USENIX ATC '24

Accelerating the Training of Large Language Models using Efficient Activation Rematerialization and Optimal Hybrid Parallelism

Tailing Yuan, Yuliang Liu*, Xucheng Ye, Shenglong Zhang, Jianchao Tan, Bin Chen, Chengru Song, and Di Zhang

Kuaishou Technology, Beijing, China

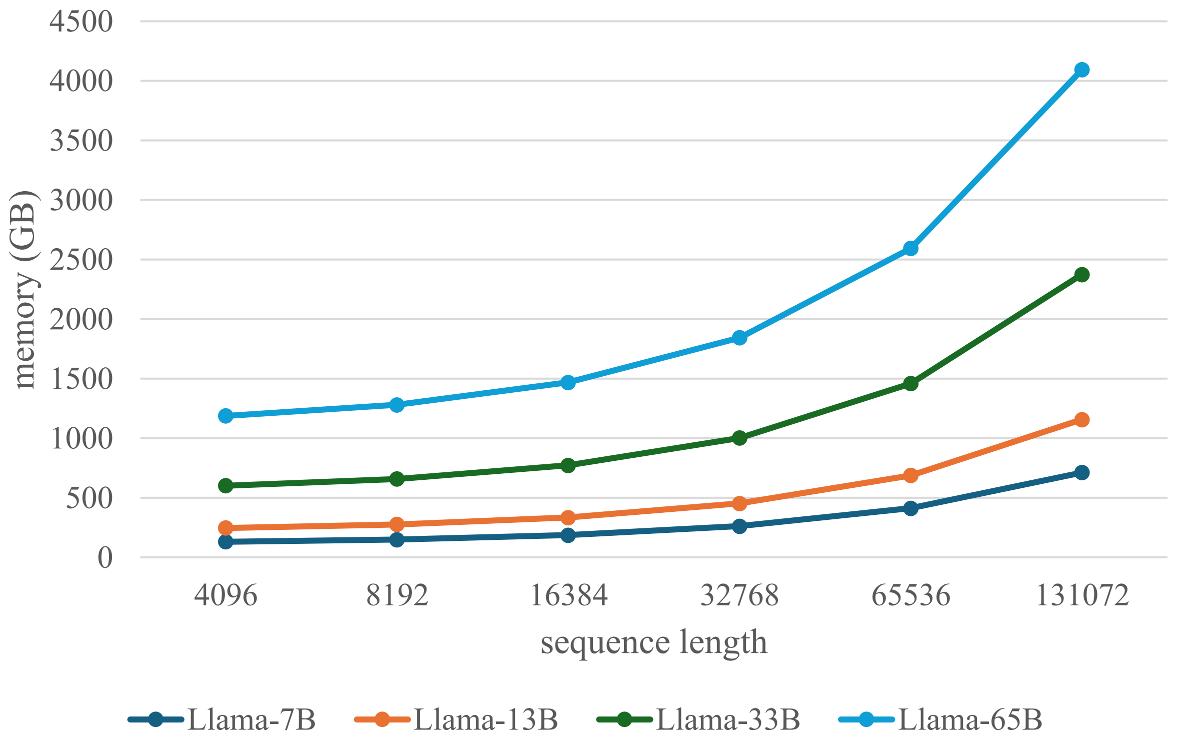
Presenter: yuantailing@kuaishou.com
*Corresponding author: liuyuliang@kuaishou.com

Outline

1. Background
2. Methods
   - Compute-Memory Balanced Checkpointing
   - Pipeline-Parallel-Aware Offloading
   - Hybrid Parallel Parameters Tuning
3. Evaluation
4. Contribution

