Enabling Parallelism Hot Switching for Efficient Training of Large Language Models

Hao Ge[1], Fangcheng Fu[1], Haoyang Li[1], Xuanyu Wang[1], Sheng Lin[1], Yujie Wang[1], Xiaonan Nie[1], Xupeng Miao[2] and Bin Cui[1]

[1]Peking University, [2]Purdue University

SOSP' 24

Presented by Qinghe Wang

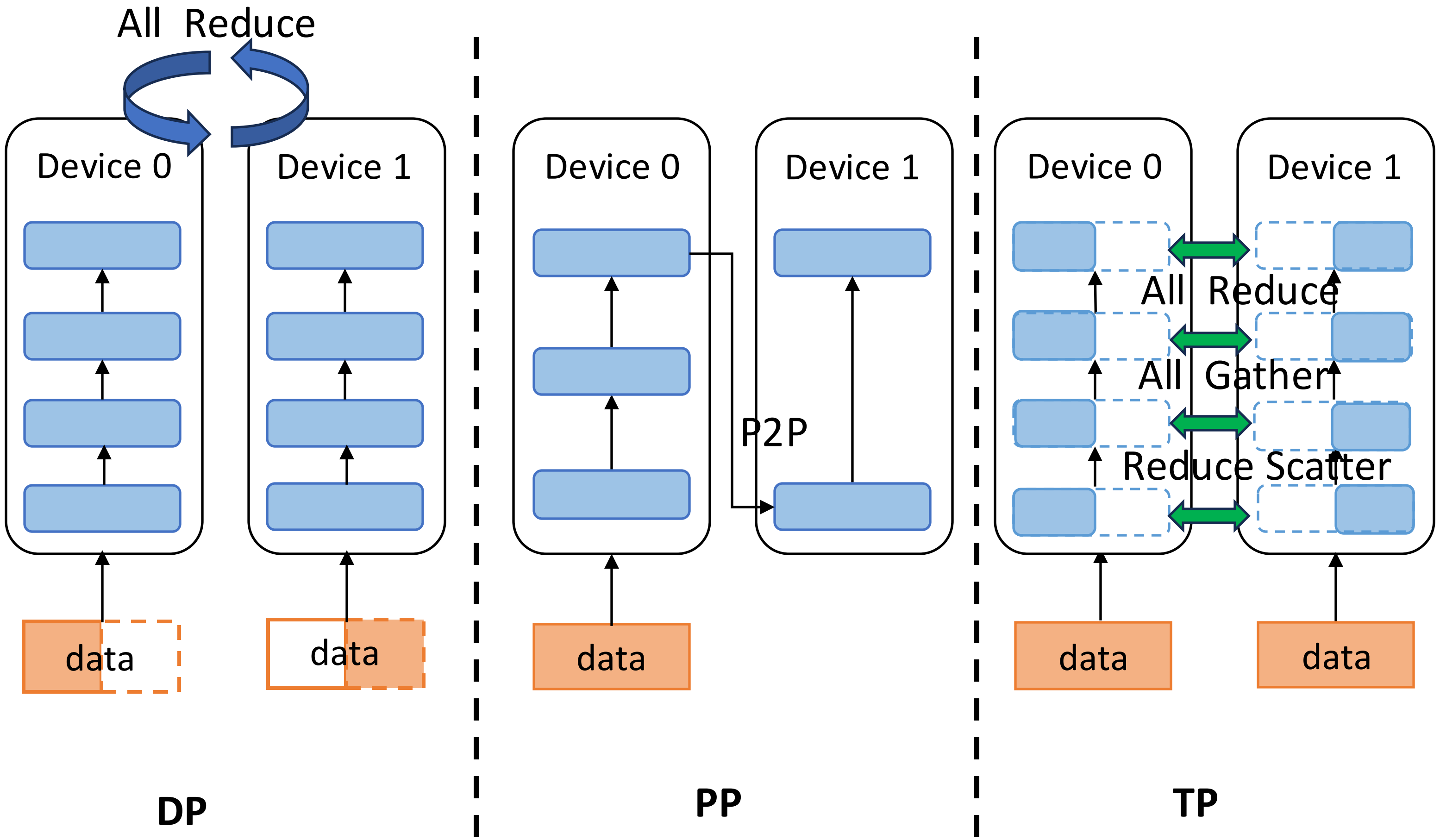
USTC, CHINA
ADSLAB
先进数据系统实验室

Outline

- Background & Motivation
- Design & Implementation
- Evaluation
- Conclusion

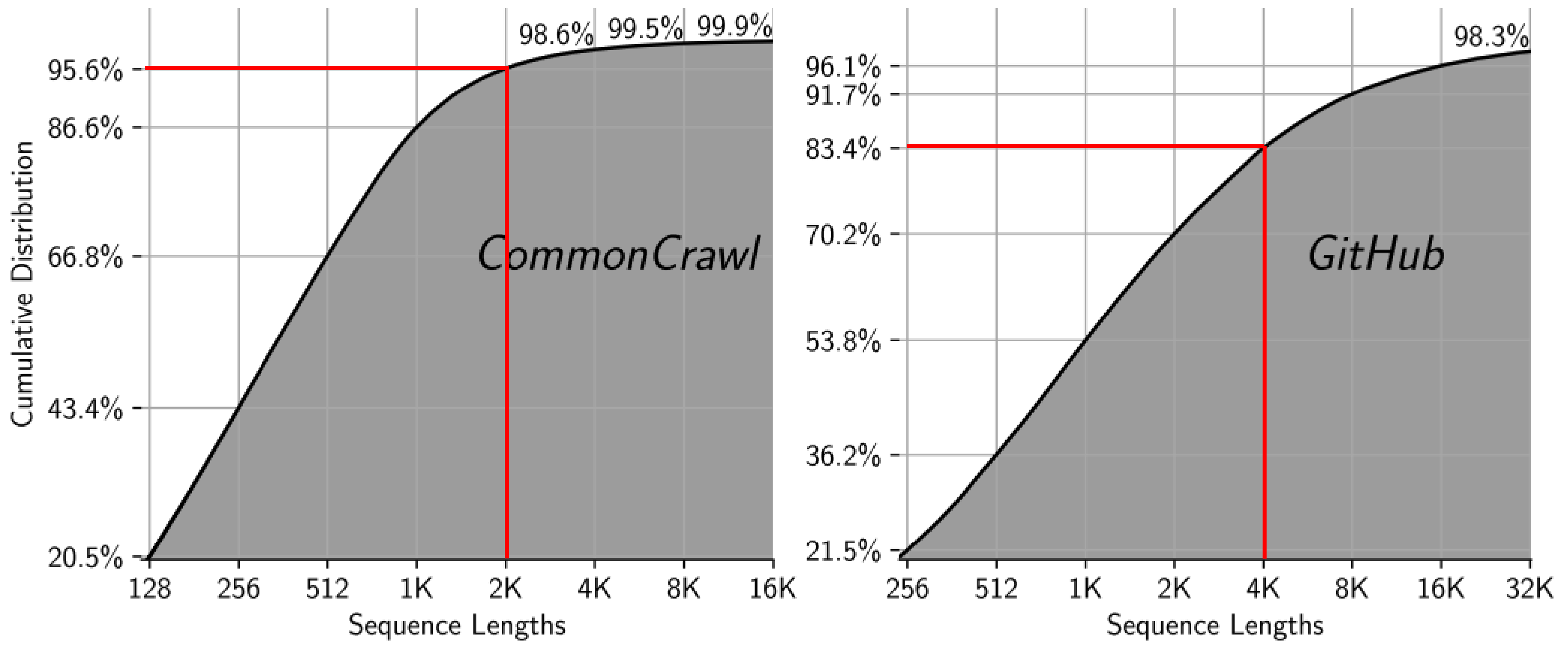
Background

- Parallelism strategies
  - Need to balance between memory consumption and training efficiency



