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

Shiwei Zhang, Lansong Diao, Chuan Wu,
Zongyan Cao, Siyu Wang, Wei Lin

The University of Hong Kong

香 港 大 學
THE UNIVERSITY OF HONG KONG

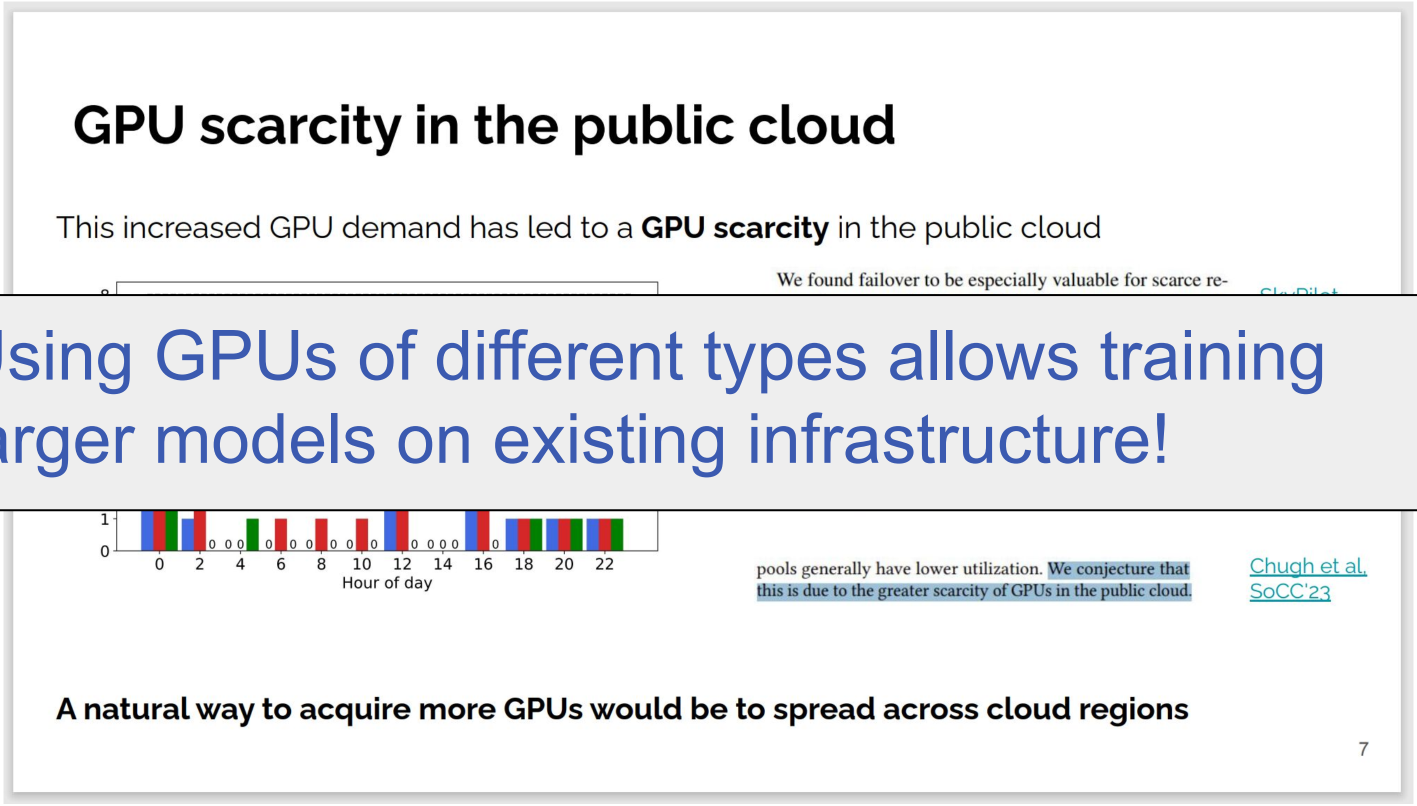
Alibaba Inc.