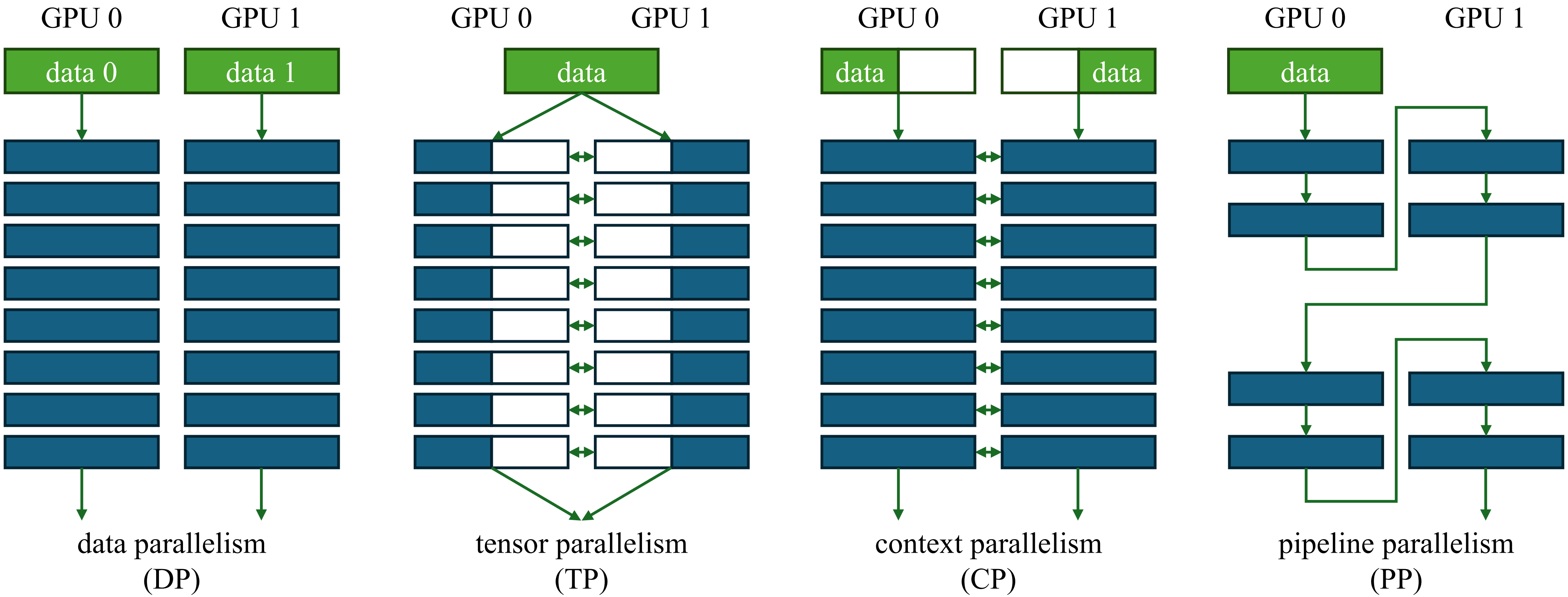
1. Background

- Memory size serves as one of the most significant challenges in LLMs' training



1. Background

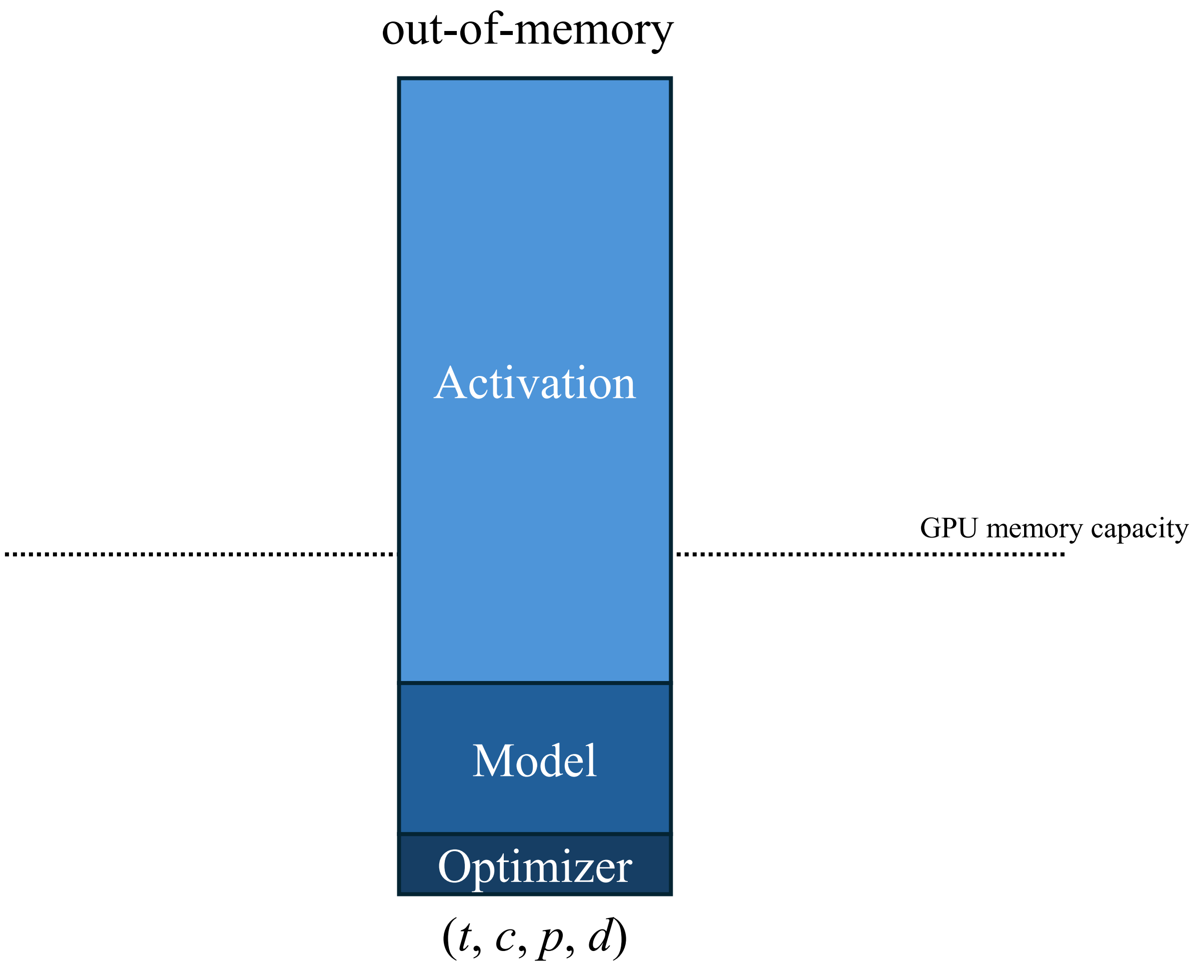
- Distribute data and model to multiple GPUs using hybrid parallelism



1. Background

- Challenge: The activation size grows along with sequence length, and may exceed GPU memory capacity

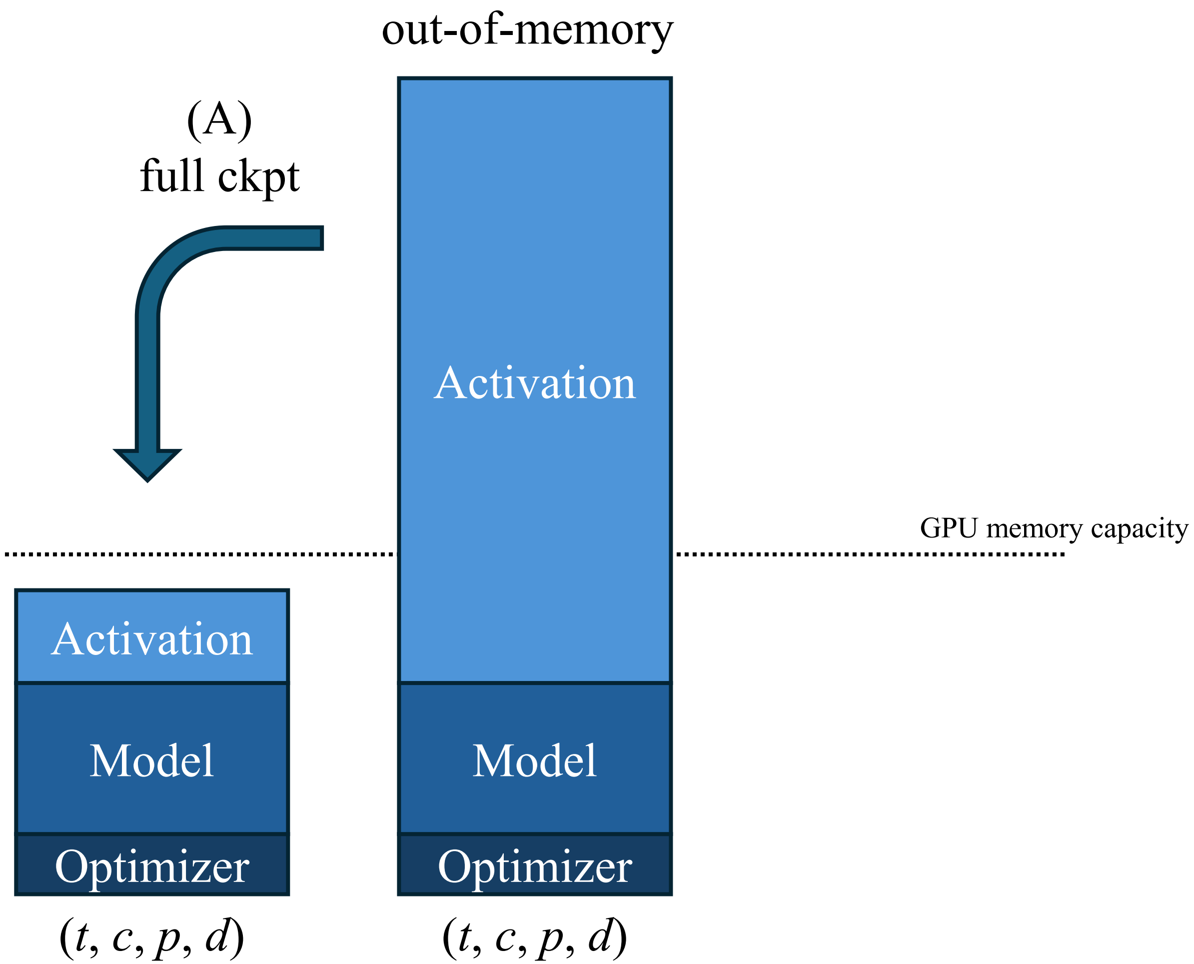
- Traditional solutions
  - A. Full checkpointing
    - Leads to 1/3 additional computation cost
  - B. Increasing TP size and/or CP size
    - Incurs substantial communication overhead and a reduction in computational intensity



1. Background

- Challenge: The activation size grows along with sequence length, and may exceed GPU memory capacity

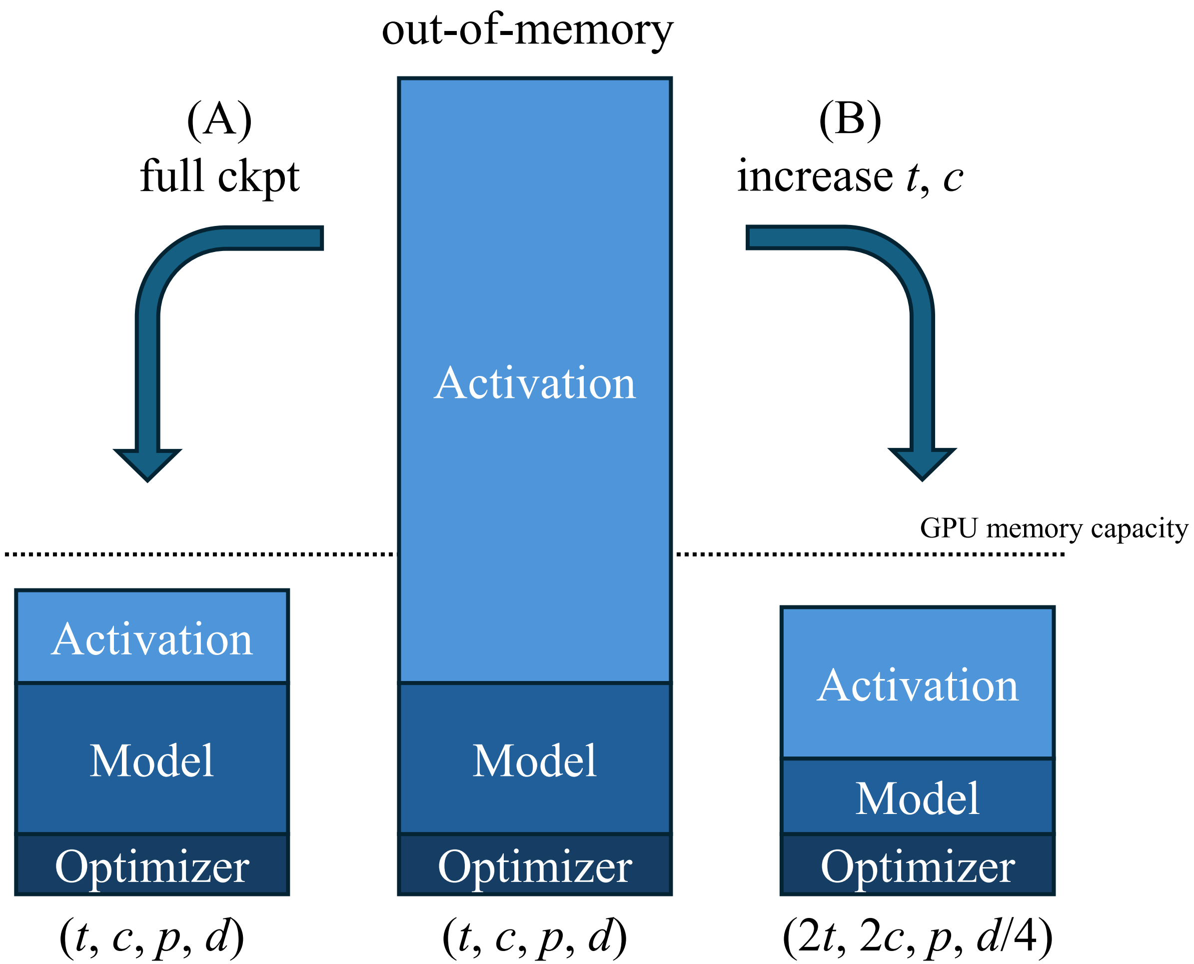
- Traditional solutions
  - A. Full checkpointing
    - Leads to 1/3 additional computation cost
  - B. Increasing TP size and/or CP size
    - Incurs substantial communication overhead and a reduction in computational intensity