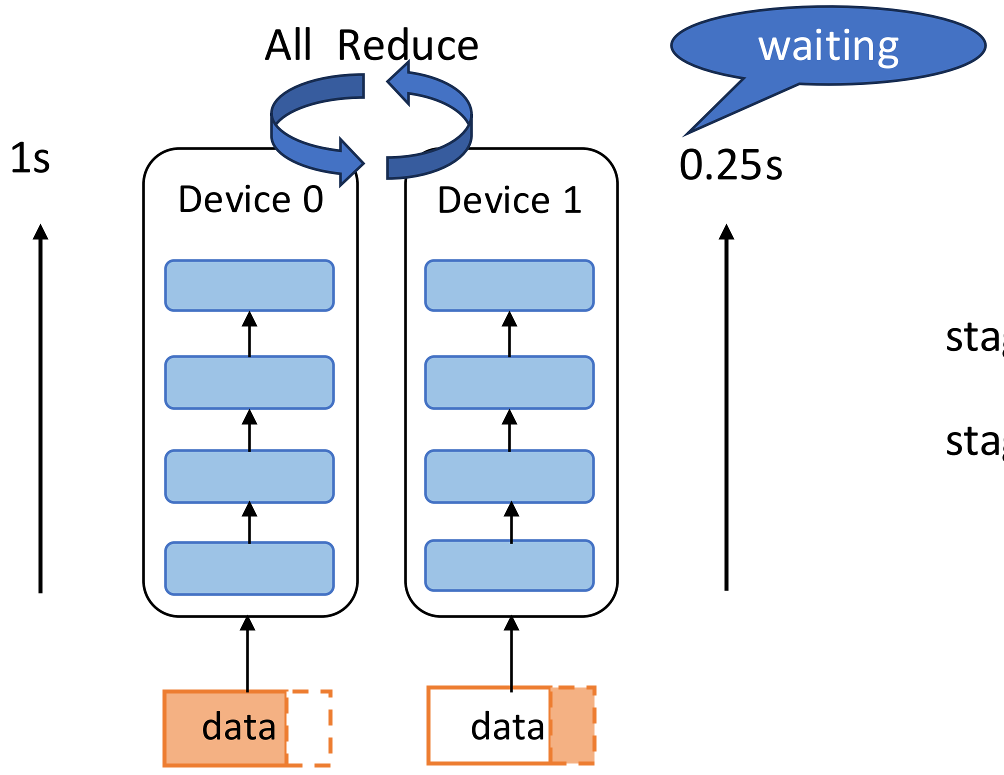
Background

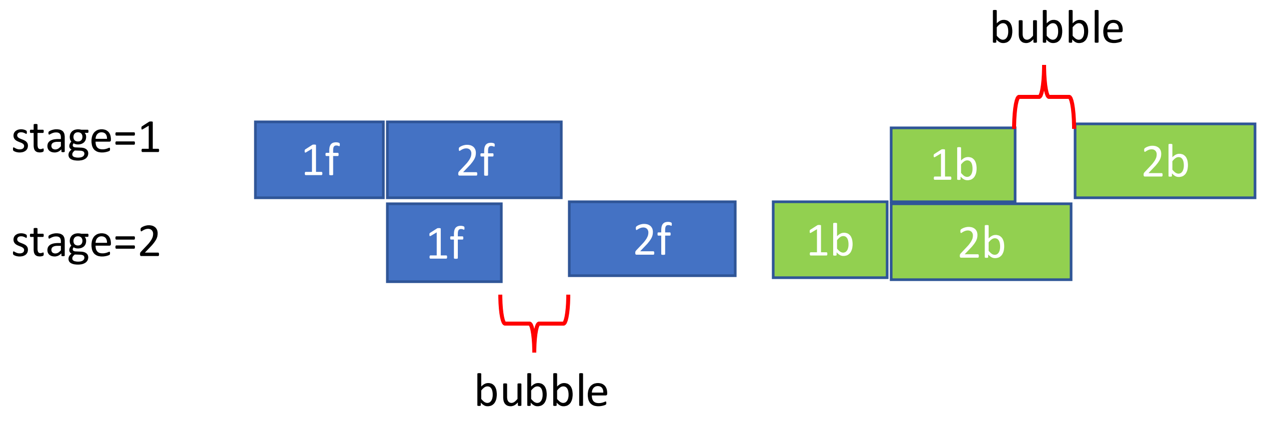
- Skewness in sequence of datasets
  - Results in imbalanced workloads across different sequences



Background

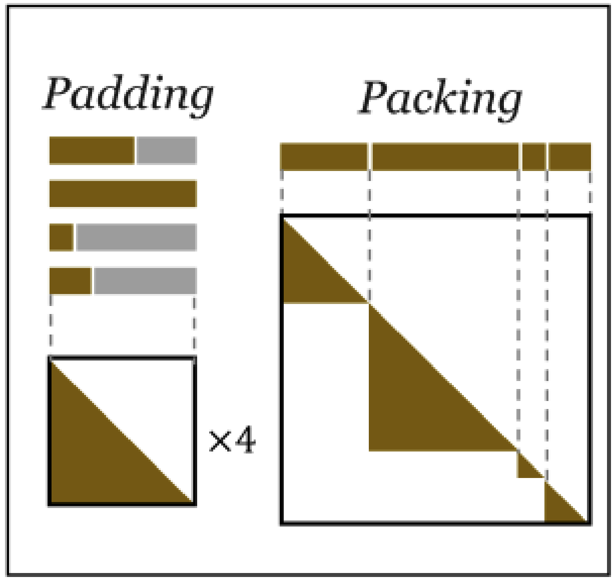
- Impact of load imbalance
  - **Device idle waiting** when using **data parallelism**
  - **Increase the bubble rate** when using **pipeline parallelism**



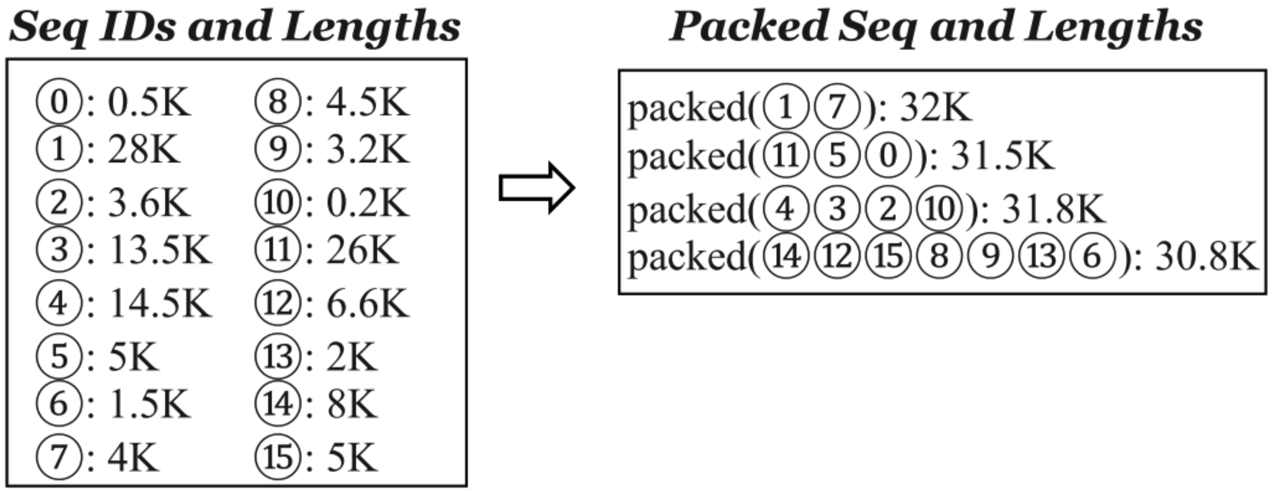


Background

- Padding
  - pad the sequences in the same length
  - waste of computation, communication, memory



- Packing
  - Pack multiple samples of different sequence lengths together and modify the self-attention mask



Observation

- Efficiency Degradation for Longer Context Lengths

Larger TP leads to degraded training efficiency

Seq Len	# Seqs	*TP=1*	*TP=2*	*TP=4*	*TP=8*
1K	512	**13.9**	14.7	16.2	19.5
2K	256	**14.2**	15.1	16.6	19.8
4K	128	**14.9**	15.8	17.4	20.6
8K	64	OOM	**17.4**	19.1	22.1
16K	32	OOM	OOM	**21.8**	25.0
32K	16	OOM	OOM	OOM	**30.8**

Small TP degree leads to long sequence OOM

Runtime (in seconds) of different tensor parallelism degrees (LLaMA2-7B, 8GPUs, DP=8/TP) when processing **the same amount of tokens**

Different sequence lengths require different parallel strategies ！

Background

- Existing systems (Megatron/Deepspeed/Alpa)
  - Ignore the workload imbalance in LLM training
  - Leverage a **static** parallelism strategy
    - Using **memory-saving** strategy for all samples to avoid OOM
    - Memory-saving strategy reduce the training efficiency of **short samples**

Motivation

- Can we adopt **separate parallelism strategies** for sequences with different levels of workloads/lengths?