Alibaba

Motivation: ML Models Are Demanding

- PanGu-Σ (2023) is trained on 512 Ascend 910 NPUs.
- Gemini 1.5 Pro (2024) is trained on multiple 4096-chip pods of TPUv4.
- Llama 3 (2024) is trained on two clusters with 24,000 GPUs.

Motivation: Shortage of Homogeneous GPUs



GPU scarcity in the public cloud

This increased GPU demand has led to a **GPU scarcity** in the public cloud

We found failover to be especially valuable for scarce re-

pools generally have lower utilization. We conjecture that this is due to the greater scarcity of GPUs in the public cloud.

Chugh et al, SoCC'23

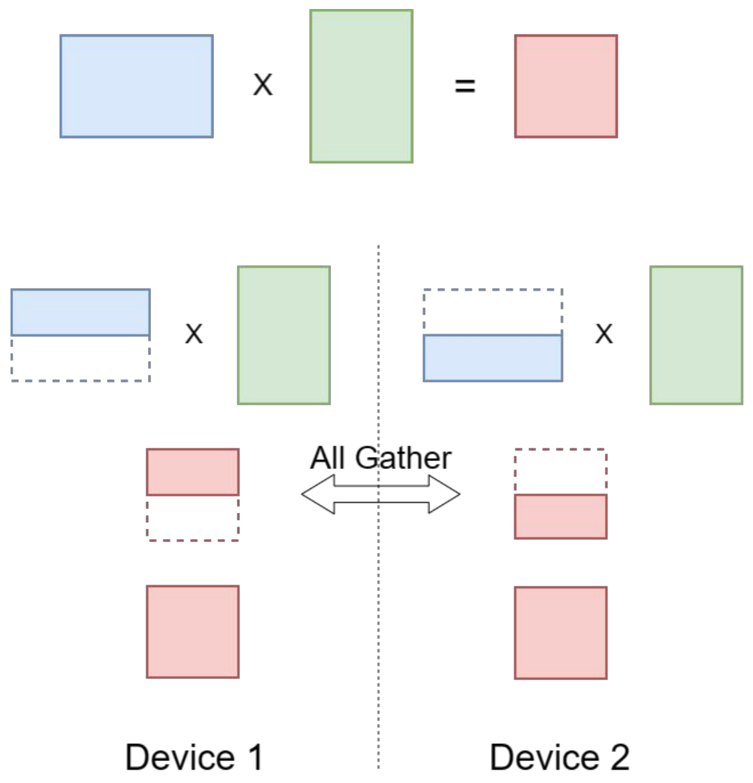
A natural way to acquire more GPUs would be to spread across cloud regions

7

Using GPUs of different types allows training larger models on existing infrastructure!

Foteini Strati et al. EuroMLSys ‘24

Background: Tensor Programs and SPMD Parallelism



z = matmul(x, w)

Device 1 (rank = 1)

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

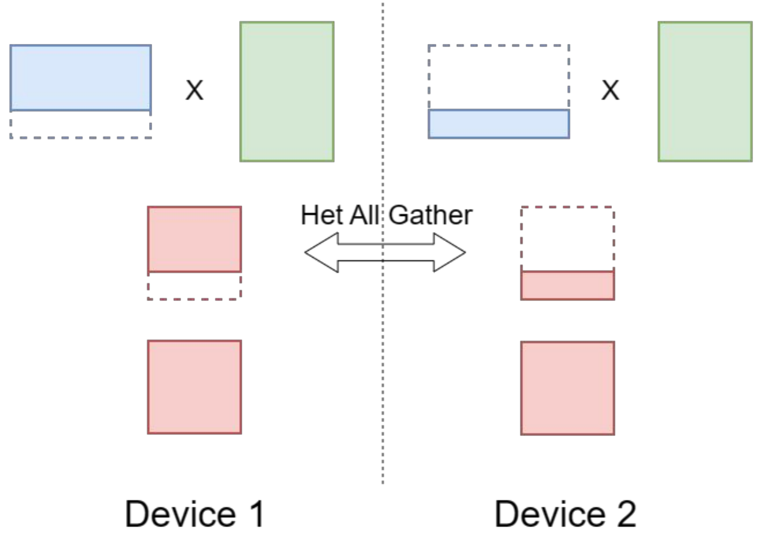
Device 2 (rank = 2)

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

Multiple devices run the same program to jointly produce a result that equals the single-card program.

