

# Poplar: Efficient Scaling of Distributed DNN Training on Heterogeneous GPU Clusters

WenZheng Zhang<sup>1</sup>, Yang Hu<sup>2\*</sup>, Jing Shi<sup>1</sup>, Xiaoying Bai<sup>3\*</sup>

<sup>1</sup>School of Computer Science, Peking University

<sup>2</sup>Center for Information Research, Academy of Military Sciences

<sup>3</sup>Advanced Institute of Big Data

zwz@stu.pku.edu.cn, huyang.oceans@gmail.com, jingshi@stu.pku.edu.cn, baixy@aibd.ac.cn

## Abstract

Scaling Deep Neural Networks (DNNs) requires significant computational resources in terms of GPU quantity and compute capacity. In practice, there usually exists a large number of heterogeneous GPU devices due to the rapid release cycle of GPU products. It is highly needed to efficiently and economically harness the power of heterogeneous GPUs, so that it can meet the requirements of DNN research and development. The paper introduces Poplar, a distributed training system that extends Zero Redundancy Optimizer (ZeRO) with heterogeneous-aware capabilities. We explore a broader spectrum of GPU heterogeneity, including compute capability, memory capacity, quantity and a combination of them. In order to achieve high computational efficiency across all heterogeneous conditions, Poplar conducts fine-grained measurements of GPUs in each ZeRO stage. We propose a novel batch allocation method and a search algorithm to optimize the utilization of heterogeneous GPU clusters. Furthermore, Poplar implements fully automated parallelism, eliminating the need for deploying heterogeneous hardware and finding suitable batch size. Extensive experiments on three heterogeneous clusters, comprising six different types of GPUs, demonstrate that Poplar achieves a training throughput improvement of 1.02 ~ 3.92x over current state-of-the-art heterogeneous training systems.

## Introduction

With the increase in model parameters, the memory and compute requirements for model training have grown significantly beyond the capability of a single accelerator (Wang et al. 2023). Training large-scale models requires the efficient utilization of aggregated computing power and memory across hundreds or even thousands of GPUs (Smith et al. 2022). There are two distributed training frameworks to this, namely 3D Parallelism (Shoeybi et al. 2019; Lai et al. 2023) and Zero Redundancy Optimizer (ZeRO) (Rajbhandari et al. 2020). 3D Parallelism requires a complex and meticulous design by domain experts (Zheng et al. 2022). Conversely, ZeRO needs no code refactoring, which has contributed to its widespread adoption in model training (Smith et al. 2022).

Acquiring a large number of homogeneous high-end GPUs within a cluster often requires a long queuing time,

\*Corresponding authors

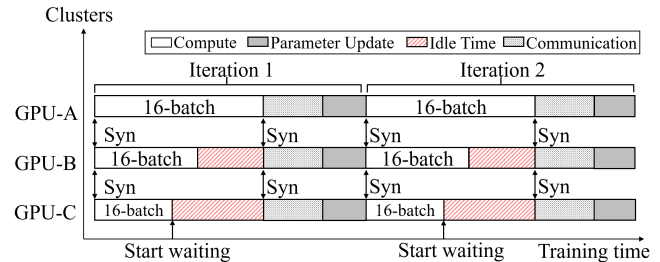


Figure 1: High-end GPUs complete their tasks first and then start waiting before synchronization. Without load balancing, there will be significant idle time.

while a sufficient quantity of heterogeneous GPUs is more readily available (Weng et al. 2022). Some researchers have access only to a variety of consumer-grade GPUs that cannot individually support the training of large models (Song et al. 2023). Furthermore, the cost of purchasing new machines is substantial due to the rapid release cycle of GPU products (Choquette 2022). In such situations, effectively utilizing heterogeneous GPU resources can help address these challenges, thereby speed up the model exploration and experiments. Current techniques and systems for distributed model training mostly assume that the workers are homogeneous (Shoeybi et al. 2019; Rasley et al. 2020). As Figure 1 showed, directly applying these techniques in heterogeneous clusters will result in substantial idle time during synchronization (Kwon et al. 2020), since powerful GPUs must wait for less ones. Achieving load balancing to minimize the idle time of each GPU in heterogeneous clusters is challenging (Tang et al. 2021), since both the compute capability and the memory capacity of GPUs need to be considered.

Many studies have focused on this issue, AMP (Li et al. 2022) and Whale (Jia et al. 2022) incorporate the elements of heterogeneity into search space of auto-parallel algorithms. AMP is equipped with an expert-design cost model that considers cluster and model configurations. Whale uses automatic graph optimizations to adapt heterogeneous GPUs (Jia, Zaharia, and Aiken 2019). Whale and AMP can enhance the utilization of heterogeneous clusters based on 3D parallelism.

However, previous research only addresses limited as-

pects of heterogeneity. Existing methods struggle in two scenarios which restrict the effective utilization of actual GPU clusters: (1) where only memory capacities differ while compute capability remain equal (e.g., A100-80GB and A100-40GB), and (2) where the number of heterogeneous GPUs is non-uniform (e.g., two V100 and one A100). Previous works have primarily relied on automatic parallelism, defining a search space and searching the parallelism strategy. If they need to accommodate scenarios with varying resource quantities, the search space would expand significantly, resulting in prohibitively high time costs. Furthermore, these scenarios are more frequently encountered in small-scale clusters, which might not be taken into consideration adequately. Besides, Whale uses single-precision floating point operations per second (FLOPs) as the only metric to measure the compute capability of GPUs. FLOPs cannot accurately reflect performance differences during training between heterogeneous GPUs, potentially introducing a degree of error in performance evaluations. Meanwhile, manually configuring heterogeneous hardware and batch size for training job in heterogeneous GPU clusters is impractical.