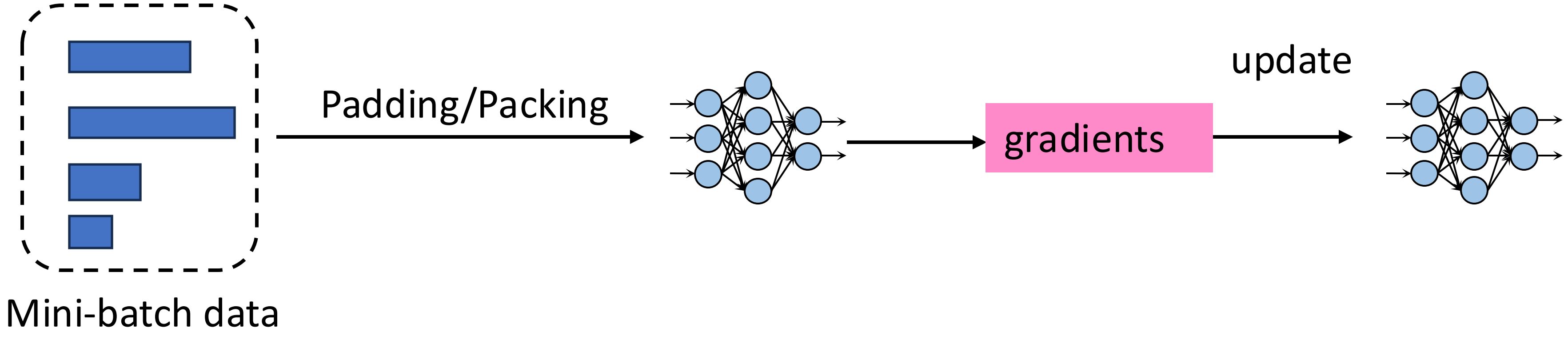
- Challenges
  - How to design parallel strategies for samples of different sequence lengths？
  - How to switch between different parallelism strategies ?

Outline

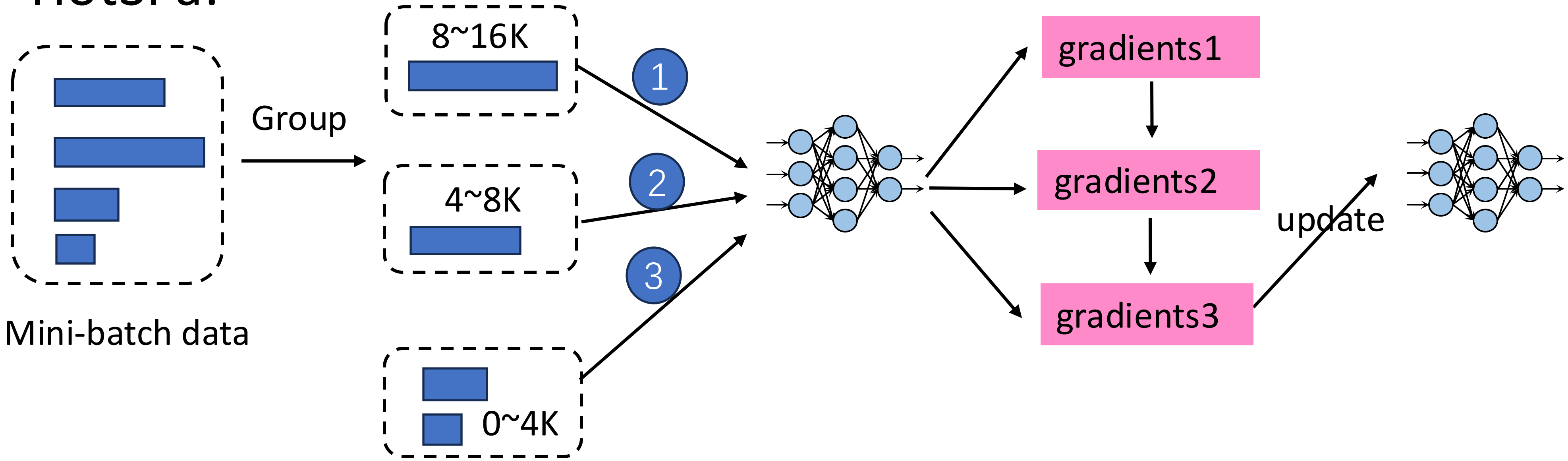
- Background & Motivation
- Design & Implementation of HotSPa
- Evaluation
- Conclusion

Key ideas of HotSPa

Original:

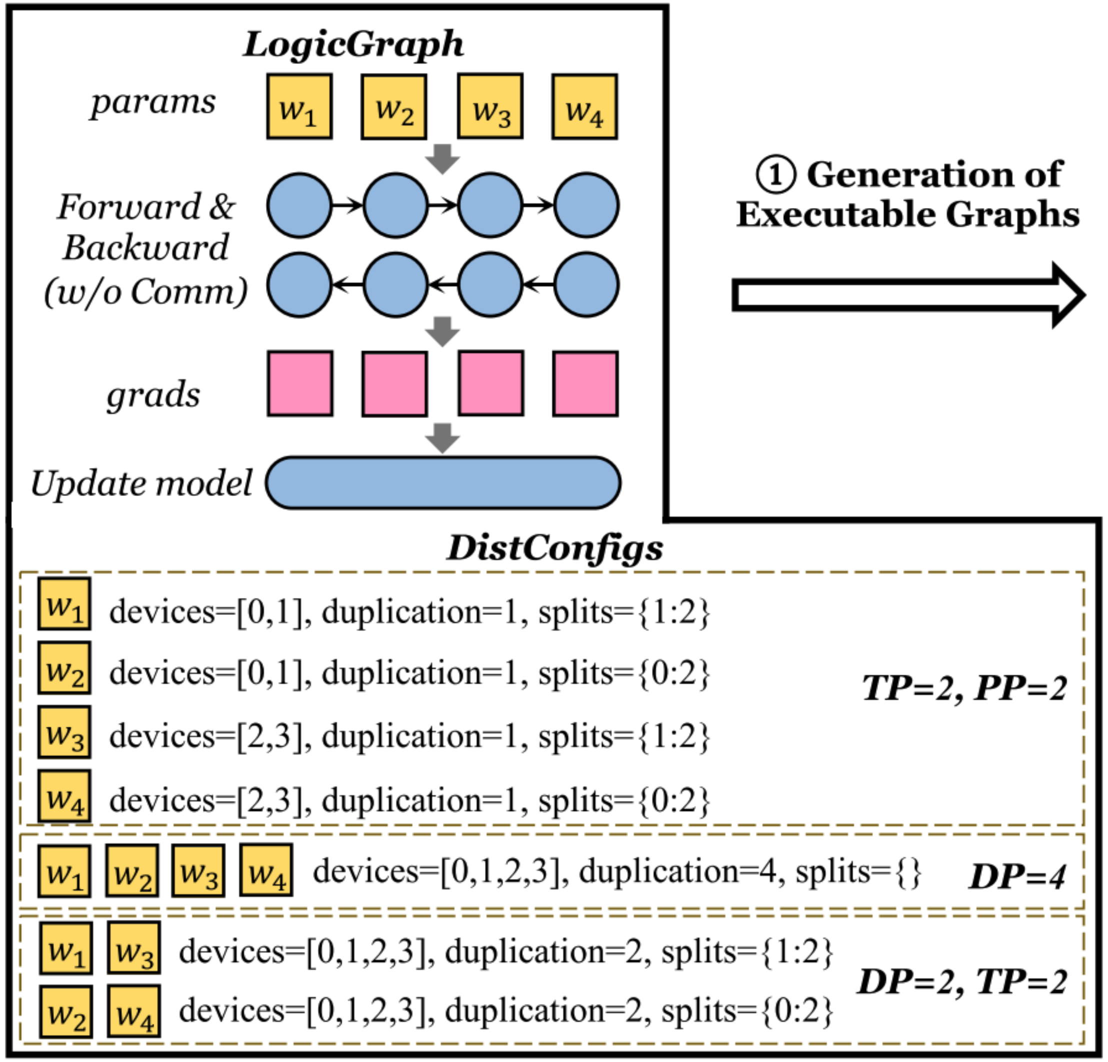


HotSPa:



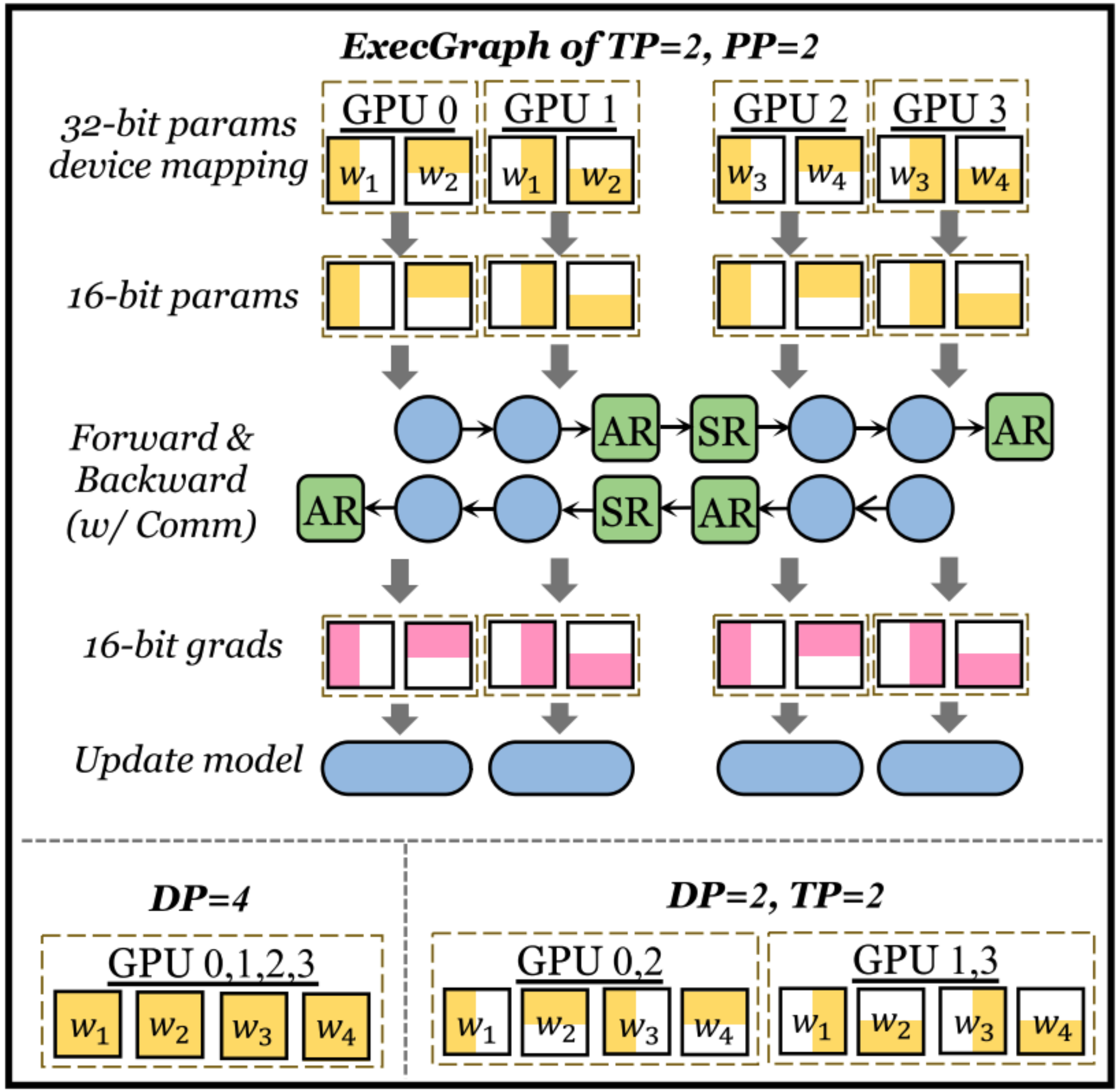
Design of HotSPa

- Logical Graph
  - Only includes calculation operators and data dependencies
- DistConfig
  - A unified representation for distributed strategies
    - device: the assigned devices
    - duplication: DP degree
    - spilt: how a multi-dimensional parameter is split among devices



Design of HotSPa

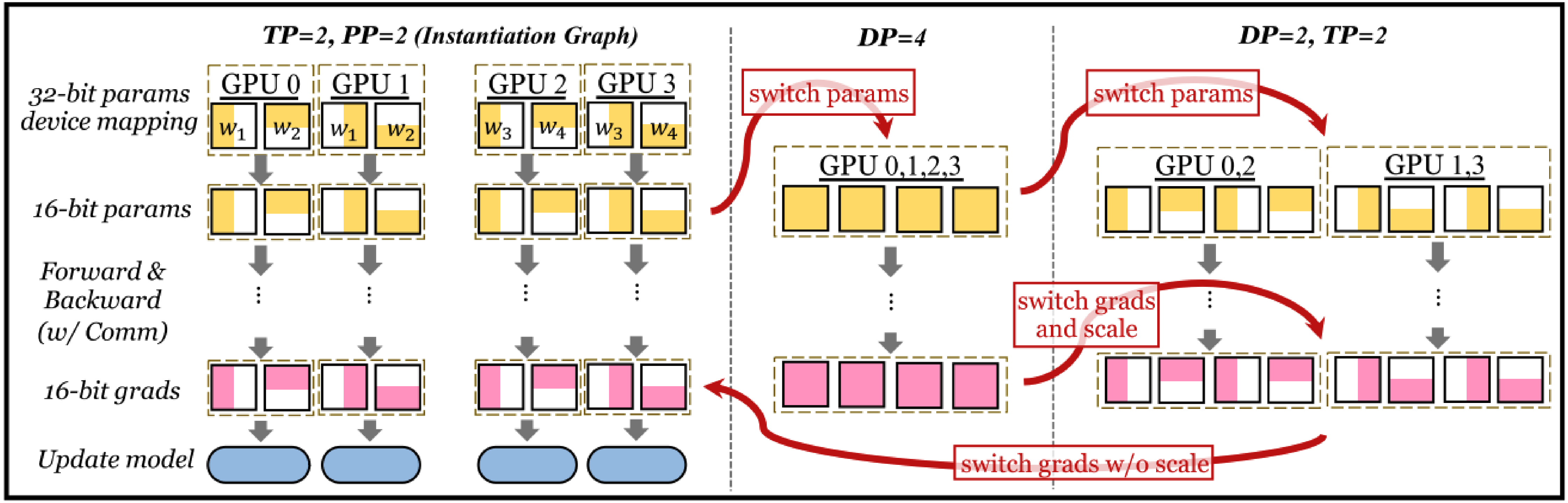
- Executable Graph
  - The computational graph bound to the specific strategy
  - Inserted three type of operators
    - Communication operators
    - Type casting operators
    - Accumulation operators
- Mixed-precision Training
  - FP16 for forward and backward
  - FP32 for weight update and gradient accumulation



② Orchestration of Executable Graphs

Design of HotSPa

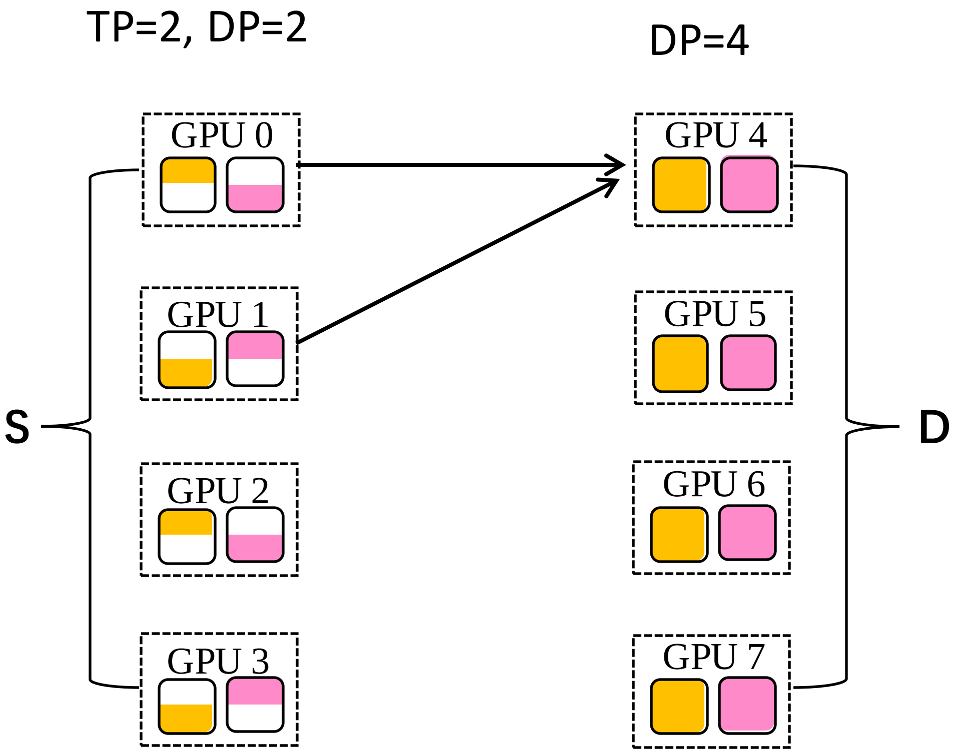
- Orchestration of Executable Graphs
  - Identify the suitable candidate ExecGraphs for **instantiation**
  - ExecGraphs are **re-ordered** for better efficiency
  - Actually start running each parallel plan and perform **hot switch**