1. Background

- Challenge: The activation size grows along with sequence length, and may exceed GPU memory capacity

- Traditional solutions
  - A. Full checkpointing
    - Leads to 1/3 additional computation cost
  - B. Increasing TP size and/or CP size
    - Incurs substantial communication overhead and a reduction in computational intensity



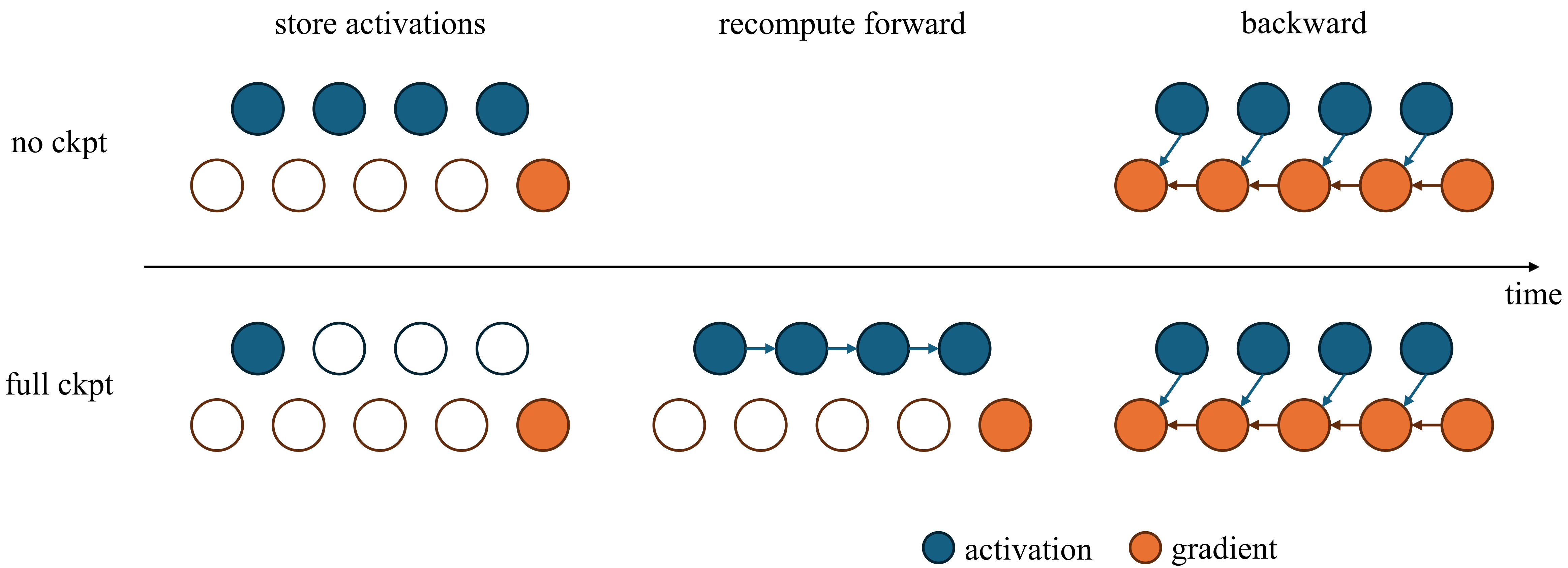
Outline

1. Background
2. Methods
   - Compute-Memory Balanced Checkpointing
   - Pipeline-Parallel-Aware Offloading
   - Hybrid Parallel Parameters Tuning
3. Evaluation
4. Contribution

2.1 Compute-Memory Balanced Checkpointing

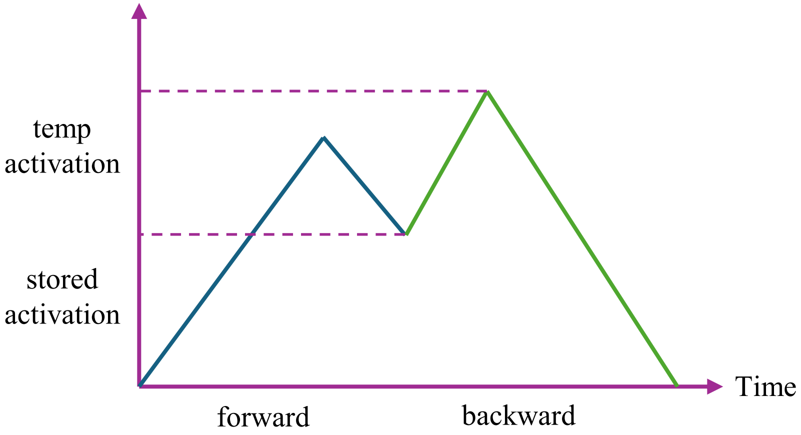
2.1 Compute-Memory Balanced Checkpointing

- Background

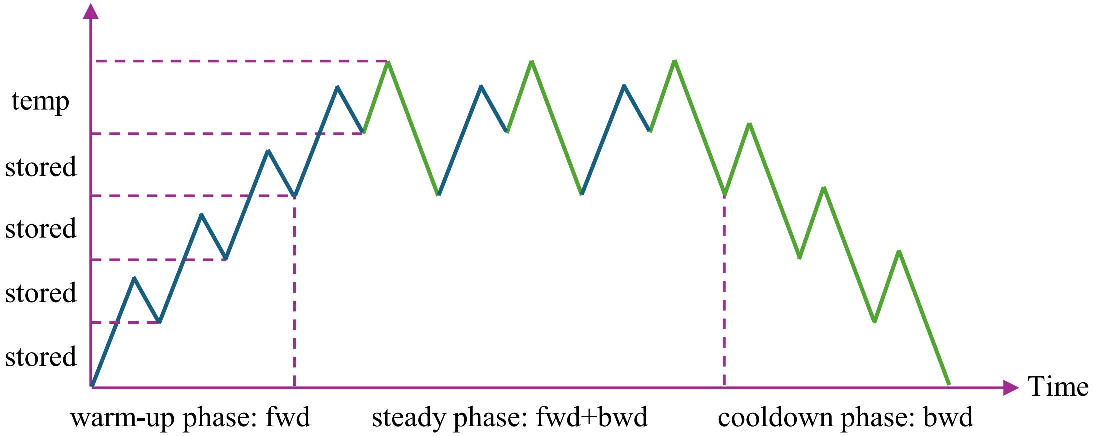


2.1 Compute-Memory Balanced Checkpointing

- Activation Size
  - Traditional checkpointing methods: focus on total activation size of the entire model
  - Pipeline parallelism scenario: should focus on stored activation of sub-models



Traditional model training: Focus on total activation



Pipeline Parallelism: Stored activation dominates

2.1 Compute-Memory Balanced Checkpointing

- Reconstruction Cost
  - Determine the computation cost for each activation tensor
  - Temporary memory can be ignored
    - Reconstruct activations layer by layer
    - All activations of previous layers can be used, no matter whether the previous activation is stored or reconstructed
  - Examples:

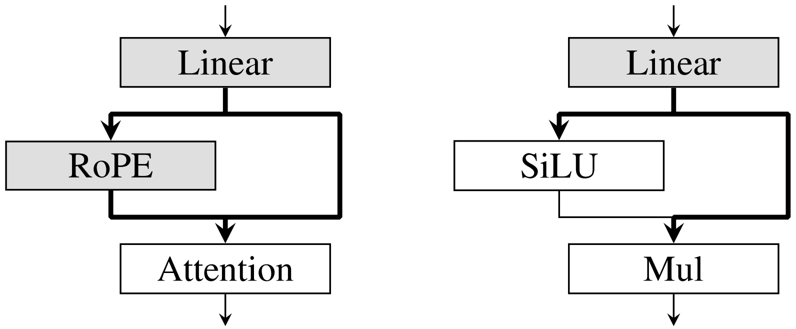
(a) Reconstruct the input of Attention. Two layers are required to recompute.