To address these challenges, we propose a distributed training system, named Poplar. Firstly, we extend ZeRO to support heterogeneous GPUs, while considering a wider range of GPU heterogeneity, including compute capability, memory capacity, quantity and their combinations. We treat each GPU as an individual unit and assign tasks separately to ensure maximum global throughput. Secondly, to bridge the gap between cost model and real-world performance, we conduct a fine-grained analysis for each ZeRO stage and propose a novel method for measuring heterogeneity of GPUs. We also develop a search algorithm along with a batch allocation method to ensure load balancing. Thirdly, we enable automated determination of the optimal configuration across heterogeneous GPUs, eliminating the need for expert experience and manual adjustments. The experiments result on three real world heterogeneous GPU clusters demonstrated that Poplar can speed up the training process  $1.02 \sim 3.92x$  than other methods. We will publish all source codes of this work on Github for further research explorations. In summary, Our contributions are as follows:

- To the best of our knowledge, we are the first to investigate training LLM with ZeRO on heterogeneous GPUs. We designed and implemented a distributed DNN training system, Poplar, which extend ZeRO to harness the power of heterogeneous GPUs.
- We explore a broader spectrum of GPU heterogeneity, and develop a heterogeneity aware method along with a batch size searching algorithm for each ZeRO stage.
- Poplar enables fully automated parallel training, allowing for automatic determining the configuration in heterogeneous GPU clusters.
- We conduct extensive experiments with diverse sets of real world heterogeneous GPU clusters. The results demonstrated that Poplar can speed up the training process  $1.02 \sim 3.92x$  than other methods.

## Related Work

### Zero Redundancy Optimizer

ZeRO is a variant of data parallelism based on model parallelism, which can substantially reduce the memory required for training models. ZeRO partitions optimizer states among GPUs in ZeRO-1, distributes gradients in ZeRO-2, and places parameter shards on different GPUs in ZeRO-3.

### Training in Heterogeneous GPU Clusters

Some researches use asynchronous synchronization (Zhou et al. 2020; Li et al. 2020) to alleviate the challenges posed by heterogeneity, such as HetPipe (Park et al. 2020), SAP-SGD (Cao, Zhu, and Zhou 2021), Heter-Train (Geng et al. 2023), LB-BSP (Chen et al. 2021) and Hop (Luo et al. 2019). Nonetheless, the approach of asynchronously updating gradients lacks a rigorous mathematical proof of Convergence. Whale (Jia et al. 2022) introduced a novel hardware-aware parallel strategy to enable auto parallelism. AMP (Li et al. 2022) develop a dynamic programming approach to handle the load imbalance issue from heterogeneous models in the pipeline layer assignment problem. Meanwhile, Whale and AMP impose restrictions on the number of different GPUs, failing to consider scenarios with non-uniform numbers of heterogeneous devices. These limitations greatly constrain the application in real-world heterogeneous GPU cluster.

## Method

Training a well performed model requires a substantial amount of data. The model operates on a portion of the dataset and update parameters in each iteration. The entire training process always consists of many iterations. For example, LLaMA-2 requires 500,000 iterations to train a model with 7B parameters (Touvron et al. 2023). Within each iteration, the model’s operations are repetitive. The goal of Poplar is to minimize the time required for every single iteration, thereby reducing the overall training time and improving the hardware utilization.

### Problem Formulation

Formally, the inputs of Poplar are (i) the model, (ii) the GPU cluster, and (iii) the global batch size ( $gbs$ ). During each iteration of the training process, the model needs to process the data of size  $gbs$ . We consider each GPU in the cluster as an independent device, the cluster can be represented as  $Cluster = \{g_1, g_2, g_3, \dots, g_n\}$ , where  $n$  is the total number of GPUs. Meanwhile, the compute capability of each GPU is denoted as  $Speed = \{p_1, p_2, p_3, \dots, p_n\}$ . We need to find a configuration  $B$  such that  $Cluster$  can process  $gbs$  data as quickly as possible. The configuration  $B$  can be represented as follows:

$$B = \{b_1, b_2, b_3, \dots, b_n\}, \sum_{i=0}^n b_i = gbs$$

$$0 \leq b_i \leq mbs_i, i = 1, 2, \dots, n$$

$mbs_i$  represents the max batch size of GPU  $g_i$  can handle without causing an out-of-memory (OOM) error. Each GPU

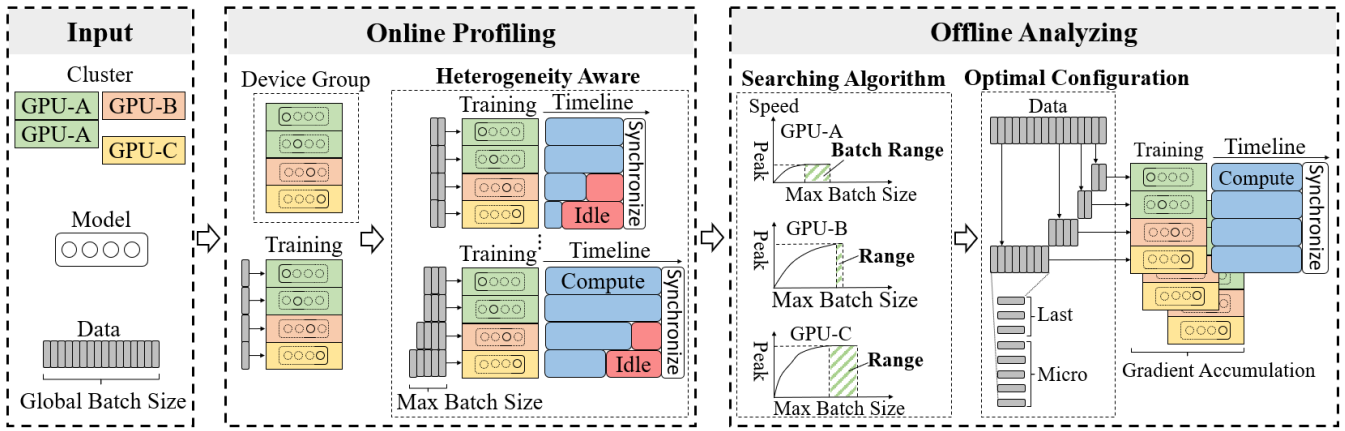


Figure 2: An overview of how Poplar automatically determines the optimal configuration.