Hot switching planner

- Heuristic hot switching planner
  - Intra-node communication is preferable than inter-node
  - GPU connectivities are full-duplex

$V_0^{(inter)}$	$V_0^{(intra)}$
0	0
0	0
0	0
0	0



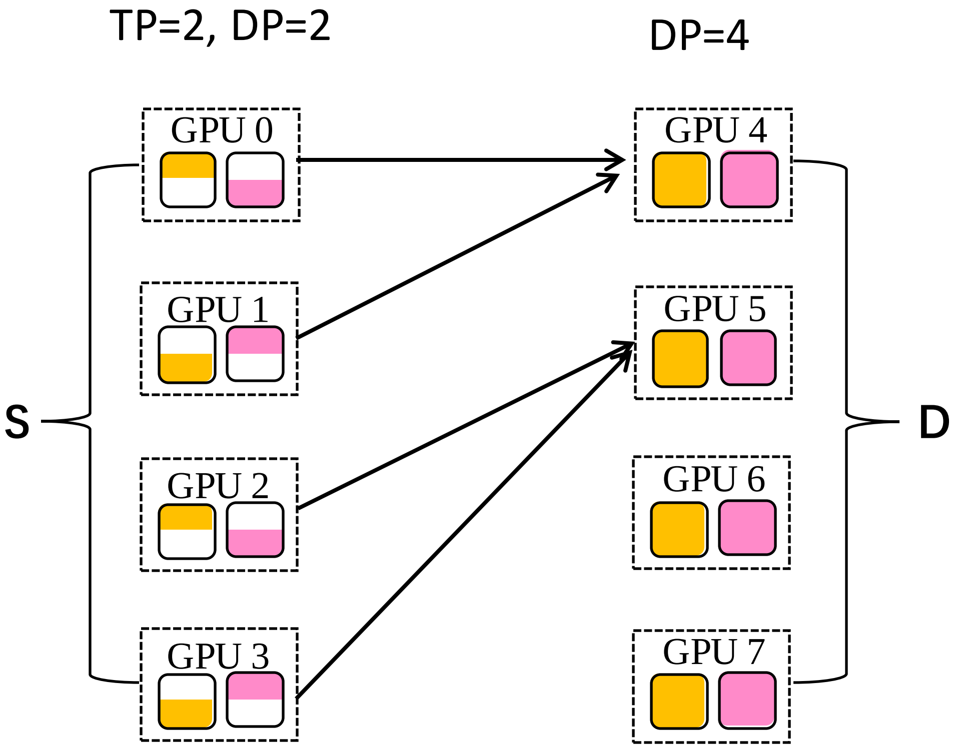
Algorithm 1: Our heuristic hot switching planner.

Initialize hot switching plan $\mathcal{P} = \{\}$;
Initialize intra- and inter-node communication volume $V_i^{(inter)}, V_i^{(intra)}$ as 0 for each device i;
foreach *model parameter/gradient slice* **do**
  Determine the owner (source) devices S;
  Determine the target (destination) devices D;
  foreach *dst in D* **do**
    if *dst* $\notin S$ **then**
      Partition S into $S^{(intra)}, S^{(inter)}$;
      if $S_i^{(intra)}$ *is not empty* **then**
        $src \leftarrow \arg\min_i \{V_i^{(intra)} | i \in S^{(intra)}\}$;
        $V_{src}^{(intra)} \leftarrow V_{src}^{(intra)} + \text{sizeof}(slice)$;
      else
        $src \leftarrow \arg\min_i \{V_i^{(inter)} | i \in S^{(inter)}\}$;
        $V_{src}^{(inter)} \leftarrow V_{src}^{(inter)} + \text{sizeof}(slice)$;
      $\mathcal{P} \leftarrow \mathcal{P} \cup (slice, src, dst)$;
return $\mathcal{P}$;

Hot switching planner

- Heuristic hot switching planner
  - Intra-node communication is preferable than inter-node
  - GPU connectivities are full-duplex

$V_0^{(inter)}$	$V_0^{(intra)}$
0	1
0	1
0	0
0	0



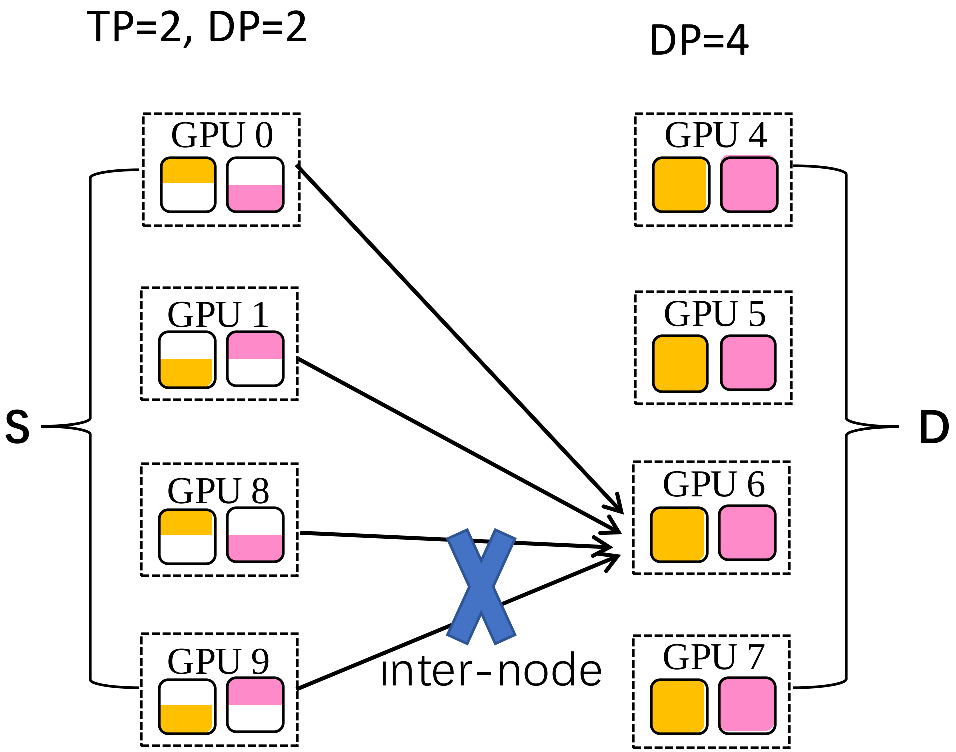
Algorithm 1: Our heuristic hot switching planner.

Initialize hot switching plan $\mathcal{P} = \{\}$;
Initialize intra- and inter-node communication volume $V_i^{(inter)}, V_i^{(intra)}$ as 0 for each device i;
foreach *model parameter/gradient slice* **do**
  Determine the owner (source) devices S;
  Determine the target (destination) devices D;
  foreach dst *in* D **do**
    if $dst \notin S$ **then**
      Partition S into $S^{(intra)}, S^{(inter)}$;
      if $S_i^{(intra)}$ *is not empty* **then**
        $src \leftarrow \arg\min_i \{V_i^{(intra)} | i \in S^{(intra)}\}$;
        $V_{src}^{(intra)} \leftarrow V_{src}^{(intra)} + \text{sizeof}(slice)$;
      else
        $src \leftarrow \arg\min_i \{V_i^{(inter)} | i \in S^{(inter)}\}$;
        $V_{src}^{(inter)} \leftarrow V_{src}^{(inter)} + \text{sizeof}(slice)$;
      $\mathcal{P} \leftarrow \mathcal{P} \cup (slice, src, dst)$;
return $\mathcal{P}$;

Hot switching planner

- Heuristic hot switching planner
  - Intra-node communication is preferable than inter-node
  - GPU connectivities are full-duplex

