HAP: SPMD DNN Training on Heterogeneous GPU Clusters with Automated Program Synthesis

Shiwei Zhang[1] Lansong Diao[2] Chuan Wu[1]
Zongyan Cao[2] Siyu Wang[2] Wei Lin[2]

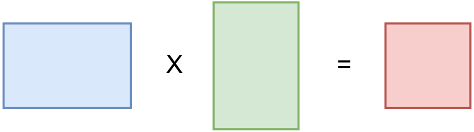
[1]The University of Hong Kong
[2]Alibaba Group

香 港 大 學
THE UNIVERSITY OF HONG KONG

Alibaba

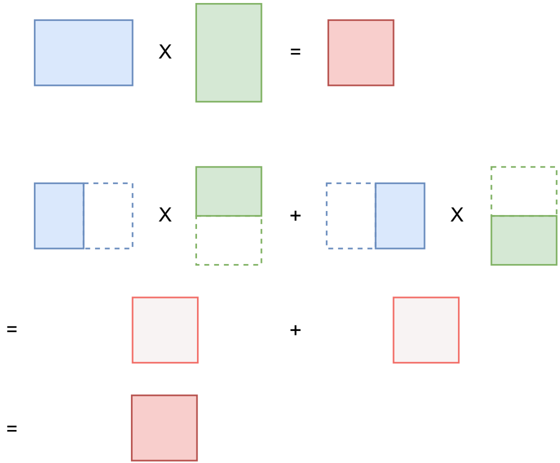
Tensor Programs

Neural network models are implemented as tensor programs, where the variables are multi-dimentional arrays (tensors). Tensor programs are usually executed on accelerator devices such as GPUs.



z = matmul(x, w)

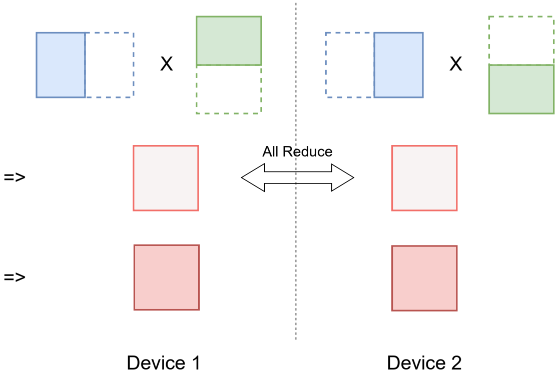
Tensor Sharding



z = matmul(x, w)

x_1 = split(x)[1] ①
x_2 = split(x)[2] ②
w_1 = split(w)[1] ③
w_2 = split(w)[2] ④
z_1 = matmul(x_1, w_1) ⑤
z_2 = matmul(x_2, w_2) ⑥
$z = z_1 + z_2$ ⑦

SPMD Parallelism



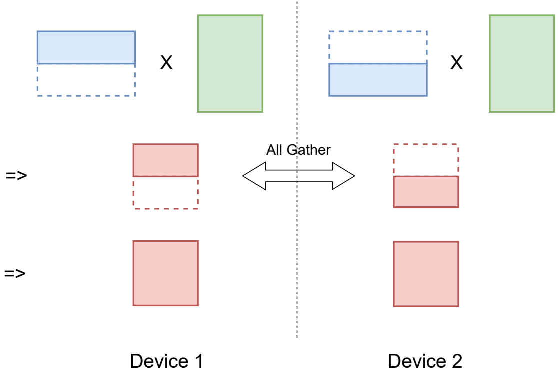
Device 1 (rank = 1)

```
x' = split(x)[rank]
w' = split(w)[rank]
z' = matmul(x', w')
z = all_reduce(z', rank)
```

Device 2 (rank = 2)

```
x' = split(x)[rank]
w' = split(w)[rank]
z' = matmul(x', w')
z = all_reduce(z', rank)
```

Multiple Ways to Parallelize a Tensor Program



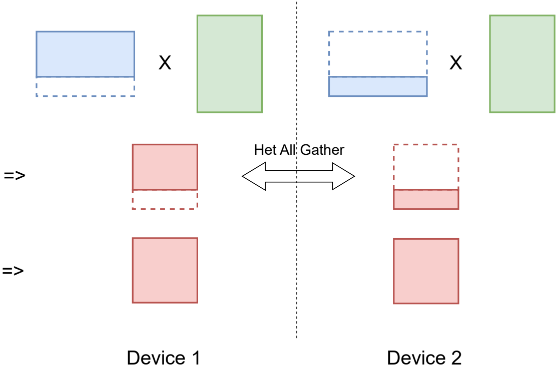
Device 1 (rank = 1)

x' = split(x)[rank]
z' = matmul(x', w)
z = all_gather(z', rank)