has a different runtime  $t_i$  during a single iteration. Synchronization is required among heterogeneous GPUs in each iteration, which involves waiting until all GPUs have completed their tasks. Consequently, the iteration time  $T$  and idle time  $\delta t_i$  of each GPU can be calculated as:

$$T = \max\{t_1, t_2, t_3, \dots, t_n\} \quad (1)$$

$$\delta t_i = T - t_i, i = 1, 2, \dots, n \quad (2)$$

To ensure the shortest computing time  $T$ , our objective is to identify the configuration  $B$  that maximizes overall GPU utilization. In other words, we aim to find the configuration that minimizes overall GPU under-utilization. The under-utilization of each GPU is denoted as  $u_i$  which can be calculated as:

$$u_i = \delta t_i \times p_i, i = 1, 2, \dots, n \quad (3)$$

Powerful GPUs typically own a larger  $p$ , and therefore have a greater impact on the entire cluster. Then the objective function of Poplar will be:

$$\arg \min_{b_1, b_2, b_3, \dots, b_n} \sum_{i=1}^n \delta t_i \times p_i, i = 1, 2, \dots, n \quad (4)$$

## System Overview

Figure 2 illustrates the overall structure of Poplar. Initially, the model, GPU clusters and global batch size serve as inputs to Poplar. In the online profiling, Poplar detect the heterogeneity among all GPUs precisely, preparing the data for the subsequent offline analyzing phase. Poplar monitors metrics for each GPU during the runtime process, where the memory capacity is quantified by the maximum batch size ( $mbs$ ) of GPU, and the compute capability is reflected by the corresponding processing time. Afterwards, during the offline analyzing phase, Poplar use a batch allocation method and a search algorithm to determine the optimal allocation configuration. Based on this configuration, tasks will be assigned to each GPU to start training. The above process is a general workflow for ZeRO to support heterogeneous environment. Different ZeRO stages have distinct process in both two phase, which will be detailed in the following sections.

## Online Profiling of GPU

In this section, Poplar evaluates the compute capability and memory capacity of each GPU simultaneously. Max batch size is an important factor for GPU utilization and amount of memory used in the GPU. Large micro-batch size can improve the GPU utilization and FLOPs, but it determining the maximum batch size is not straightforward. Researchers often need to manually adjust the batch size to meet the global batch size requirement and ensure no OOM error occurs before starting training. This is already a cumbersome task for homogeneous clusters, and it becomes even more complex for heterogeneous clusters as the varying memory capacities of GPUs. Therefore, Poplar achieves fully automated determination of the maximum batch size of each GPU. Besides, Poplar obtains performance data by collecting runtime metrics during online profiling.

**Heterogeneity Aware** Accurately measuring the memory capacity of each GPU is challenging. The first challenge is how to ensure that the identified  $mbs$  does not result in out-of-memory(OOM) errors during subsequent training phases. Previous work always use a cost model to estimate the memory capacity, and reserve a large amount of memory to prevent OOM errors, which would reduce GPU utilization. This is particularly unacceptable in a heterogeneous GPU cluster since some lower-end GPUs already own very limited memory. Another challenge is the search cost, which becomes extremely high when we try every possible number of batch.

Based on these, Poplar use two steps to search  $mbs$  for each GPU, as shown in Algorithm 1. Poplar use two methods to accelerate the progress correspond to the two **for** loops in Algorithm 1. Firstly, given the linear relationship between the batch size and memory consumption during model training, Poplar initially estimate the  $mbs$  under the condition of one batch. The  $mbs$  obtained at this step represents a theoretical maximum number. The actual  $mbs$  on the GPU is typically lower than this value, necessitating a more precise determination. Poplar measures the compute performance of each GPU alone with the  $mbs$  during next step. Secondly, Poplar runs the model multiple times, gradually increasing the batch size exponentially until it reaches the  $mbs$

---

**Algorithm 1: Heterogeneity Aware of each GPU**

---

**Input:** GPU cluster:  $Cluster = \{g_1, g_2, \dots, g_n\}$ ,  $model$   
**Output:** compute capability:  $\mathbf{P} = \{p_1, p_2, \dots, p_n\}$ ,  
max batch size:  $\mathbf{Mbs} = \{mbs_1, mbs_2, \dots, mbs_n\}$

- 1: **for** each GPU  $g_i$  in  $Cluster$  **in parallel do**
- 2:    $batch\_size_i \leftarrow 1$
- 3:    $bf_i \leftarrow \text{CurrentMemoryAlloced}()$
- 4:    $model.forward(batch\_size_i)$
- 5:    $af_i \leftarrow \text{CurrentMemoryAlloced}()$
- 6:    $memory_i \leftarrow \text{TotalMemory}()$
- 7:    $mbs_i \leftarrow (memory_i - bf_i) / \frac{af_i - bf_i}{batch\_size_i}$
- 8: **end for**
- 9: **for** each GPU  $g_i$  in  $Cluster$  **in parallel do**
- 10:   **for**  $batch\_size = 1, 2, 4, 8, \dots, mbs_i$  **do**
- 11:      $model.step(batch\_size)$
- 12:      $p_i.append(\text{TimeConsumedDuringStep})$
- 13:     **if** find OOM error **then**
- 14:        $mbs_i \leftarrow batch\_size - 1$
- 15:     **end if**
- 16:   **end for**
- 17:    $low \leftarrow mbs_i / 2$
- 18:    $high \leftarrow mbs_i$
- 19:   **while**  $low \leq high$  **do**
- 20:      $mid \leftarrow \lfloor (low + high) / 2 \rfloor$
- 21:      $batch\_size_i \leftarrow mid$
- 22:      $model.step(batch\_size_i)$
- 23:      $p_i.append(\text{TimeConsumedDuringStep})$
- 24:     **if** find OOM error **then**
- 25:        $high \leftarrow mid$
- 26:     **else**
- 27:        $low \leftarrow mid$
- 28:     **end if**
- 29:   **end while**
- 30:    $mbs_i \leftarrow batch\_size_i$
- 31: **end for**