$V_0^{(inter)}$	$V_0^{(intra)}$
0	1
0	1
0	0
0	0



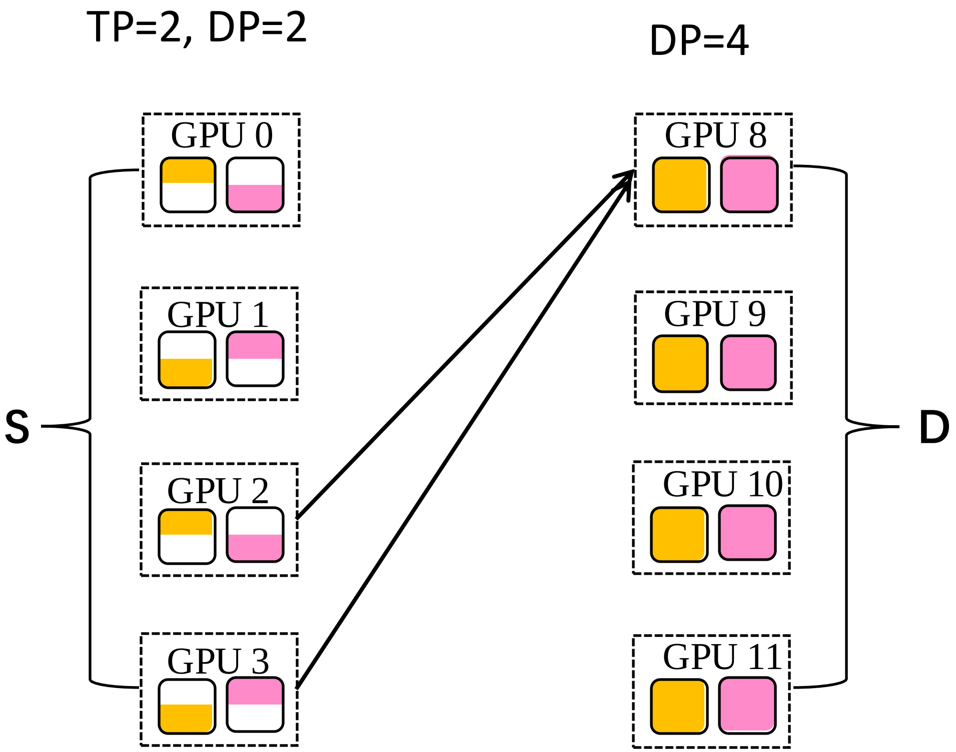
Algorithm 1: Our heuristic hot switching planner.

Initialize hot switching plan $\mathcal{P} = \{\}$;
Initialize intra- and inter-node communication volume $V_i^{(inter)}, V_i^{(intra)}$ as 0 for each device i;
foreach *model parameter/gradient slice* **do**
  Determine the owner (source) devices S;
  Determine the target (destination) devices D;
  foreach dst *in* D **do**
    if $dst \notin S$ **then**
      Partition S into $S^{(intra)}, S^{(inter)}$;
      if $S_i^{(intra)}$ *is not empty* **then**
        $src \leftarrow \arg\min_i \{V_i^{(intra)} \mid i \in S^{(intra)}\}$;
        $V_{src}^{(intra)} \leftarrow V_{src}^{(intra)} + \text{sizeof}(slice)$;
      else
        $src \leftarrow \arg\min_i \{V_i^{(inter)} \mid i \in S^{(inter)}\}$;
        $V_{src}^{(inter)} \leftarrow V_{src}^{(inter)} + \text{sizeof}(slice)$;
      $\mathcal{P} \leftarrow \mathcal{P} \cup (slice, src, dst)$;
return $\mathcal{P}$;

Hot switching planner

- Heuristic hot switching planner
  - Intra-node communication is preferable than inter-node
  - GPU connectivities are full-duplex

$V_0^{(inter)}$	$V_0^{(intra)}$
1	0
1	0
0	0
0	0

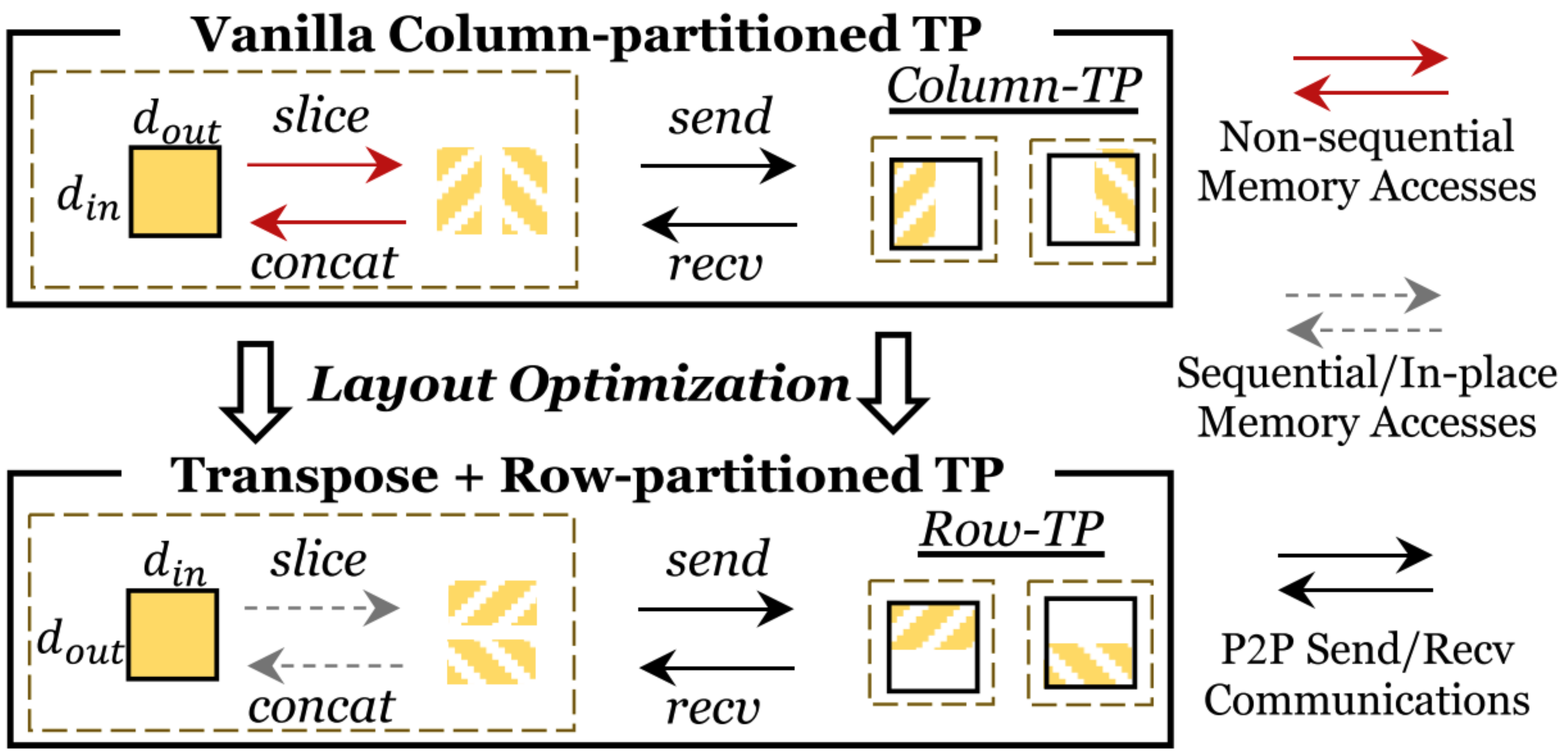


Algorithm 1: Our heuristic hot switching planner.

Initialize hot switching plan $\mathcal{P} = \{\}$;
Initialize intra- and inter-node communication volume $V_i^{(inter)}, V_i^{(intra)}$ as 0 for each device i;
foreach *model parameter/gradient slice* **do**
  Determine the owner (source) devices S;
  Determine the target (destination) devices D;
  foreach dst *in* D **do**
    if $dst \notin S$ **then**
      Partition S into $S^{(intra)}, S^{(inter)}$;
      if $S_i^{(intra)}$ *is not empty* **then**
        $src \leftarrow \arg\min_i \{V_i^{(intra)} | i \in S^{(intra)}\}$;
        $V_{src}^{(intra)} \leftarrow V_{src}^{(intra)} + \text{sizeof}(slice)$;
      else
        $src \leftarrow \arg\min_i \{V_i^{(inter)} | i \in S^{(inter)}\}$;
        $V_{src}^{(inter)} \leftarrow V_{src}^{(inter)} + \text{sizeof}(slice)$;
      $\mathcal{P} \leftarrow \mathcal{P} \cup (slice, src, dst)$;
return $\mathcal{P}$;

Layout optimization

- The problem
  - Column-partitioning degree change causes **non-sequential memory accesses**
- Layout optimization
  - **Flip the layout argument** as column-major ordered and do **row-partitioned** TP



Implementation

- Build on top of **Hetu**
  - AI framework developed by a Peking University team
  - Automatic parallel strategy
    - Supports 3D parallelism
  - Communication optimization
  - Memory management

- Performance of parallel strategy
  - Similar performance against Megatron-LM

Outline

- Background & Motivation
- Design & Implementation
- Evaluation
- Conclusion

Evaluation

- Experimental Setup
- End-to-end Evaluation
- Case Studies
- Overhead of Hot Switching
- Ablation Studies of Hot Switching