Device 2 (rank = 2)

x' = split(x)[rank]
z' = matmul(x', w)
z = all_gather(z', rank)

SPMD Parallelism on Heterogeneous Clusters



Device 1 (rank = 1)

x' = split(x, sharding_ratios[rank])
z' = matmul(x', w)
z = het_all_gather(z', rank)

Device 2 (rank = 2)

x' = split(x, sharding_ratios[rank])
z' = matmul(x', w)
z = het_all_gather(z', rank)

Challenges

z = matmul(x, w)

①

②

x' = split(x, **sharding_ratios**[rank])

z' = matmul(x', w)

z = **het_all_gather**(z', rank)

③

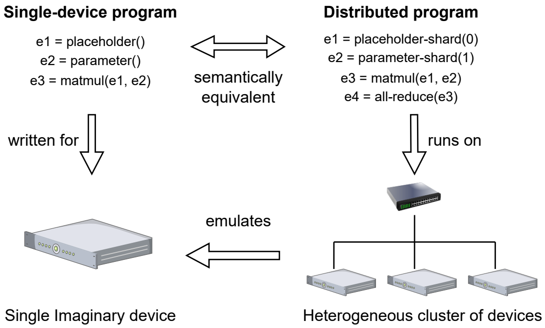
1. How to generate a distributed program that produces equivalent results
2. How to determine the sharding ratios for the heterogeneous devices
3. How to choose the communication method on heterogeneous networks

Automated Program Synthesis

Automated Program Synthesis

We synthesize a distributed program from scratch on a distributed instruction set that emulates the single-device program.

To ensure the equivalence of the original and synthesized programs, we formalize the semantics of the single-device program and generate the distributed program under a semantic constraint.



Automated Program Synthesis

program	$\in$	[*instruction*]
instruction	:=	*computation* | *communication*
computation	:=	*tensor* $\leftarrow$ *optype*([*tensor*])
communication	:=	*tensor* $\leftarrow$ *collective*(*tensor*, *dim*)
dim	$\in$	$\{0, 1, \ldots\}$
collective	:=	`All-Reduce` | `All-Gather` | ...
optype	:=	`MatMul` | `Sigmoid` | ...

Syntax Rules

$$\frac{\forall e \in E}{\{\, e \mid \texttt{All-Reduce} \,\}\ \texttt{All-Reduce}(\bar{e})\ \{\, e \mid \texttt{Identity} \,\}}$$

$$\frac{\forall e \in E,\ \forall d \in \mathrm{dims}(e)}{\{\, e \mid \texttt{All-Reduce} \,\}\ \texttt{Reduce-Scatter}(\bar{e}, d)\ \{\, e \mid \texttt{All-Gather}(d) \,\}}$$

$$\frac{\forall e \in E,\ \forall d_1, d_2 \in \mathrm{dims}(e),\ d_1 \neq d_2}{\{\, e \mid \texttt{All-Gather}(d_1) \,\}\ \texttt{All-To-All}(\bar{e}, d_1, d_2)\ \{\, e \mid \texttt{All-Gather}(d_2) \,\}}$$

$$\frac{\forall e \in E,\ \forall d \in \mathrm{dims}(e)}{\{\, e \mid \texttt{All-Gather}(d) \,\}\ \texttt{All-Gather}(\bar{e}, d)\ \{\, e \mid \texttt{Identity} \,\}}$$

$$\frac{\forall e_1, e_2, e_3 \in E,\ e_3 = \texttt{MatMul}(e_1, e_2)}{\{\, e_1 \mid \texttt{All-Gather}(0),\ e_2 \mid \texttt{Identity} \,\}\ \texttt{MatMul}(\bar{e}_1, \bar{e}_2)\ \{\, e_3 \mid \texttt{All-Gather}(0) \,\}}$$

$$\frac{\forall e_1, e_2, e_3 \in E,\ e_3 = \texttt{MatMul}(e_1, e_2)}{\{\, e_1 \mid \texttt{Identity},\ e_2 \mid \texttt{All-Gather}(1) \,\}\ \texttt{MatMul}(\bar{e}_1, \bar{e}_2)\ \{\, e_3 \mid \texttt{All-Gather}(1) \,\}}$$

$$\frac{\forall e_1, e_2, e_3 \in E,\ e_3 = \texttt{MatMul}(e_1, e_2)}{\{\, e_1 \mid \texttt{All-Gather}(1),\ e_2 \mid \texttt{All-Gather}(0) \,\}\ \texttt{MatMul}(\bar{e}_1, \bar{e}_2)\ \{\, e_3 \mid \texttt{All-Reduce} \,\}}$$

