, to develop Huawei AI framework MindSpore. His main research interests include artificial intelligence frameworks and large-scale parallel architecture.

**Teng Su** received the PhD degree from Zhejiang University, China. He is currently an architect with Huawei AI framework MindSpore. He has lead the development of Huawei task based distributed computing framework, resource management framework.