Evaluation Setup

- Environments
  - 8~32 NVIDIA **A800-80GB** GPUs
  - NVLink with a bandwidth of **400GB/s** in servers
  - IB with a bandwidth of **200GB/s** between servers
- Baselines
  - Megatron
  - Deepspeed
- Models

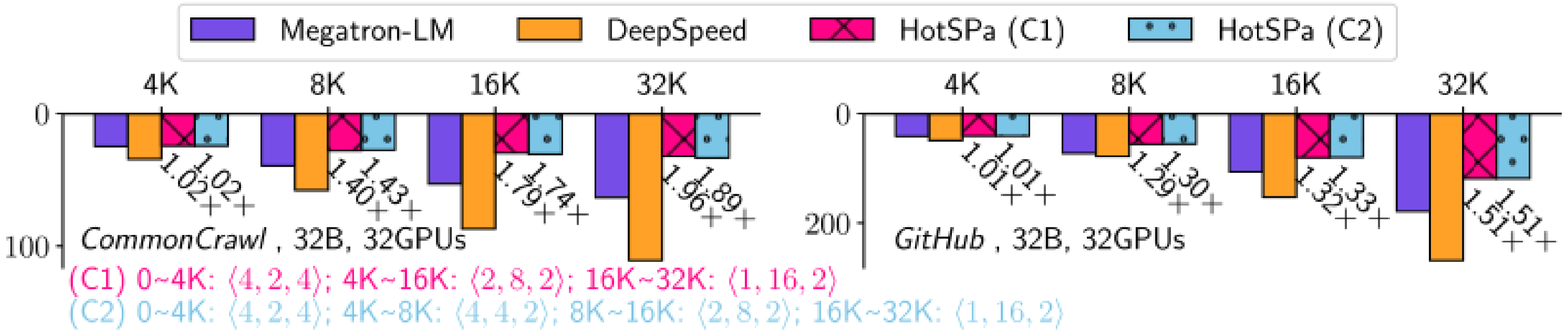
Model	Layer	Heads	Hidden Dim
LLaMA2-7B	32	32	4096
LLaMA2-13B	40	40	5120
LLaMA2-32B	60	64	6656

Evaluation Setup

- Datasets
  - CommonCrawl
  - Github
  - Changed the sequence length from **4K to 32K**
  - Batchsize is set to **512** (Large gradient accumulation steps)

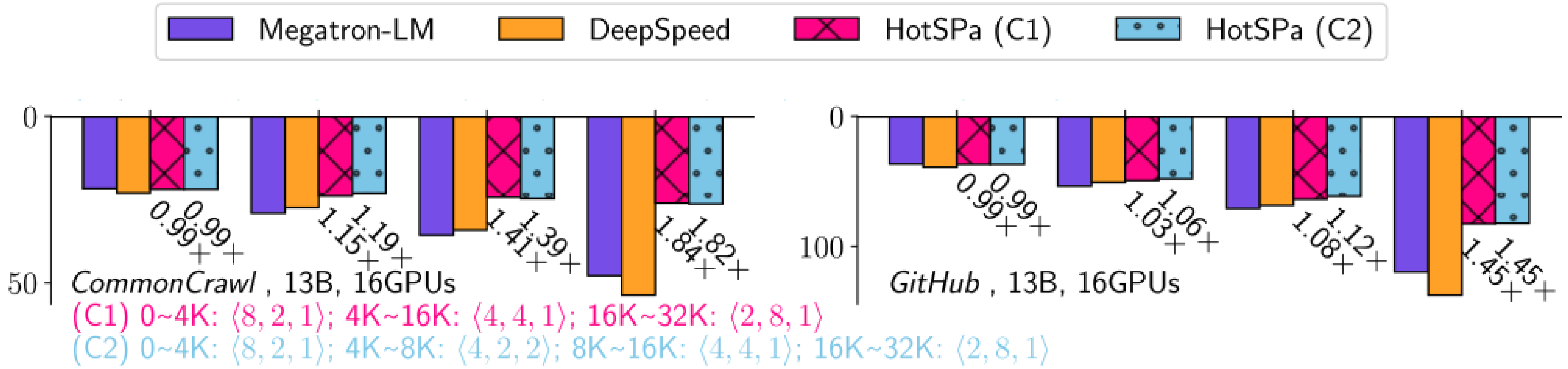
End-to-end Evaluation

- LLama-32B, 32GPUs
  - 4K: static parallelism strategies, similar performance
  - 8K~32K: HotSPa achieves up to **1.96x** speed up



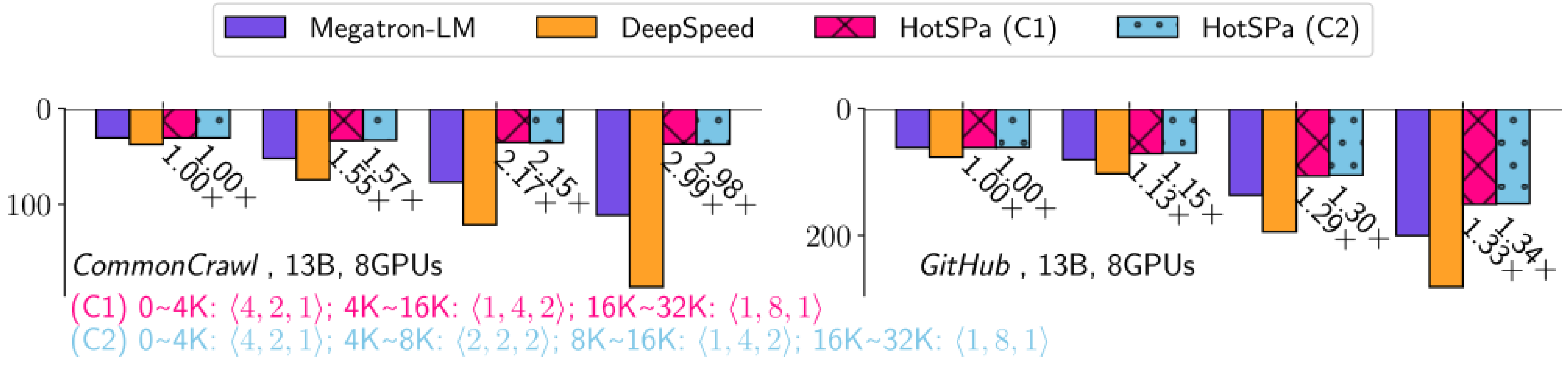
End-to-end Evaluation

- LLama-13B, 16GPUs
  - 4K: static parallelism strategies, similar performance
  - 8K~32K: HotSPa achieves up to **1.84x** speed up



End-to-end Evaluation

- LLama-13B, 8GPUs
  - ◆ 4K: static parallelism strategies, similar performance
  - ◆ 8K~32K: HotSPa achieves up to **2.99x** speed up



Case studies

- LLama-32B, 32GPUs, s=32K
  - The benefits are most evident on **short sequence lengths**
  - Hot switching **incurs little overhead**

Breakdown	*CommonCrawl* (time in seconds)				*GitHub* (time in seconds)			
	Static	C1	C2	C3	Static	C1	C2	C3
0~1K	42.1	22.7 (2.40×)	22.7 (2.40×)	12.8 (3.28×)	34.9	26.3 (2.78×)	26.3 (2.78×)	8.6 (4.05×)
1K~4K	12.5			8.2 (1.52×)	38.4			17.1 (2.24×)
4K~8K	3.5	5.4 (1.18×)	2.7 (1.29×)	2.7 (1.29×)	25.0	37.4 (1.40×)	15.4 (1.62×)	15.4 (1.62×)
8K~16K	2.9		2.2 (1.31×)	2.2 (1.31×)	27.3		20.2 (1.35×)	20.2 (1.35×)
16K~32K	2.4	2.4 (1.00×)	2.4 (1.00×)	2.4 (1.00×)	53.0	53.0 (1.00×)	53.0 (1.00×)	53.0 (1.00×)
Others	-	1.8	3.5	4.9	-	1.8	3.5	4.9
Total	63.4	32.3 (1.96×)	33.5 (1.89×)	33.2 (1.90×)	178.6	118.5 (1.50×)	118.4 (1.50×)	119.2 (1.49×)