Semantic Rules

Load Balancing

z = matmul(x, w)

1

2

x' = split(x, **sharding_ratios**[rank])

z' = matmul(x', w)

z = **het_all_gather**(z', rank)

3

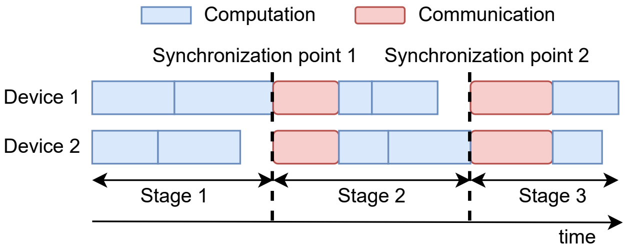
1 How to generate a distributed program that produces equivalent results

2 How to determine the sharding ratios for the heterogeneous devices

3 How to choose the communication method on heterogeneous networks

Cost Model

As synchronization among devices is required during collective communication, the execution of the distributed program can be divided into stages.



Optimization Problem Formulation

$$
\begin{aligned}
\min \quad & \sum_{i \in \mathrm{stages}(Q)} \left(\mathrm{comm}^{(i)}(B) + \max_{j \in [m]} \mathrm{comp}_j^{(i)}(B_j)\right) \\
\text{subject to:} \quad & \sum_{j=1}^{m} B_j = 1, \\
& B_j \geq 0, \quad \forall j \in [m]
\end{aligned}
$$

Q: distributed program
B: sharding ratios for the devices
comm, **comp**: linear models of communication and computation times.

Communication Optimization

```
z = matmul(x, w)
```

①

②

```
x' = split(x, sharding_ratios[rank])
z' = matmul(x', w)
z = het_all_gather(z', rank)
```

③

① How to generate a distributed program that produces equivalent results

② How to determine the sharding ratios for the heterogeneous devices

③ How to choose the communication method on heterogeneous networks

Optimal Communication Method Depends

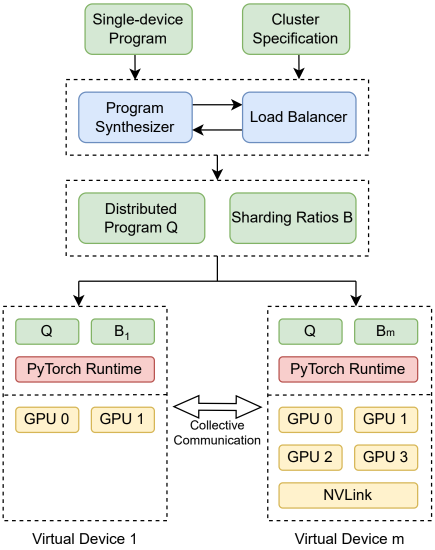
Sufficient Factor Broadcasting

Implementation

Implementation

We implement HAP on top of PyTorch.

We round the sharding ratios to ensure that the program runs with any number of devices, even if it does not evenly divide the tensor dimensions.



Implementation

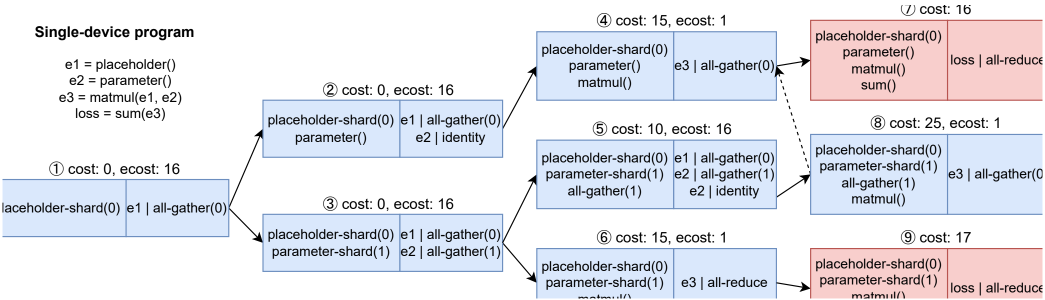
- **Annotator**: Label the possible sharding methods for each operator, infer the tensor sizes, and estimate the FLOPs.
- **Strategy Searcher**: Take annotated graph as input and search for the optimal sharding strategy.
- **Compiler**: Modify the graph according to the strategy and applies Duplex.