(b) Reconstruct the input of SiLU. The second operand of Mul is also reconstructed.

2.1 Compute-Memory Balanced Checkpointing

- Reconstruction Cost
  - Determine the computation cost for each activation tensor
  - Temporary memory can be ignored
    - Reconstruct activations layer by layer
    - All activations of previous layers can be used, no matter whether the previous activation is stored or reconstructed

- Examples:

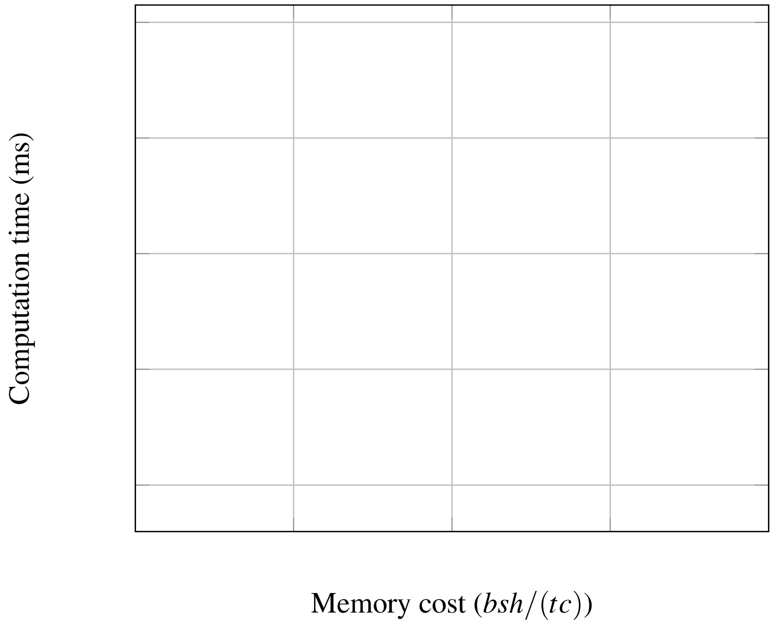


(a) Reconstruct the input of Attention. Two layers are required to recompute.

(b) Reconstruct the input of SiLU. The second operand of Mul is also reconstructed.

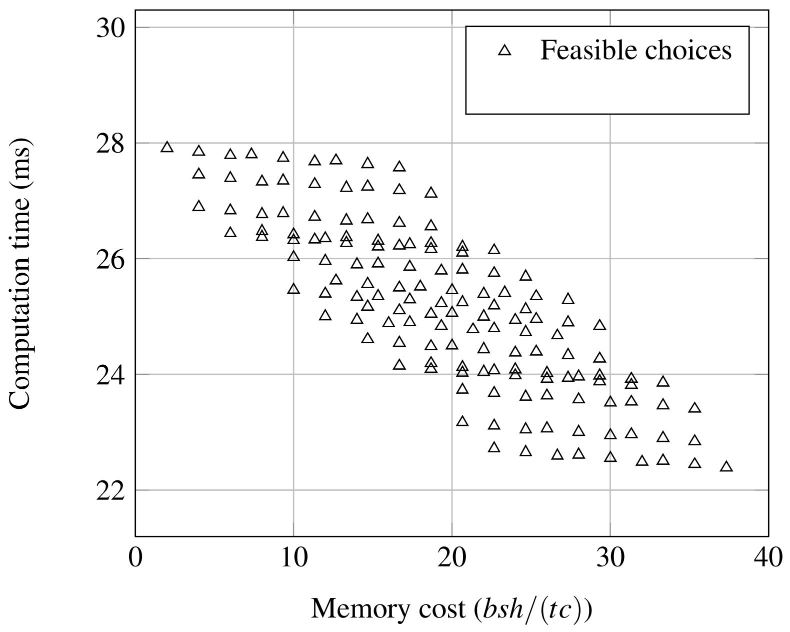
2.1 Compute-Memory Balanced Checkpointing

- Pareto Frontier
  - By enumerating the set of stored activations
  - Determine the minimum computational expenditure for each enumerated memory budget
  - Compute-Memory Balanced Solution
    - Recompute RMSNorm and GLU (SiLU and Mul)
    - Saves 39% memory using only 1.5% recomputing cost



2.1 Compute-Memory Balanced Checkpointing

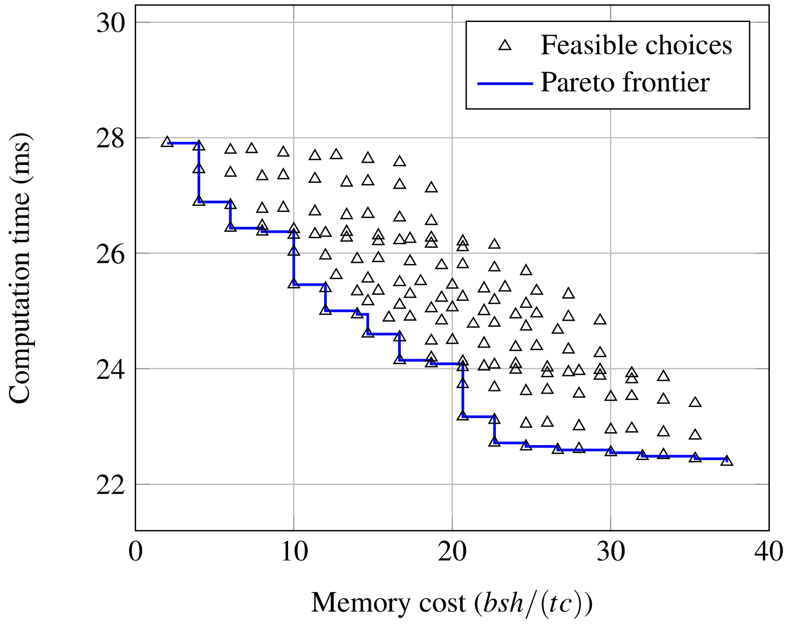
- Pareto Frontier
  - By enumerating the set of stored activations
  - Determine the minimum computational expenditure for each enumerated memory budget
- Compute-Memory Balanced Solution
  - Recompute RMSNorm and GLU (SiLU and Mul)
  - Saves 39% memory using only 1.5% recomputing cost



KUAISHOU

2.1 Compute-Memory Balanced Checkpointing

- Pareto Frontier
  - By enumerating the set of stored activations
  - Determine the minimum computational expenditure for each enumerated memory budget