---

or encounter an OOM error. Then, Poplar employs a binary search to iteratively run the model until it finds the accurate  $mbs$  that does not cause an OOM error. At the end of this phase, Poplar will obtain the  $mbs$  for each GPU and its running time under several batch sizes. The overall time is acceptable because Poplar run only a single iteration, rather than the entire  $gbs$ . Meanwhile, starting from ZeRO-0, if Poplar find that the current stage cannot even run a single batch, it will automatically increase the ZeRO stage.

**Time Consumed Estimation** In online profiling, the overall process across different ZeRO stages is almost the same, with the only difference lying in the computation of *TimeConsumedDuringStep*, which records the GPU runtime during model execution. The most time-consuming operations are typically tensor multiplications and additions during the model computation process, but other operations cannot be neglected, such as tensor transpositions, activations, normalizations, convolution matrix inversions, eigenvalue decompositions, and memory accesses (Zhu et al. 2018). Therefore, we gather the wall time of each GPU to measuring compute

capability instead of FLOPs alone. Besides, the communication occurs in Poplar among GPUs using Collective Operations (Thakur, Rabenseifner, and Gropp 2005) such as All-Reduce and All-Gather (Patarasuk and Yuan 2009). Therefore, we only need to consider the heterogeneity of compute performance due to the consistent communication time.

In ZeRO-0, the synchronization point occurs just before the optimizer updates the parameters, following the completion of both the forward and backward passes on all GPUs. At this point, the model needs to collect and accumulate gradients across all GPUs. The optimizer update time is very short, and even equal when using parameter offloading. Thus, Poplar record the total GPU runtime for both the forward and backward in ZeRO-0, which serves as the result for *TimeConsumedDuringStep*. In ZeRO-1, the situation is identical to ZeRO-0 until the step of optimizer, where a second synchronization point is encountered. Therefore, Poplar use the same *TimeConsumedDuringStep* method as in ZeRO-0.

In ZeRO-2, the first synchronization point occurs in backward. During the entire backward process, multiple synchronizations take place, with each synchronization using Reduce-Scatter to distribute the gradients to the GPUs responsible for maintaining the corresponding weights. The large number of synchronizations make the situation complex due to the overlap between computation and communication. Therefore, Poplar uses a novel method based on the communicate time. In each iteration, all GPUs are synchronized at the beginning and end of the backward, ensuring that the backward time is consistent across GPUs. However, the synchronization time is inconsistent, with faster GPUs doing Collective Operations earlier and consequently start to wait. The idle time is included in the time of Collective Operations. Thus, Poplar obtains the computation time during the backward process for each GPU by directly subtracting the time of Collective Operations, and obtain *TimeConsumedDuringStep* by adding the time of forward.

In ZeRO-3, there are multiple synchronization points during the whole pass. An extra synchronization is performed compared to ZeRO-2. ZeRO-3 uses all-gather to collect weights before each computation. Thus, Poplar calculate the GPU runtime by subtracting three parts: (1) the time of All-Gather in the forward process, (2) the time of All-Gather and (3) Reduce-Scatter in the backward process.

## Offline Analyzing of Configuration

At this stage, Poplar first construct comprehensive performance curves which contain the computation times for each GPU across all batch sizes. Subsequently, Poplar uses a search algorithm to determine the optimal batch allocation strategy. The factors in search algorithm include compute power, memory capacity, the number of each type of GPU and network communication time. Finally, Poplar completes preparatory tasks for model initialization, such as configuring a data loader with dynamic batch sizes, and distributing model shards to each GPU. Poplar keep the global batch size consistent and the micro batch size dynamic, which has a negligible impact of batch-related operations (e.g., batch normalization (Bjorck et al. 2018)).

---

**Algorithm 2: Optimal Batch Size Searching**

---

**Input:** global batch size:  $gbs$ ,  
compute capability:  $\mathbf{P} = \{p_1, p_2, \dots, p_n\}$   
**Output:** Configuration:  $\mathbf{B} = \{b_1, b_2, \dots, b_n\}$

```
1: if  $ZeRO\_stage == 0$  or  $1$  then
2:   for  $i = 1, 2, \dots, n$  do
3:      $speed_i \leftarrow \max(p_i)$ 
4:   end for
5:    $speed_{cluster} \leftarrow \sum_{i=1}^n speed_i$ 
6:    $time_{optimal} \leftarrow gbs / speed_{cluster}$ 
7:   for  $i = 1, 2, \dots, n$  do
8:      $gmb_i \leftarrow \lfloor time_{optimal} \times speed_i \rfloor$ 
9:      $\delta_i \leftarrow time_{optimal} / speed_i$ 
10:     $u_i \leftarrow \delta_i \times speed_i$ 
11:   end for
12:    $batch_{remain} \leftarrow gbs - \text{sum}(\mathbf{B})$ 
13:   while  $batch_{remain} > 0$  do
14:      $i \leftarrow \min(u_i)$ 
15:      $update(u_i, gmb_i, batch_{remain})$ 
16:   end while
17: else if  $ZeRO\_stage == 2$  or  $3$  then
18:   for  $t$  in  $range(time_{min}, time_{max})$  do
19:     for  $i = 1, 2, \dots, n$  do
20:        $batch\_size_i \leftarrow find(g_i, t)$ 
21:     end for
22:      $micro\_batch\_size \leftarrow \sum_{i=1}^n batch\_size_i$ 
23:      $gas \leftarrow gbs / micro\_batch\_size$ 
24:      $wall\_time \leftarrow (t + time\_communication) \times gas$ 
25:     if  $wall\_time < optimal\_time$  then
26:        $optimal\_time \leftarrow wall\_time$ ;
27:        $b_i \leftarrow batch\_size_i$ 
28:     end if
29:   end for
30: end if
```

---

**Optimal Batch Size Searching** The choice of the batch size significantly affects throughput of GPU (Narayanan et al. 2021). During the process of increasing the batch size from 0 to  $mbs$ , the GPU’s speed for computing one batch data initially rises rapidly, then gradually levels off, eventually reaching a plateau where further increases in batch size do not improve computational speed. We conducted extensive experiments to validate this trend, and the results confirm that the relationship between batch size and compute capability of different GPU is universal.