Evaluation

Experimental Setup

- **Testbed**: 8 machines on public cloud, each with 8 V100 GPUs and NVLink, connected by 10Gbps network.
- **Benchmarks**: BERT (language modeling) and ViT (image classification), with two variants of MoE layers, SGMoE and Switch.
- **Baselines**: DeepSpeed, FastMoE, PyTorch DDP, and Horovod.

Per-iteration training time



HiDup outperforms baselines when scaling up, because the collective communication becomes slower with more cards and HiDup can mitigate the increased communication overhead with our Duplex design.

Single Machine Performance

Figure 11. A* search example. Names of distributed tensors (e.g., $\bar{e}_1$) are omitted.

append multiple communication instructions for each
nsor because they introduce new properties to the pro-
am, even though most of these properties are not utilized.
or a Hoare triple that generates a communication instruc-
on of reference tensor e, we append a special property
| ¬`Communicated` to its precondition and e | `Communicated`
its postcondition. This makes communication instructions
the same reference tensor conflict with each other, so that
most one of them can appear in one distributed program.

stage and the collective communication operations take tl
same time across devices (Fig. 6), the computation time
the i-th stage is the maximum computation time among tl
devices, i.e., $\max_j \text{comp}_j^{(i)}(B_j)$. We then solve the followi
problem to obtain B:

$$\min \sum_{i\in\text{stages}(Q)} \left(\text{comm}^{(i)}(B) + \max_{j\in[m]} \text{comp}_j^{(i)}(B_j)\right) \quad ($$

$$\text{subject to:} \quad \sum^{m} B_j = 1,$$

Pure-DP methods perform well with high bandwidth. HiDup automatically identifies similar strategies and achieves comparable performance despite of the additional overheads introduced by our Duplex design.

Per-iteraion time breakdown

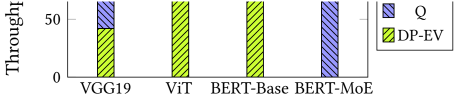


Figure 15. Ablation study.

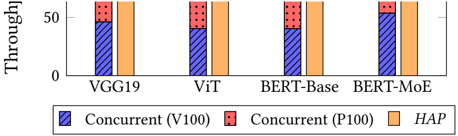


Figure 16. Training multiple models.

4 Ablation Study

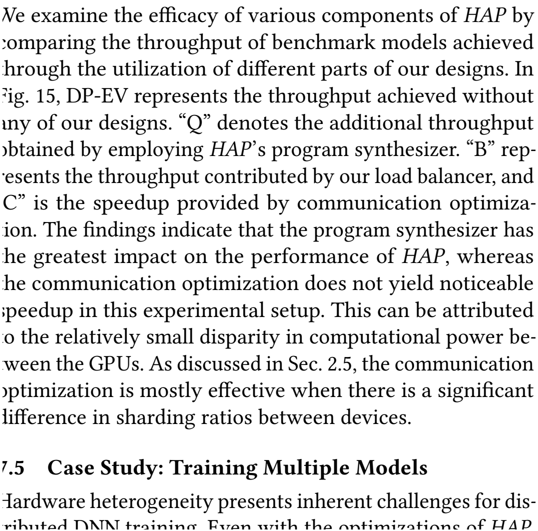
e examine the efficacy of various components of *HAP* by demand for the ability to utilize all available resources

HiDup's total time is similar to the baselines, but it achieves shorter wall time by overlapping computation and communication.

SPMD Strategy

HiDup generates different strategies for the same model on different clusters.

It automatically identifies expert-designed strategies for common models.


Ve examine the efficacy of various components of *HAP* by :omparing the throughput of benchmark models achieved :hrough the utilization of different parts of our designs. In Fig. 15, DP-EV represents the throughput achieved without any of our designs. “Q” denotes the additional throughput obtained by employing *HAP*’s program synthesizer. “B” represents the throughput contributed by our load balancer, and C” is the speedup provided by communication optimization. The findings indicate that the program synthesizer has the greatest impact on the performance of *HAP*, whereas the communication optimization does not yield noticeable speedup in this experimental setup. This can be attributed o the relatively small disparity in computational power between the GPUs. As discussed in Sec. 2.5, the communication optimization is mostly effective when there is a significant difference in sharding ratios between devices.

7.5 Case Study: Training Multiple Models

Hardware heterogeneity presents inherent challenges for distributed DNN training. Even with the optimizations of *HAP*

Thank you!

Email: swzhang@cs.hku.hk