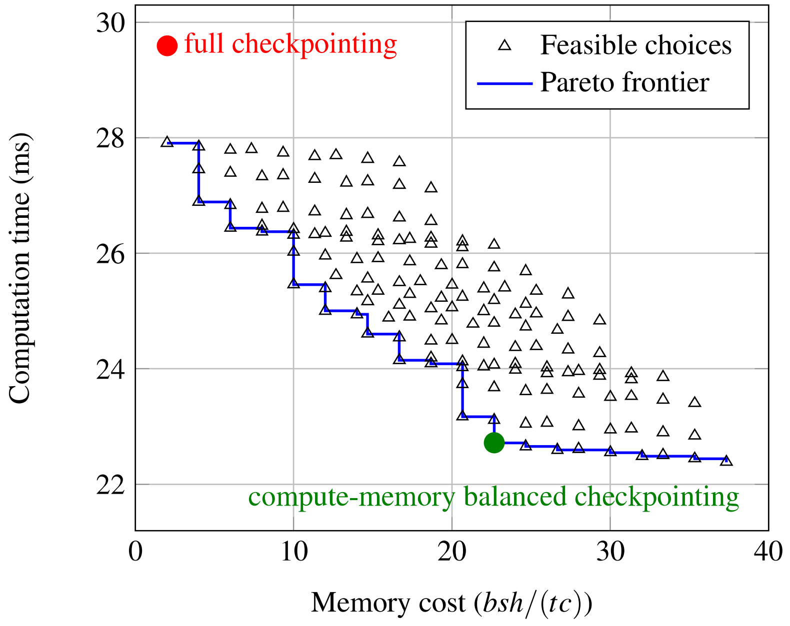


- Compute-Memory Balanced Solution
  - Recompute RMSNorm and GLU (SiLU and Mul)
  - Saves 39% memory using only 1.5% recomputing cost

2.1 Compute-Memory Balanced Checkpointing

- Pareto Frontier
  - By enumerating the set of stored activations
  - Determine the minimum computational expenditure for each enumerated memory budget

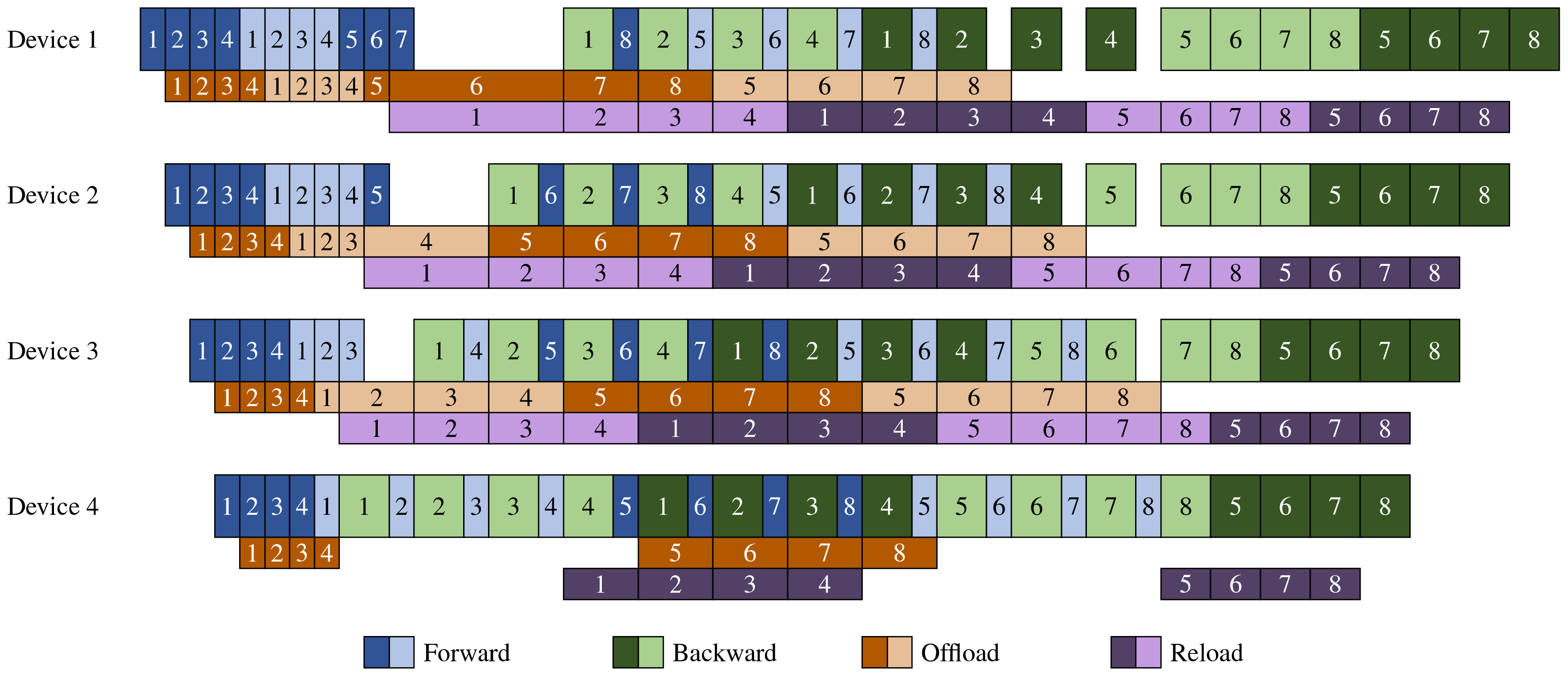
  - Compute-Memory Balanced Solution
    - Recompute RMSNorm and GLU (SiLU and Mul)
    - Saves 39% memory using only 1.5% recomputing cost



2.2 Pipeline-Parallelism-Aware Offloading

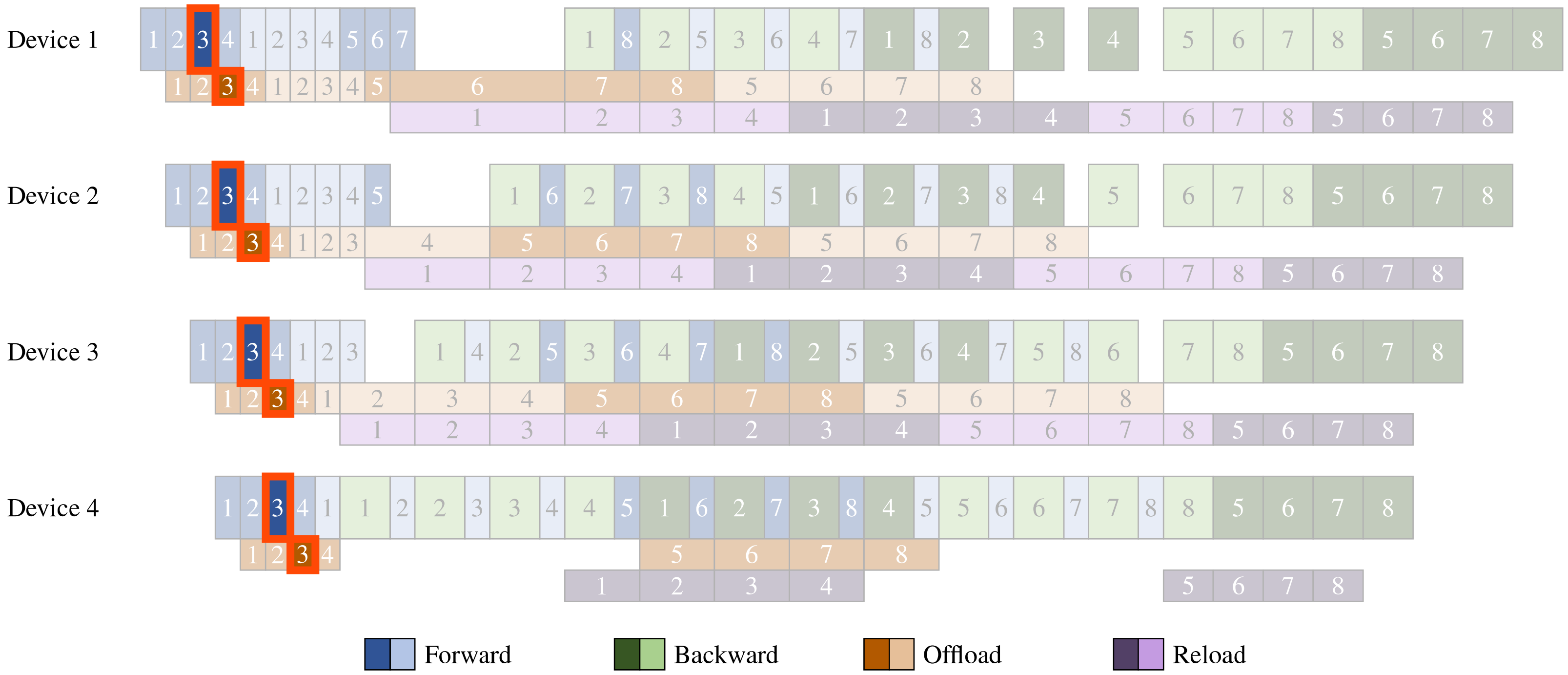
2.2 Pipeline-Parallelism-Aware Offloading