Based on this observation, Poplar constructs complete performance curves with  $p_i$ . Initially, Poplar divides each  $TimeConsumedDuringStep$  in  $p_i$  by the corresponding batch size to determine the speed in some discrete points. Then, Poplar employs cubic spline interpolation (McKinley and Levine 1998) to fit the computing performance data for each GPU. Cubic spline interpolation approximates the original data by constructing a series of local cubic polynomials that join smoothly at each data point, ensuring both continuity and smoothness. In our work, the data obtained after Online Profiling exhibit characteristics well-suited to cubic spline interpolation, which also needs low computa-

tional complexity. In all of our experiments, we verified the optimality of the algorithm using local search. Specifically, once the algorithm generated a configuration, we tested all possible variations of each batch size within that configuration. Through this process, we confirmed that our algorithm can always identifies the configuration with the highest utilization. After obtaining the curve, Poplar can determine the batch size range that ensures each GPU operates at its peak performance. During the search process, Poplar aims to allocate batches within these ranges for each GPU. Given that Poplar treats each card independently, the performance of the cluster can be directly calculated through the sum of each GPU’s data. The process is detailed in Algorithm 2.

In ZeRO-1 and ZeRO-2, the search algorithm needs to ensure that all nodes complete their computations simultaneously after backward. Consequently, Poplar directly allocates  $gbs$  based on each GPU’s peak computational capacity, and determines the global micro batch size ( $gmb_i$ ) for each GPU. The  $gbs$  represents the amount of data each GPU needs to compute in a single iteration through gradient accumulation (Soydaner 2020) before the first synchronization. The sum of all GPUs’  $gmb_i$  equals  $gbs$ . Then, Poplar assigns a suitable  $b_i$  to each GPU, ensuring that  $b_i$  falls within the range that maximizes the GPU’s compute capability. Since the allocation progress is an integer programming problem, there will be some  $remain\_batch$  that remain unallocated. Therefore, in the second step, Poplar uses equation (2) and (3) to iteratively allocate  $remain\_batch$  to nodes with relatively lower workload, resulting in the last batch size ( $lbs$ ) for gradient accumulation.  $lbs$  is the final iteration for the gradient accumulation loop on each GPU.

In ZeRO-2 and ZeRO-3, the situation differs due to the additional communication. The search algorithm need to consider both load balancing and communication overhead. Selecting a smaller  $micro\_batch\_size$  requires a larger number of gradient accumulation steps to complete  $gbs$ , leading to a increase in communication time. Conversely, a larger  $micro\_batch\_size$  may increase the imbalance in workload, thus increasing idle times and decreasing cluster utilization. During the search process, Poplar explores every possible value of  $t$ , which represents the time it takes for the entire cluster to complete a data of  $micro\_batch\_size$ . Each value of  $t$  corresponds to a potential configuration. Poplar calculates the time required to complete  $gbs$  and identifies a configuration  $B$  that minimizes the wall time by iterating try a large amount of  $t$ . The wall time can be calculated as  $t$  plus the communication time per step  $time\_communication$ , and then multiplied by the total number of gradient accumulation steps. The function  $find$  is responsible for determining the maximum batch size GPU can compute given the current value of  $t$ .

The use of the last batch size ( $lbs$ ) and gradient accumulation steps ( $gabs$ ), along with independent allocation for each GPU, allows Poplar to overcome the traditional constraints between global batch size and micro batch size, thereby supporting the heterogeneity of quantity. This ability enables the workload to be distributed across GPUs in a fine-grained manner, thereby minimizing GPU idle times. Poplar strives to select larger batch sizes for each GPU to reduce overall

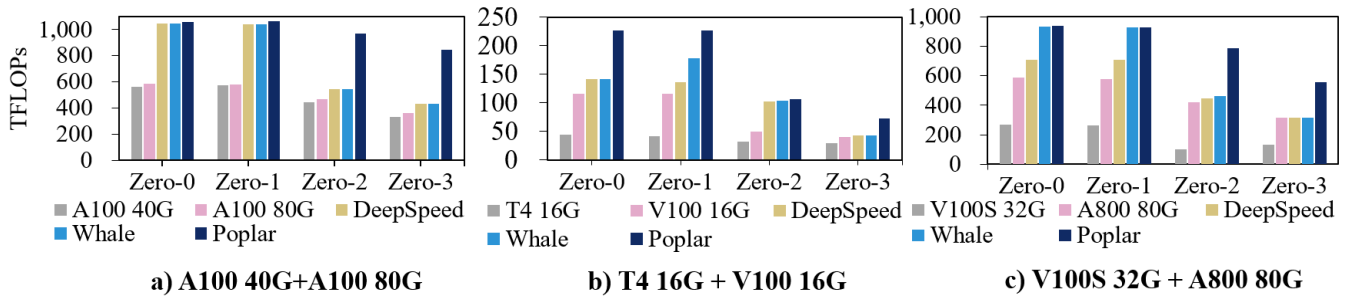


Figure 3: The performance on three types of heterogeneous environments. Poplar performs better than all baselines.

communication, as network links between devices in heterogeneous clusters often presents a bottleneck. Besides, we modify the dataloader to support the use of *lbs* and *gabs* in Poplar.