Background: Uneven Sharding

To fully utilize devices with different computation powers, we can unevenly shard the tensors to balance the workloads.



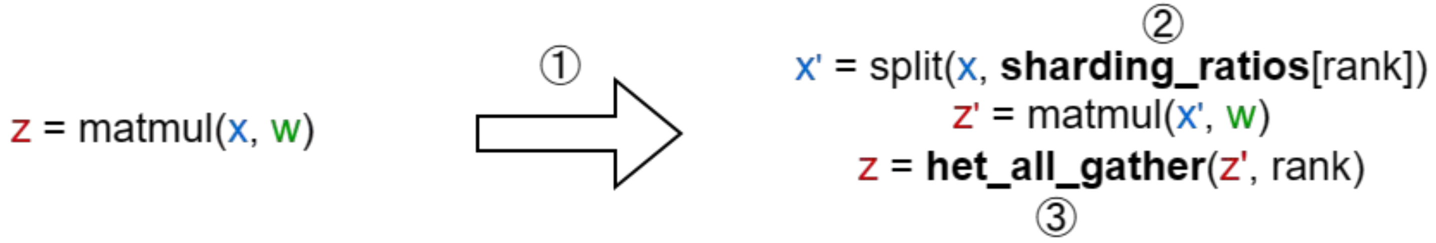
Device 1 (rank = 1)

```
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)
```

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

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

Challenge 1: Tensor Sharding

- Different operators can be sharded on different dimensions
  - Sharding a BatchNorm operation may reduce training stability
  - An Einsum operator can be sharded on dimensions that depend on the actual code string
- Collective communication needs to be inserted
  - AllGather and AllReduce are used to aggregate tensor shards
  - AllToAll may be used when two consecutive operators use different sharding methods

Solution 1: Automated Program Synthesis

We extract the semantics of the single-card program into Hoare triples.

$$\{\, precondition \,\}\ \ instruction\ \ \{\, postcondition \,\}$$

When the precondition is met, inserting the instruction establishes the postcondition.

The precondition and postcondition are represented as sets of properties in the form of **tensor | op**, which denotes that if we execute the operator **op**, we can obtain a result that equals **tensor**.

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

Solution 1: Automated Program Synthesis (cont'd)

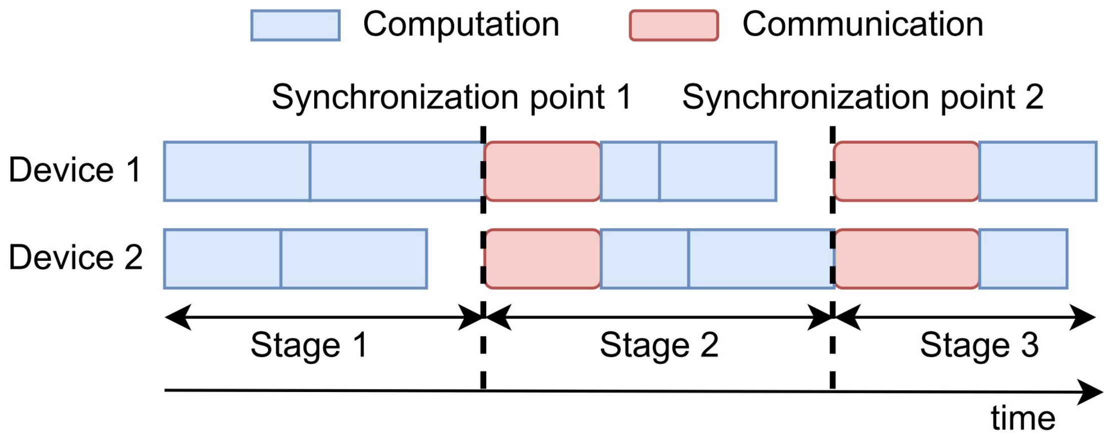
The search space of distributed programs is huge.