- “Activation blocks” are offloaded to the host memory
  - Activation block: generated by one pipeline stage (typically 1 ~ 2 transformer layers)



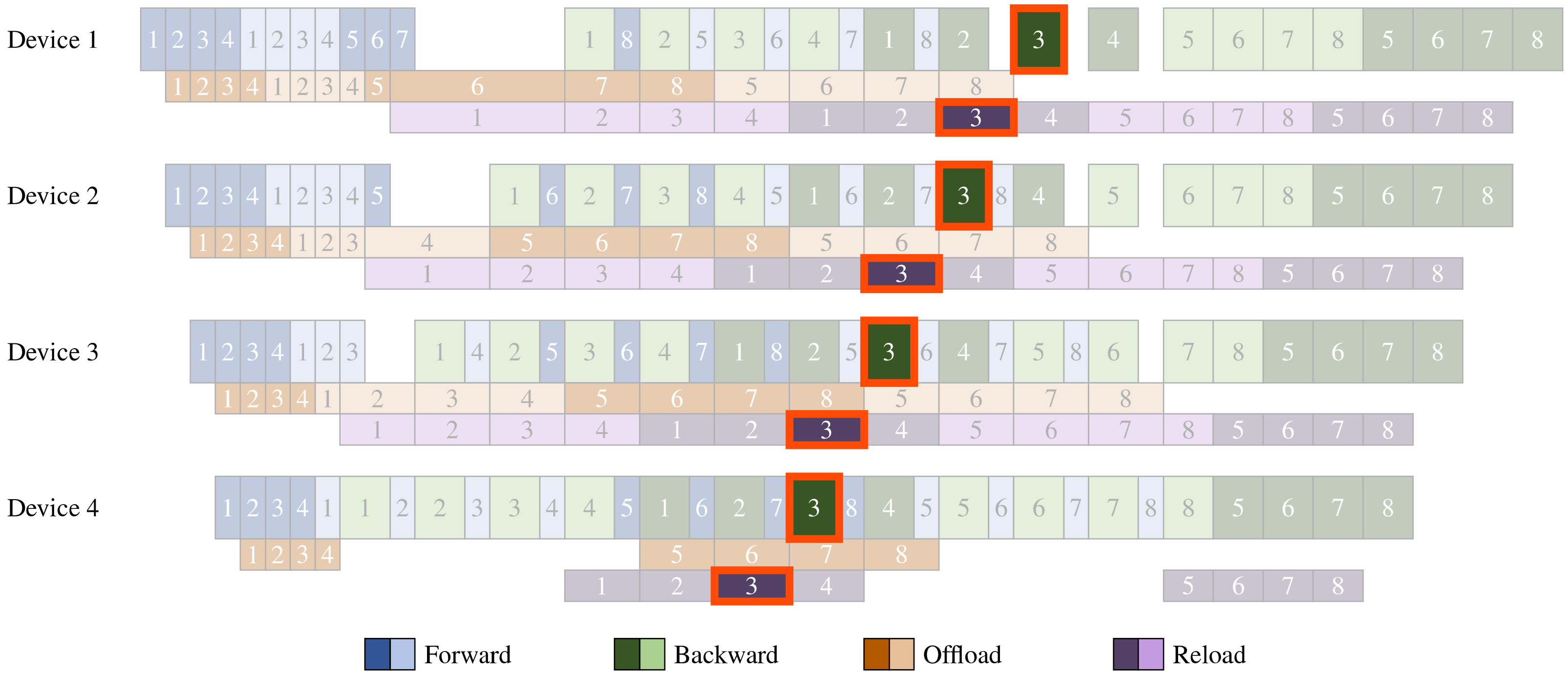
2.2 Pipeline-Parallelism-Aware Offloading

- Schedule of Offloading and Reloading
  - Offloading starts as soon as possible after the end of each pipeline stage forward
  - Reloading starts at the beginning of the previous pipeline stage backward



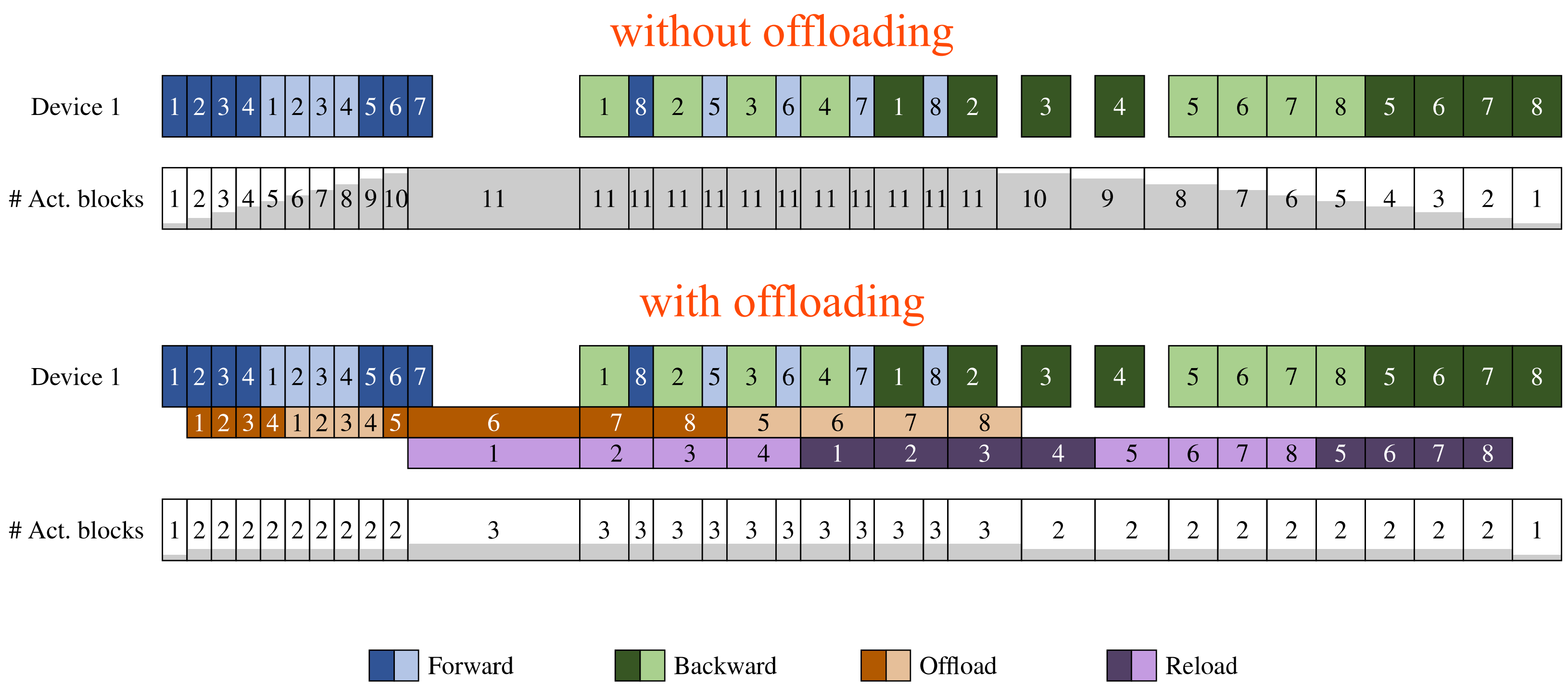
2.2 Pipeline-Parallelism-Aware Offloading

- Schedule of Offloading and Reloading
  - Offloading starts as soon as possible after the end of each pipeline stage forward
  - Reloading starts at the beginning of the previous pipeline stage backward