## Experiments

### Setup

We have implemented our work on PyTorch with around 2000+ lines of code. Poplar is decoupled from model training, thus any techniques that enhance LLM training efficiency can be seamlessly integrated through just a few lines of code, such as pipeline parallelism (Huang et al. 2019; Narayanan et al. 2019), parameter offloading (Ren et al. 2021; Rajbhandari et al. 2021), ZeRO++ (Wang et al. 2023).

Our experiments are conducted on three heterogeneous GPU clusters, each cluster contains two types of GPUs, as shown in Table 1. All experiments are evaluated on wikitext-2-v1 dataset (Merity et al. 2016). In our experiments, we use TFLOPs (FLOPs/1e12) as the metric for evaluation end-to-end utilization of cluster. For each experiment, we conduct 50 iterations and use the average value.

### Models and Baselines

In the main experiments, we use Llama (Touvron et al. 2023) to validate performance. Subsequently, we employ Llama and BERT (Devlin et al. 2019) of varying sizes to evaluate generality. We maintain a global batch size of 2 million tokens throughout our experiments. To clearly illustrate Poplar’s acceleration capabilities, we establish four baselines. Baseline 1 uses less powerful homogeneous GPUs, while baseline 2 uses more powerful homogeneous GPUs. The third baseline employs DeepSpeed, a state-of-the-art distributed training system. We manually allocate maximum batch sizes that meet the constraints for baseline 3. The fourth baseline is Whale (Jia et al. 2022), a state-of-the-art heterogeneous training system that supports hetero-aware load balancing. In baseline 4, we also manually configure the maximum batch sizes to be consistent with its strategy.

### Performance

In the main experiments, we comprehensive verify and compare Poplar’s performance across three types of heterogeneous environments. To better illustrate the differences among all heterogeneous GPUs and Poplar’s acceleration in

| Cluster | GPU        | Number | Intra-Link |
|---------|------------|--------|------------|
| A       | A100 80GB  | 4      | NVLink     |
|         | A100 40GB  | 4      | PCIE       |
| B       | V100 16GB  | 2      | PCIE       |
|         | T4 16GB    | 2      |            |
| C       | A800 80GB  | 4      | PCIE       |
|         | V100S 32GB | 4      |            |

Table 1: Three clusters in our experiments, each cluster contains two types of GPUs. Number represents the quantity of corresponding GPUs in the cluster, while Inter-Link denotes the networking connection between GPUs.

various clusters, we set the model’s parameter to 0.5 billion, ensuring that all baselines can run. As shown in Figure 3, the three clusters represent three distinct heterogeneous scenarios: (1) different memory capacities with equal compute capabilities, (2) different compute capabilities with equal memory capacities, and (3) different memory capacities with different compute capabilities. Figures 3(a), 3(b), and 3(c) display the performance of clusters A, B, and C, respectively. Within each subplot, the five baselines are presented from left to right for each ZeRO stage.

The experimental results demonstrate that Poplar performs well across all scenarios. The improvements in ZeRO-0 and ZeRO-1, as shown in Figures 3(a) and 3(c), arise from a larger *gabs* to A100 80G and A800 80G. Notably, despite having the same FLOPs rating for A100 40G and A100 80G, the A100 80G supports a larger *mbs*, leading to an improvement per iteration. In cluster A, Whale performs similarly to DeepSpeed, which lacks heterogeneity awareness due to the homogeneous matrix computation capabilities of the two kind of GPUs. The enhancements in ZeRO-0 and ZeRO-1 depicted in Figure 3(b) mainly come from the finer-grained aware of GPU compute capability, indicating that, under certain heterogeneous hardware conditions, the overhead of components other than matrix multiplication should not be overlooked. The improvements in ZeRO-2 and ZeRO-3 are substantial across all three cases, as Poplar effectively reduces the number of gradient accumulation steps, thereby decreasing communication volume. This ability is crucial for training on heterogeneous clusters, where the network

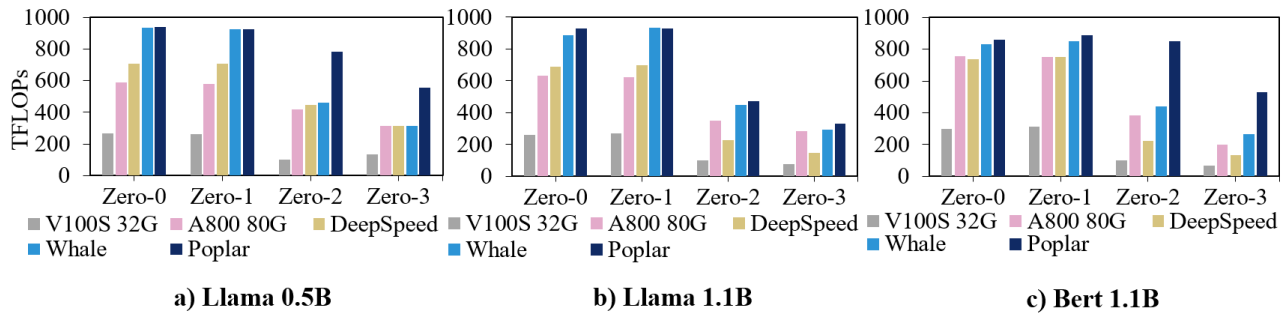


Figure 4: Results on different models. Poplar performs better on BERT compared to Llama. Due to a small micro batch size, Poplar performs less well at 1.1B parameters than 0.5B parameters.