We design an A*-based search algorithm combined with the idea of dynamic programming to practically synthesize distributed programs for common models.

```
Input: computation graph $(V, E)$, sharding ratios $B$
Output: Optimal distributed program $Q^*$

Initialize a priority queue $S$ with an empty program $Q_\emptyset$
Initialize best program $Q^*$ = null and set $\text{cost}(Q^*) = \infty$
while $\exists Q \in S, \text{score}(Q) < \text{cost}(Q^*)$ do
  Remove the program $Q$ with lowest score from $S$
  for $\{\textit{precondition}\}$ instruction $\{\textit{postcondition}\} \in \mathcal{T}$ of the single-device program where $\textit{precondition} \subseteq P(Q)$ and $\textit{postcondition} \not\subseteq P(Q)$ do
    $Q' = Q \cup \textit{instruction}$;   $P(Q') = P(Q) \cup \textit{postcondition}$
    if $\exists Q_s \in S$ s.t. $P(Q_s) \supseteq P(Q')$ and $\text{cost}(Q_s) \leq \text{cost}(Q')$ then
      continue
    end if
    for $Q_s \in S$ where $P(Q') \supseteq P(Q_s)$ and $\text{cost}(Q') \leq \text{cost}(Q_s)$ do
      remove $Q_s$ from $S$
    end for
    if $Q'$ is complete then
      $Q^* \leftarrow Q'$ if $\text{cost}(Q') < \text{cost}(Q^*)$
    else
      add $Q'$ into $S$
    end if
  end for
end while
```

Challenge 2: Load Balancing



- Communication time depends on the largest shard
  - Even sharding minimizes communication time
- Computation time on each device is proportional to the shard size
  - Sharding sizes proportional to device speeds balance the computation time

Solution 2: Linear Programming

We profile the computation and communication operations with different input shapes and fit a linear model for each operator.

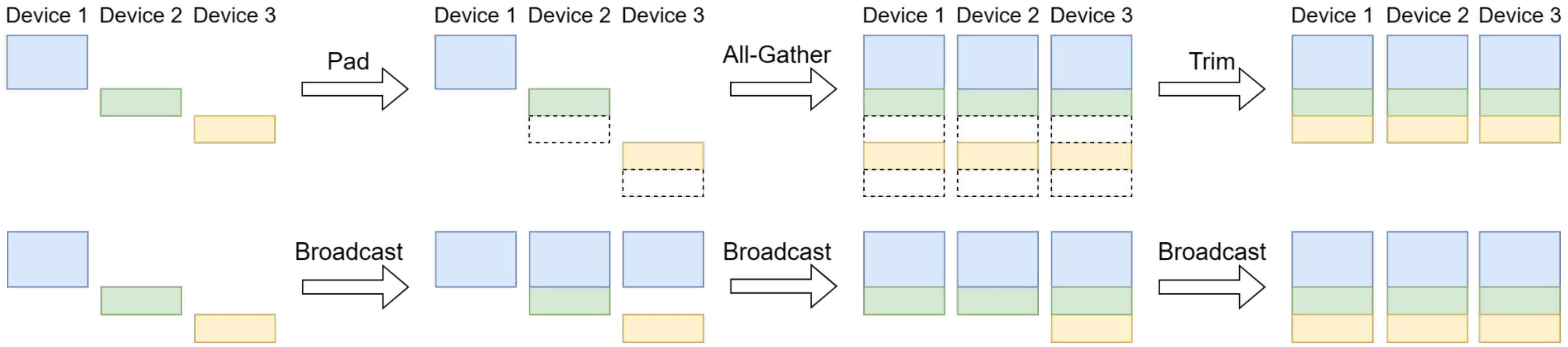
The training time can be expressed as a linear function of the sharding ratios.