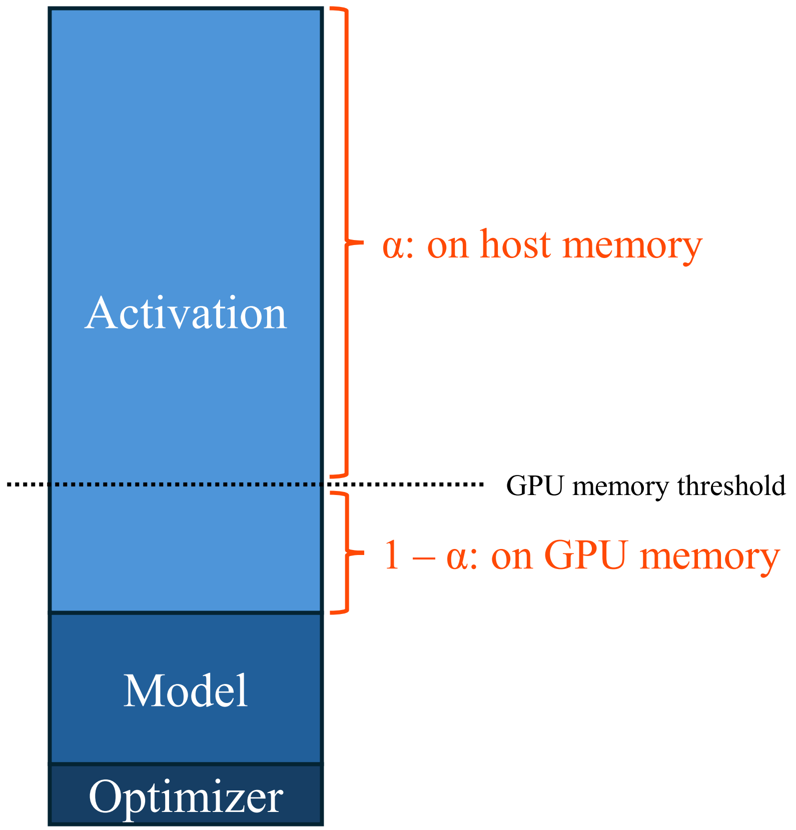
2.2 Pipeline-Parallelism-Aware Offloading

- Reduced number of "Activation Blocks" on GPU

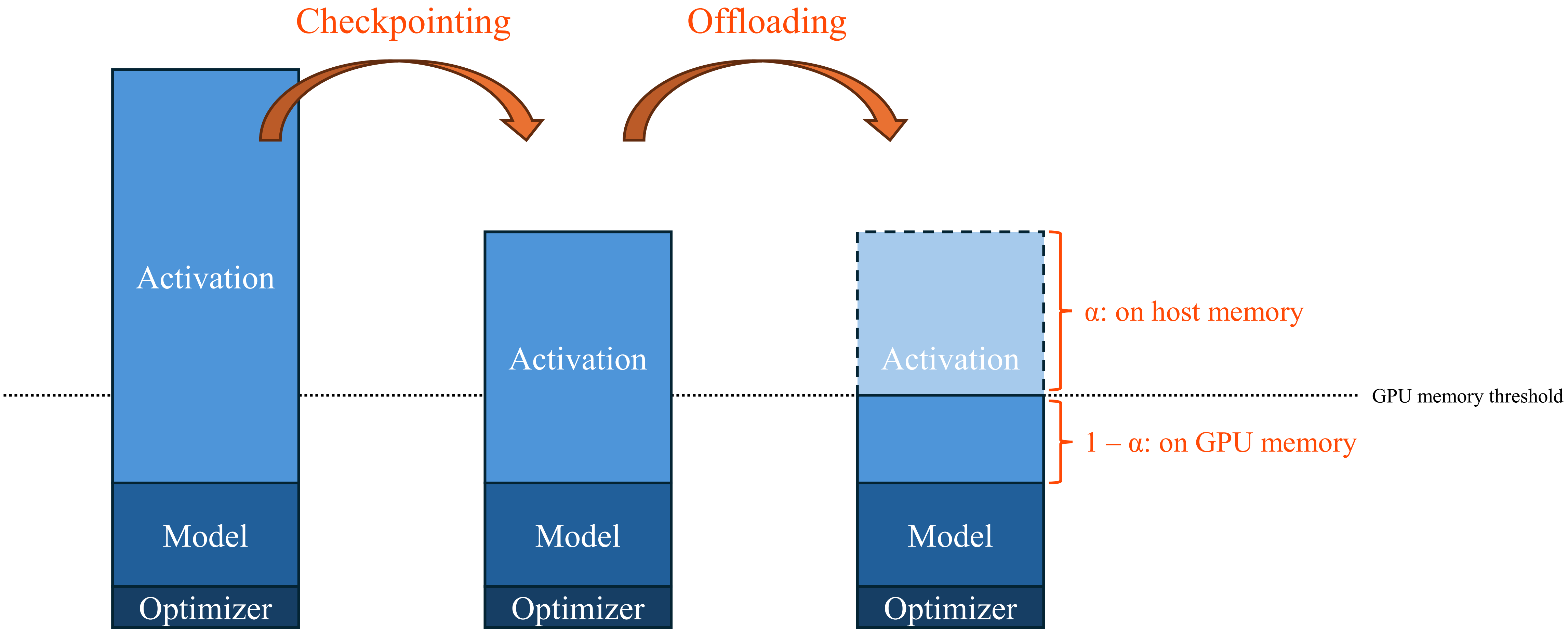


2.2 Pipeline-Parallelism-Aware Offloading

- Bandwidth Utilization Enhancing
    - Bidirectional memory copy
    - Bind to Non-Uniform Memory Access (NUMA) node
    - Use page-locked memory
- Offload Ratio
    - Activation is partially offloaded to host memory
    - Offload ratio α ($0 \leq \alpha \leq 1$) is used to control how much activation is offloaded to host memory
    - Select offload ratio as low as possible for two reasons
        1. Memory copy between host and device may slightly slow down computation due to resource competition;
        2. Offloading may not always be completely overlapped with computation.



2.1 & 2.2: Memory View



2.3 Hybrid Parallel Parameters Tuning

Model
- #layers
- hidden size
- sequence length
- global batch size

Cluster
- GPU type
- #GPUs
- Network

Configuration
- TP size = ?
- CP size = ?
- PP size = ?
- virtual PP size = ?
- ckpt = no/full?

further amplifies the complexity of parameter tuning

- ckpt = balanced?
- offload α = ?

2.3 Hybrid Parallel Parameters Tuning

- Challenge
  - Given a model and a cluster, the number of combinations of (t, c, p, l, ckpt) is vast,
  - even if we have some prior knowledge
    - Avoid inter-node TP communication for all models: $t \leq 8$
    - Avoid inter-node CP communication for multi-head attention (MHA) models: $tc \leq 8$

#GPUs	Llama-175B		Llama-65B		Llama2-70B	
	$\#(t,c)$	$\#(t,c,p,l)$	$\#(t,c)$	$\#(t,c,p,l)$	$\#(t,c)$	$\#(t,c,p,l)$
64	10	141	10	86	14	106
192	10	287	10	86	14	106
240	10	175	10	125	14	141
256	10	160	10	90	22	178
1024	10	160	10	90	30	250
7680	10	310	10	190	34	514

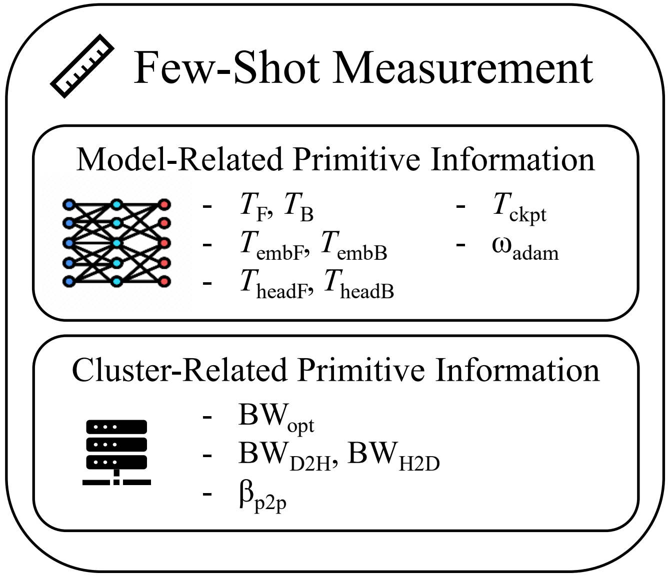
  - Hand-crafted parameters often result in suboptimal combinations of parallel options

2.3 Hybrid Parallel Parameters Tuning

- Cost Model
  - Primitive information: Reduce measurement while pursuing accuracy of the cost model
  - Equations: Take forward/backward time, pipeline bubble, optimizer update time, impact of overlap, and memory size into account