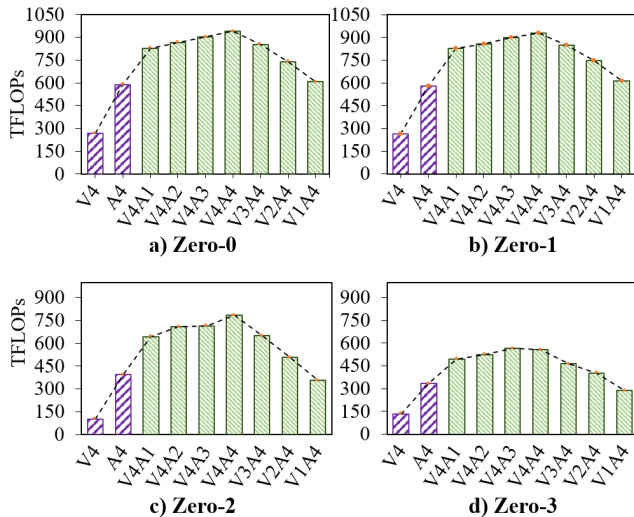


Figure 5: The evaluation on Poplar’s training capabilities across varying numbers of heterogeneous GPUs. The numbers and letters in the figure indicate the quantity of corresponding GPUs, for example, V4 denotes four V100S, A4 denotes four A800, and V4A1 denotes four V100S with one A800.

may encounter a bottleneck. Our current method does not address the issue of heterogeneous networking, and we recognize the importance of this aspect and plan to include considerations in our future work. To ensure the practicality of our approach, all experiments in the paper were conducted on PCIe. The ablation study of Poplar is given in Appendix.

**Different Quantity** In the second experiment, we evaluate Poplar’s training capabilities across varying numbers of heterogeneous GPUs. We select cluster C to evaluate the support for arbitrary numbers of heterogeneous GPUs, with both differing memory capacities and compute capabilities. As shown in Figure 5, each subplot includes three sections of data: the first section uses only the weaker V100S, the second section uses only the stronger A800, and the third section uses different proportions of the two heterogeneous GPU types, including ratios of 4:1, 4:2, 4:3, 4:4, 3:4, 2:4, and 1:4 for A800 to V100S. From the figure, it is clear that

as the number of GPUs increases, Poplar achieves progressively higher cluster performance. Additionally, it is evident that a reduction in the number of A800 leads to a more significant decrease in overall cluster performance compared to a reduction in the number of V100S.

**Different models** Heterogeneous clusters are highly sensitive to model size and structure. Therefore, we conducted additional experiments to demonstrate Poplar’s acceleration capabilities on heterogeneous clusters. We used a 1.1B Llama model and a 1.1B BERT model for comparison with the 0.5B Llama model, as shown in Figure 4. In Figure 4(b), Poplar achieves up to a 2.27x improvement over DeepSpeed and up to a 1.14x improvement over Whale. In Figure 4(c), Poplar demonstrates even more substantial cluster performance improvements, achieving up to a 3.92x improvement over DeepSpeed and up to a 1.98x improvement over Whale.

## Time Overhead

Poplar achieves performance gains through fine-grained planning before the start of training. In this section, we analyze the time overhead incurred during Poplar’s preliminary analysis phase. The majority of the time overhead is concentrated in Online Profiling, as Poplar requires multiple runs of the model to aware more accurate heterogeneity. However, this approach eliminates the need for researchers to manually find suitable batch sizes. The runtime of the Online Profiling phase can vary depending on several factors, including the state of the GPUs, network links, ZeRO stage, model size and architecture. The Offline Analysis phase primarily handles numerical calculations and incurs relatively minimal time overhead. We detail the time overhead for our main experiment in Appendix.

## Conclusion

In this paper, we propose a distributed training system which extends each ZeRO stage with heterogeneous-aware capabilities. Our system can adapt to a broader range of heterogeneous environments by dynamically assign batches to each GPU independently. In the experiments, we show that our system outperforms state-of-the-art training systems.

## Acknowledgements

This work was sponsored by the National Key R&D Program of China(No.2022YFB3103600).