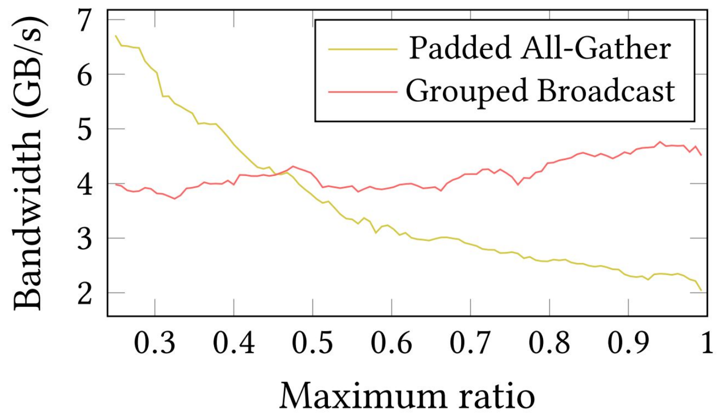
$$\min \sum_{i \in \text{stages}(Q)} \left(\text{comm}^{(i)}(B) + \max_{j \in [m]} \text{comp}_j^{(i)}(B_j)\right)$$

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

$$B_j \geq 0, \quad \forall j \in [m]$$

Challenges 3: Communication with Uneven Shards





The optimal communication method depends on the sharding ratios.

Solution 3: Integrate Communication Into Program Synthesis

We support different communication methods by adding multiple rules during program synthesis.

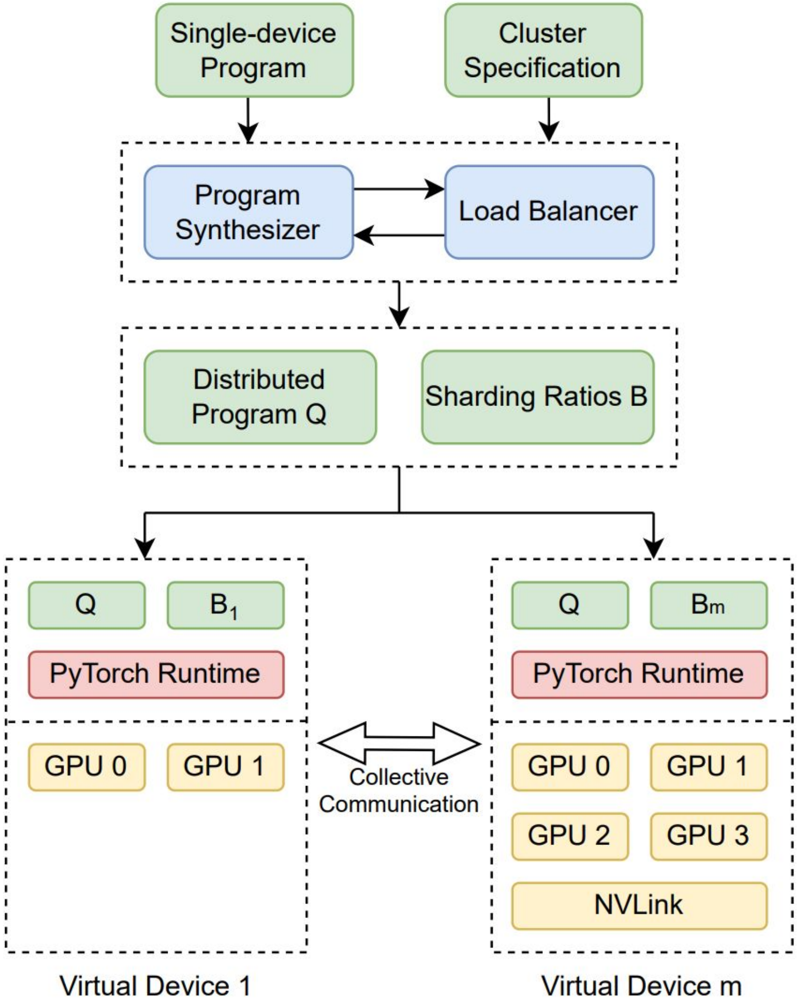
$$\frac{\forall e \in E,\ \forall d \in \mathrm{dims}(e)}{\{\, e \mid \texttt{All-Gather}(d) \,\}\ \texttt{Grouped-Broadcast}(\bar{e}, d)\ \{\, e \mid \texttt{Identity} \,\}}$$