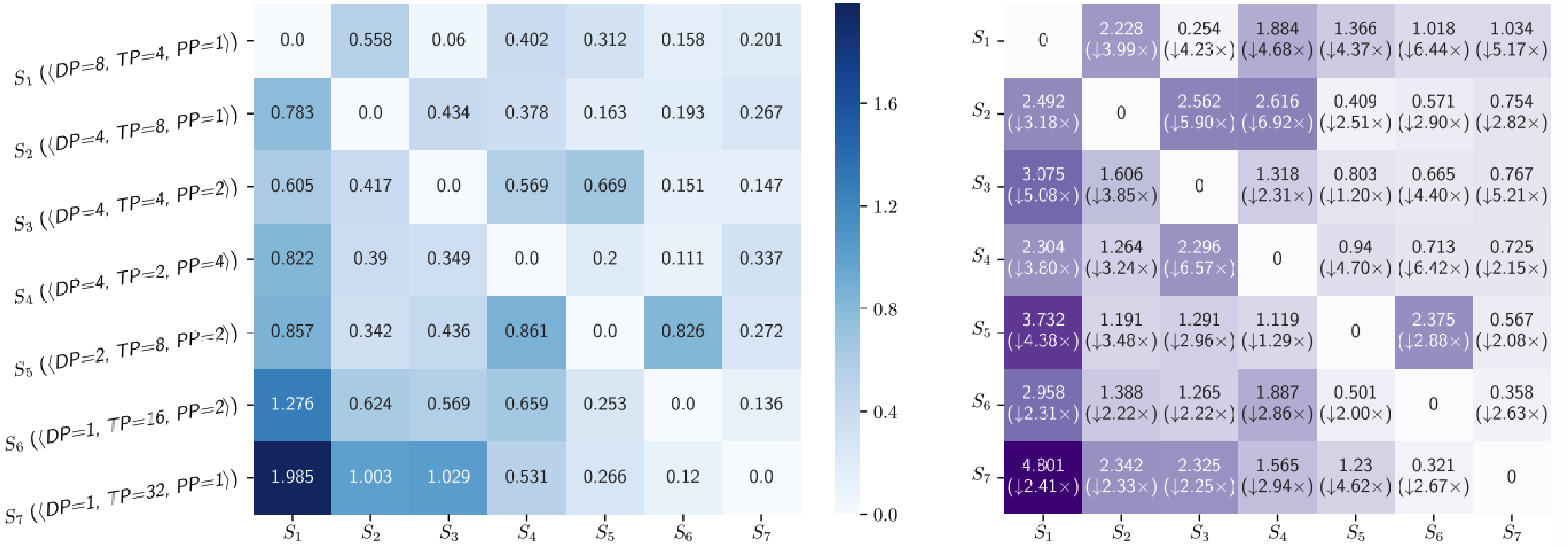
(Static) 0~32K: ⟨1, 16, 2⟩ (C1) 0~4K: ⟨4, 2, 4⟩; 4K~16K: ⟨2, 8, 2⟩; 16K~32K: ⟨1, 16, 2⟩

(C2) 0~4K: ⟨4, 2, 4⟩; 4K~8K: ⟨4, 4, 2⟩; 8K~16K: ⟨2, 8, 2⟩; 16K~32K: ⟨1, 16, 2⟩

(C3) 0~1K: ⟨8, 4, 1⟩; 1K~4K: ⟨4, 2, 4⟩; 4K~8K: ⟨4, 4, 2⟩; 8K~16K: ⟨2, 8, 2⟩; 16K~32K: ⟨1, 16, 2⟩

Overhead of Hot Switching

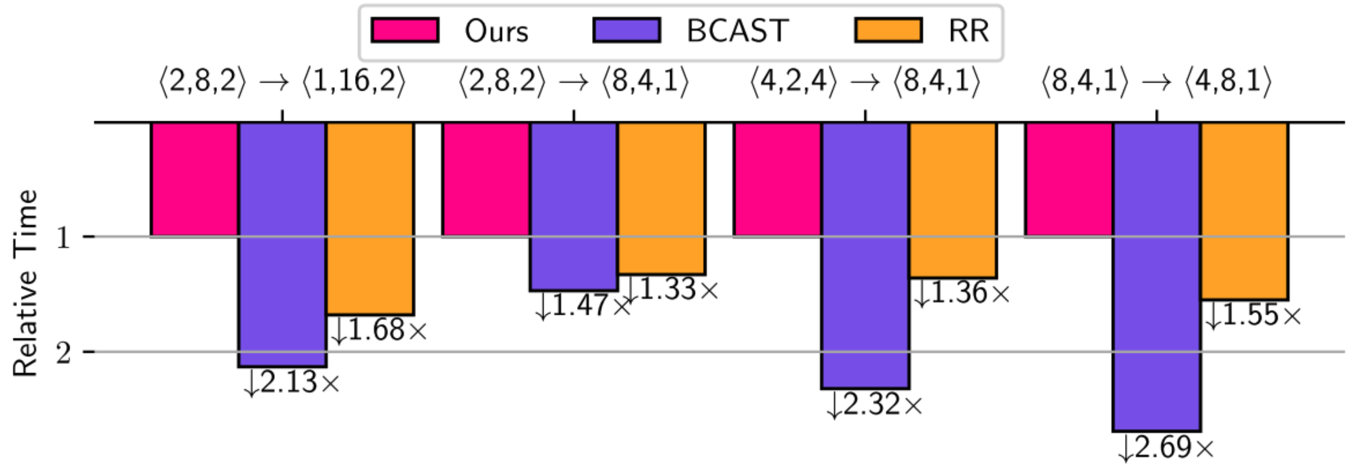
- LLama-32B, 32GPUs, s=32K
  - Hot switching incurs less overhead after optimization



Time cost (in seconds) of switching between parallelism strategies

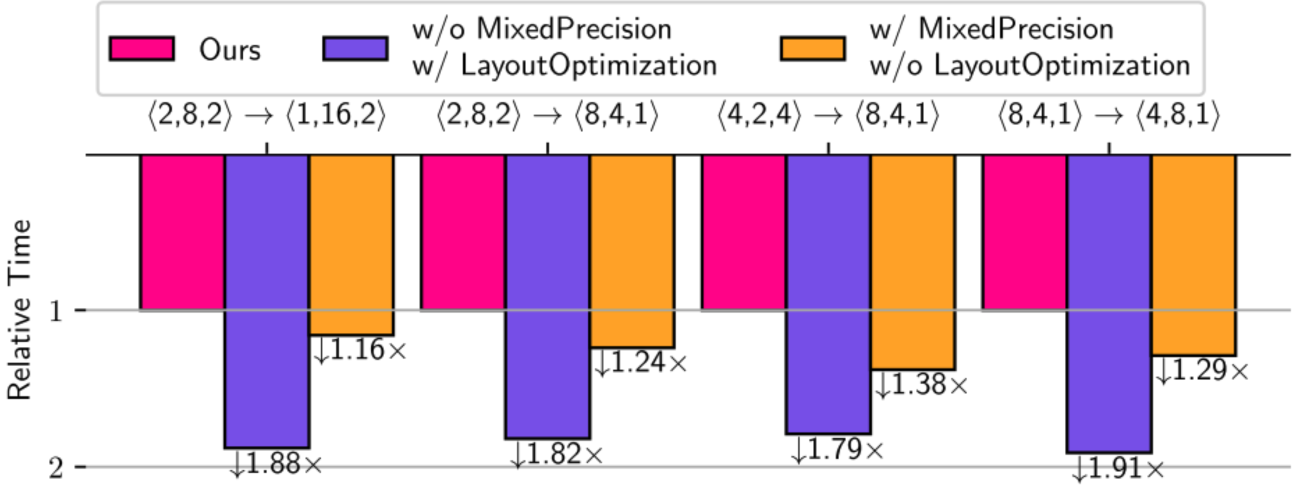
Ablation Studies of Hot Switching

- LLama-32B, 32GPUs, s=32K
  - Baseline
    - BCAST: a naïve algorithm that **ignores workload balance**
    - RR: focus on workload balance but **ignores the bandwidth differences**
  - Results
    - Improving by 1.33-1.68x and 1.47-2.69x compared to BCAST and RR



Ablation Studies of Hot Switching

- LLama-32B, 32GPUs, s=32K
  - Other Optimization techniques
    - Mixed precision: communicates only the 16-bit values
    - Layout optimization: designed for column-partitioning TP
  - Results
    - Mixed precision saves **79%-90%** of hot switching cost
    - Layout optimization contributes **1.16-1.38x** of speedup



Outline

- Background & Motivation
- Design & Implementation
- Evaluation
- Conclusion

Conclusion

- Strength
  - Points out that **static** strategies can lead to **reduced training efficiency**
  - Proposed a novel distributed training paradigm of parallelism **hot switching**
  - Developed a system(HotSPa) that supports parallelism hot switching

- Weakness
  - Reliance on **user-provided** combinations of strategies
    - Group partitioning influences **the number of sequences** in each group
    - **High time cost** of searching optimal parallelism strategies
  - Hot switching is **not worthwhile sometimes**
    - More devices and less gradient accumulation steps