## References

- Bjorck, N.; Gomes, C. P.; Selman, B.; and Weinberger, K. Q. 2018. Understanding batch normalization. *Advances in neural information processing systems*, 31.
- Cao, J.; Zhu, Z.; and Zhou, X. 2021. SAP-SGD: Accelerating distributed parallel training with high communication efficiency on heterogeneous clusters. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, 94–102. IEEE.
- Chen, C.; Weng, Q.; Wang, W.; Li, B.; and Li, B. 2021. Accelerating distributed learning in non-dedicated environments. *IEEE Transactions on Cloud Computing*, 11(1): 515–531.
- Choquette, J. 2022. Nvidia hopper gpu: Scaling performance. In *2022 IEEE Hot Chips 34 Symposium (HCS)*, 1–46. IEEE Computer Society.
- Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Burstein, J.; Doran, C.; and Solorio, T., eds., *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171–4186. Minneapolis, Minnesota: Association for Computational Linguistics.
- Geng, J.; Cao, J.; Jia, H.; Zhu, Z.; Fang, H.; Gao, C.; Ji, C.; Jia, G.; Han, G.; and Zhou, X. 2023. Heter-Train: A Distributed Training Framework Based on Semi-Asynchronous Parallel Mechanism for Heterogeneous Intelligent Transportation Systems. *IEEE Transactions on Intelligent Transportation Systems*, 25(1): 959–972.
- Huang, Y.; Cheng, Y.; Bapna, A.; Firat, O.; Chen, D.; Chen, M.; Lee, H.; Ngiam, J.; Le, Q. V.; Wu, Y.; et al. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32.
- Jia, X.; Jiang, L.; Wang, A.; Xiao, W.; Shi, Z.; Zhang, J.; Li, X.; Chen, L.; Li, Y.; Zheng, Z.; et al. 2022. Whale: Efficient giant model training over heterogeneous {GPUs}. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 673–688.
- Jia, Z.; Zaharia, M.; and Aiken, A. 2019. Beyond data and model parallelism for deep neural networks. *Proceedings of Machine Learning and Systems*, 1: 1–13.
- Kwon, W.; Yu, G.-I.; Jeong, E.; and Chun, B.-G. 2020. Nimble: Lightweight and parallel gpu task scheduling for deep learning. *Advances in Neural Information Processing Systems*, 33: 8343–8354.
- Lai, Z.; Li, S.; Tang, X.; Ge, K.; Liu, W.; Duan, Y.; Qiao, L.; and Li, D. 2023. Merak: An efficient distributed dnn training framework with automated 3d parallelism for giant foundation models. *IEEE Transactions on Parallel and Distributed Systems*, 34(5): 1466–1478.
- Li, D.; Wang, H.; Xing, E.; and Zhang, H. 2022. Amp: Automatically finding model parallel strategies with heterogeneity awareness. *Advances in Neural Information Processing Systems*, 35: 6630–6639.
- Li, S.; Ben-Nun, T.; Girolamo, S. D.; Alistarh, D.; and Hoefer, T. 2020. Taming unbalanced training workloads in deep learning with partial collective operations. In *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 45–61.
- Luo, Q.; Lin, J.; Zhuo, Y.; and Qian, X. 2019. Hop: Heterogeneity-aware decentralized training. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 893–907.
- McKinley, S.; and Levine, M. 1998. Cubic spline interpolation. *College of the Redwoods*, 45(1): 1049–1060.
- Merity, S.; Xiong, C.; Bradbury, J.; and Socher, R. 2016. Pointer Sentinel Mixture Models. arXiv:1609.07843.
- Narayanan, D.; Harlap, A.; Phanishayee, A.; Seshadri, V.; Devanur, N. R.; Ganger, G. R.; Gibbons, P. B.; and Zaharia, M. 2019. PipeDream: Generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM symposium on operating systems principles*, 1–15.
- Narayanan, D.; Shoeybi, M.; Casper, J.; LeGresley, P.; Patwary, M.; Korthikanti, V.; Vainbrand, D.; Kashinkunti, P.; Bernauer, J.; Catanzaro, B.; et al. 2021. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–15.
- Park, J. H.; Yun, G.; Chang, M. Y.; Nguyen, N. T.; Lee, S.; Choi, J.; Noh, S. H.; and Choi, Y.-r. 2020. {HetPipe}: Enabling large {DNN} training on (whimpy) heterogeneous {GPU} clusters through integration of pipelined model parallelism and data parallelism. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 307–321.
- Patarasuk, P.; and Yuan, X. 2009. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2): 117–124.
- Rajbhandari, S.; Rasley, J.; Ruwase, O.; and He, Y. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–16. IEEE.
- Rajbhandari, S.; Ruwase, O.; Rasley, J.; Smith, S.; and He, Y. 2021. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, 1–14.
- Rasley, J.; Rajbhandari, S.; Ruwase, O.; and He, Y. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 3505–3506.
- Ren, J.; Rajbhandari, S.; Aminabadi, R. Y.; Ruwase, O.; Yang, S.; Zhang, M.; Li, D.; and He, Y. 2021. {Zero-offload}: Democratizing {billion-scale} model training. In

2021 *USENIX Annual Technical Conference (USENIX ATC 21)*, 551–564.

Shoeybi, M.; Patwary, M.; Puri, R.; LeGresley, P.; Casper, J.; and Catanzaro, B. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. arXiv:1909.08053.

Smith, S.; Patwary, M.; Norick, B.; LeGresley, P.; Rajbhandari, S.; Casper, J.; Liu, Z.; Prabhume, S.; Zerveas, G.; Korthikanti, V.; et al. 2022. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. arXiv:2201.11990.

Song, Y.; Mi, Z.; Xie, H.; and Chen, H. 2023. Powerinfer: Fast large language model serving with a consumer-grade gpu. arXiv preprint arXiv:2312.12456.

Soydaner, D. 2020. A comparison of optimization algorithms for deep learning. *International Journal of Pattern Recognition and Artificial Intelligence*, 34(13): 2052013.

Tang, Z.; Du, L.; Zhang, X.; Yang, L.; and Li, K. 2021. AEML: An acceleration engine for multi-GPU load-balancing in distributed heterogeneous environment. *IEEE Transactions on Computers*, 71: 1344–1357.

Thakur, R.; Rabenseifner, R.; and Gropp, W. 2005. Optimization of collective communication operations in MPICH. *The International Journal of High Performance Computing Applications*, 19(1): 49–66.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023. Llama 2: Open foundation and fine-tuned chat models. arXiv:2307.09288.

Wang, G.; Qin, H.; Jacobs, S. A.; Holmes, C.; Rajbhandari, S.; Ruwase, O.; Yan, F.; Yang, L.; and He, Y. 2023. Zero++: Extremely efficient collective communication for giant model training. arXiv:2306.10209.

Weng, Q.; Xiao, W.; Yu, Y.; Wang, W.; Wang, C.; He, J.; Li, Y.; Zhang, L.; Lin, W.; and Ding, Y. 2022. {MLaaS} in the wild: Workload analysis and scheduling in {Large-Scale} heterogeneous {GPU} clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 945–960.

Zheng, L.; Li, Z.; Zhang, H.; Zhuang, Y.; Chen, Z.; Huang, Y.; Wang, Y.; Xu, Y.; Zhuo, D.; Xing, E. P.; et al. 2022. Alpa: Automating inter- and {Intra-Operator} parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 559–578.

Zhou, Q.; Guo, S.; Qu, Z.; Li, P.; Li, L.; Guo, M.; and Wang, K. 2020. Petrel: Heterogeneity-aware distributed deep learning via hybrid synchronization. *IEEE Transactions on Parallel and Distributed Systems*, 32(5): 1030–1043.

Zhu, H.; Akrouf, M.; Zheng, B.; Pelegri, A.; Jayarajan, A.; Phanishayee, A.; Schroeder, B.; and Pekhimenko, G. 2018. Benchmarking and analyzing deep neural network training. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, 88–100. IEEE.