We alternatively optimize the distributed program **Q** and the sharding ratios **B** to approach the global optimum.

$$Q^{(s)} = \arg\min_{Q}\ t(Q, B^{(s-1)})$$
$$B^{(s)} = \arg\min_{B}\ t(Q^{(s)}, B)$$

Implementation

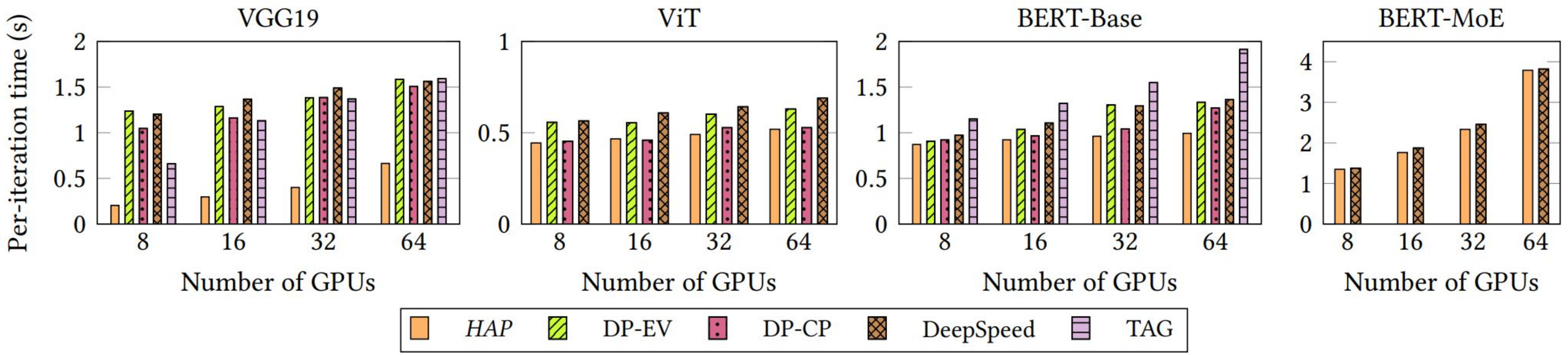
- PyTorch 1.13.1
- 91 semantic rules
- Round sharding dimensions



Experimental Setup

- 8 Machines with 64 GPUs on Alibaba Cloud
  - 6 machines with 8 P100
  - 2 machines with 8 V100 and NVLink
- 4 Models
  - VGG19 with 133M parameters
  - ViT with 54M parameters
  - BERT with 102M parameters
  - BERT-MoE with up to 2.3B parameters
- 4 Baselines
  - DP-EV: Data parallelism with even partitions
  - DP-CP: Data parallelism with sharding ratios proportional to the computation speeds
  - DeepSpeed
  - TAG

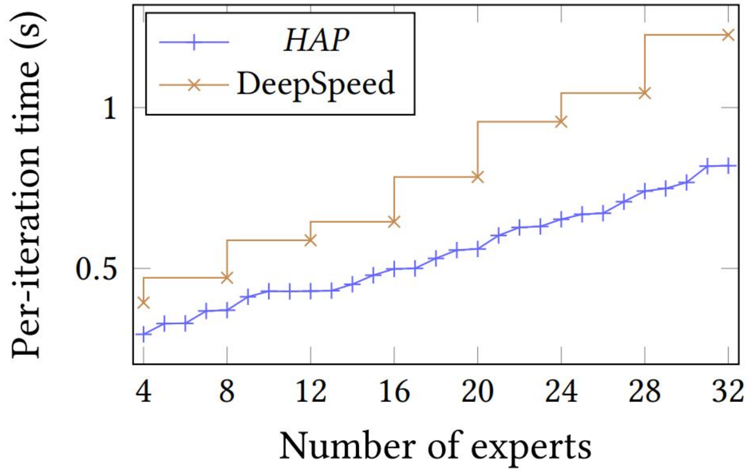
Experiment Result



- Consistently outperform baselines
- Up to 2.41x speed up

Case Study: Uneven Distribution of Experts

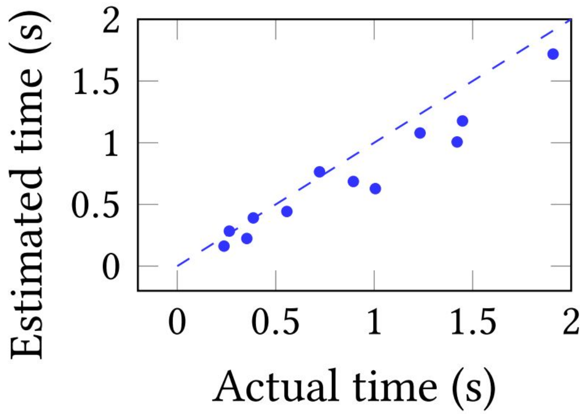
- Most MoE frameworks can only put the same number of experts on all devices.
- HAP places more experts on faster devices to achieve 64% speed up.

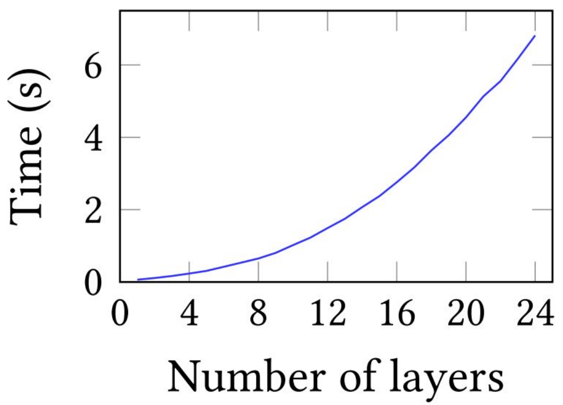


Cost Model Accuracy and Overhead

Training iteration time predicted by our cost model exhibits high linear correlation with actual measurements.

Distributed program can be synthesized within seconds for models of common sizes.





Summary

- HAP enables training larger model by utilizing heterogeneous resources.
- HAP automates distributed training by automated program synthesis.

Future Work

- Support branches and loops.

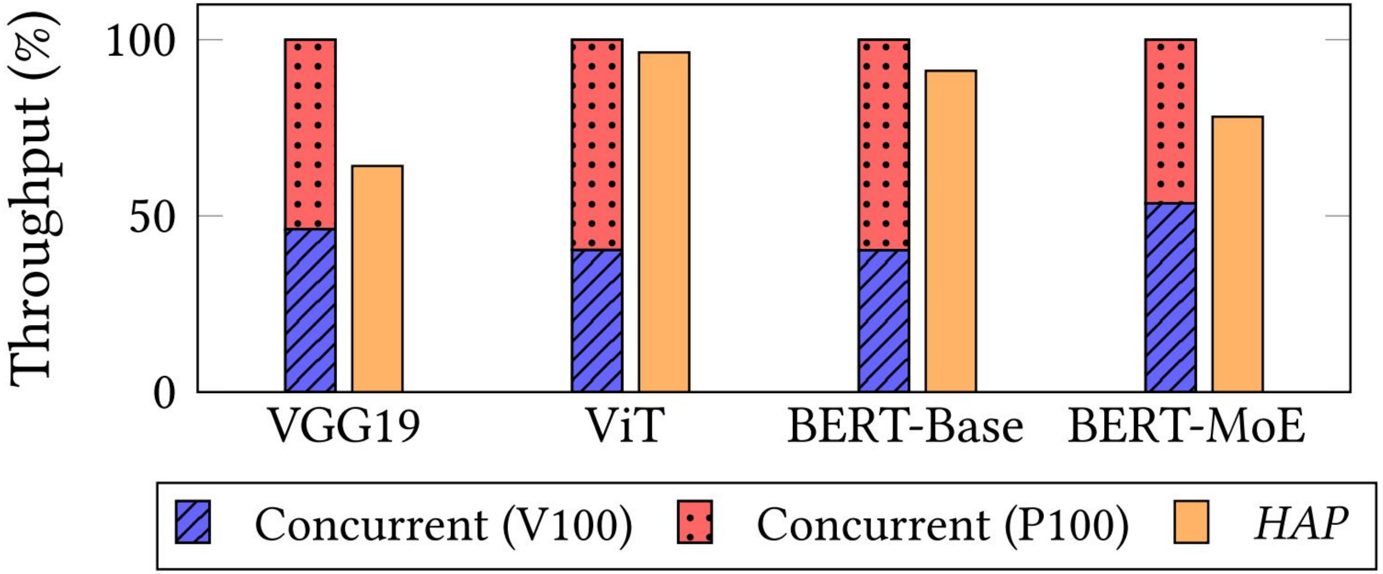
Thank you!

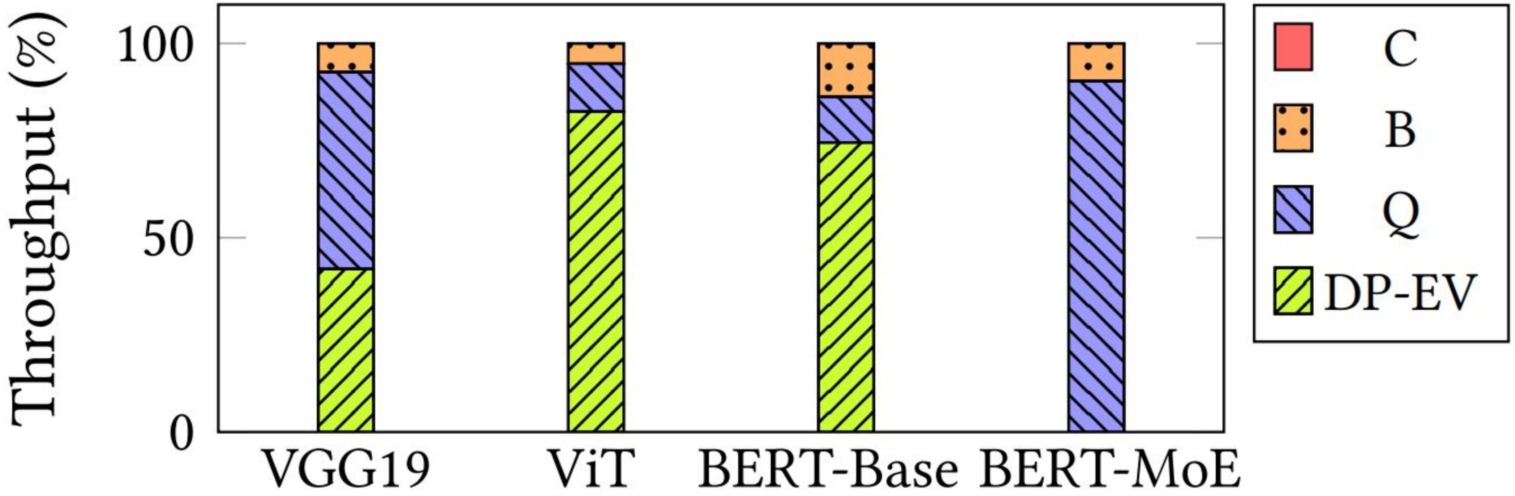
Contact: swzhang@cs.hku.hk

Source Code: https://github.com/alibaba/hap

Artifacts Available V1.1

Artifacts Evaluated Functional V1.1


Throughput (%)
100
50
0
VGG19
ViT
BERT-Base
BERT-MoE
Concurrent (V100)
Concurrent (P100)
HAP


Throughput (%)
100
50
0
VGG19
ViT
BERT-Base
BERT-MoE
C
B
Q
DP-EV