2.3 Hybrid Parallel Parameters Tuning

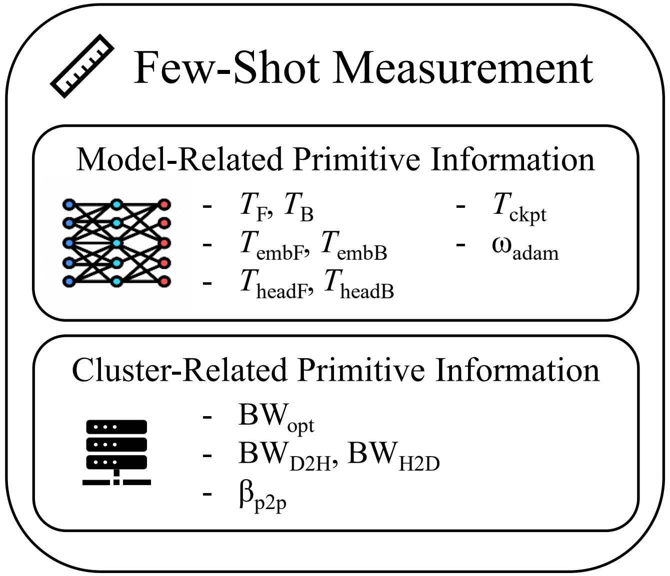
- Cost Model
  - Primitive information: Reduce measurement while pursuing accuracy of the cost model
  - Equations: Take forward/backward time, pipeline bubble, optimizer update time, impact of overlap, and memory size into account



Symbol	Measured times	Time to measure
T_{embF}, T_{embB} T_F, T_B T_{headF}, T_{headB}	each model, each (b,s,t,c)	$2 \sim 15$ min for each model each (b,s)
T_{ckpt}	each $(b, s/(tc), h, s/c, H/t)$	total <15 min for all models (shared among models)
T_{p2p}	each $(2bsh/(tc))$	
BW_{opt}	each (t, cd)	
BW_{DtoH} BW_{HtoD} BW_{bidir} ω_{adam} β_{p2p} $\beta_{offload}$	once	total <10 min for all models (shared among models)

2.3 Hybrid Parallel Parameters Tuning

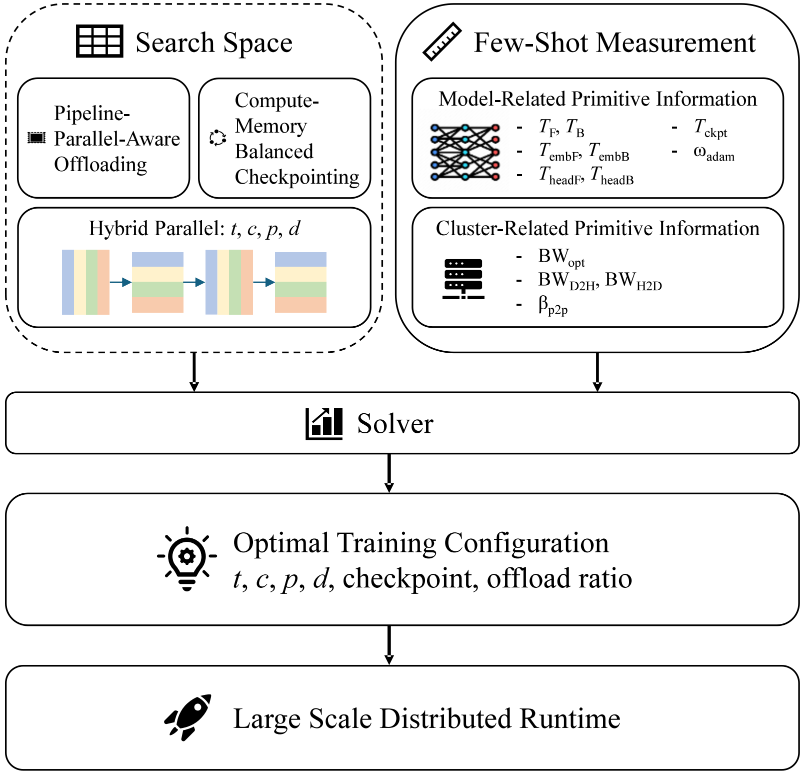
- Cost Model
  - Primitive information: Reduce measurement while pursuing accuracy of the cost model
  - Equations: Take forward/backward time, pipeline bubble, optimizer update time, impact of overlap, and memory size into account



Symbol	Measured times	Time to measure
T_{embF}, T_{embB} T_F, T_B T_{headF}, T_{headB}	each model, each (b,s,t,c)	$2 \sim 15$ min for each model each (b,s)
T_{ckpt}	each $(b, s/(tc), h, s/c, H/t)$	total <15 min for
T_{p2p}	each $(2bsh/(tc))$	all models (shared
BW_{opt}	each (t, cd)	among models)
BW_{DtoH} BW_{HtoD} BW_{bidir} ω_{adam} β_{p2p} $\beta_{offload}$	once	total <10 min for all models (shared among models)

2.3 Hybrid Parallel Parameters Tuning

- Solver
  - Minimize the modeled time under memory constraints



T_{model}

$$\min_{t,c,p,l,ckpt} T_{model}$$

$$\text{s.t. } M_{gpu} \leq M_{gpu}^{thresh} \wedge M_{host} \leq M_{host}^{thresh}$$

Outline

1. Background
2. Methods
   - Compute-Memory Balanced Checkpointing
   - Pipeline-Parallel-Aware Offloading
   - Hybrid Parallel Parameters Tuning
3. Evaluation
4. Contribution

3.1 Evaluation: Experimental Settings

- Cluster
  - The cluster consists of 32 nodes
  - Each node is equipped with eight NVIDIA H800 80GB GPUs
  - Each node has 1TB of host memory
- Training Info
  - Precision is BF16 with FP32 gradients accumulation
  - Optimizer is Adam with FP32 optimizer states
- Software
  - Megatron-LM + industry level improvement

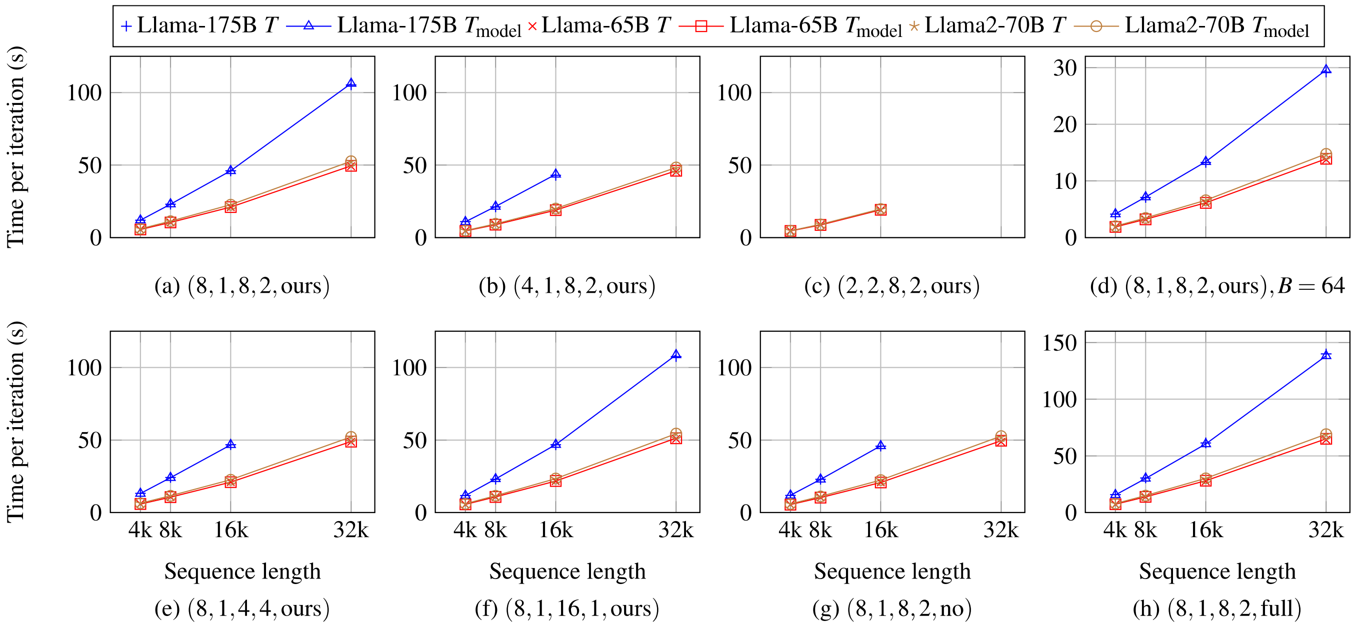
KUAISHOU

3.2 Evaluation: Cost Model

- Verify the cost model using various combinations of

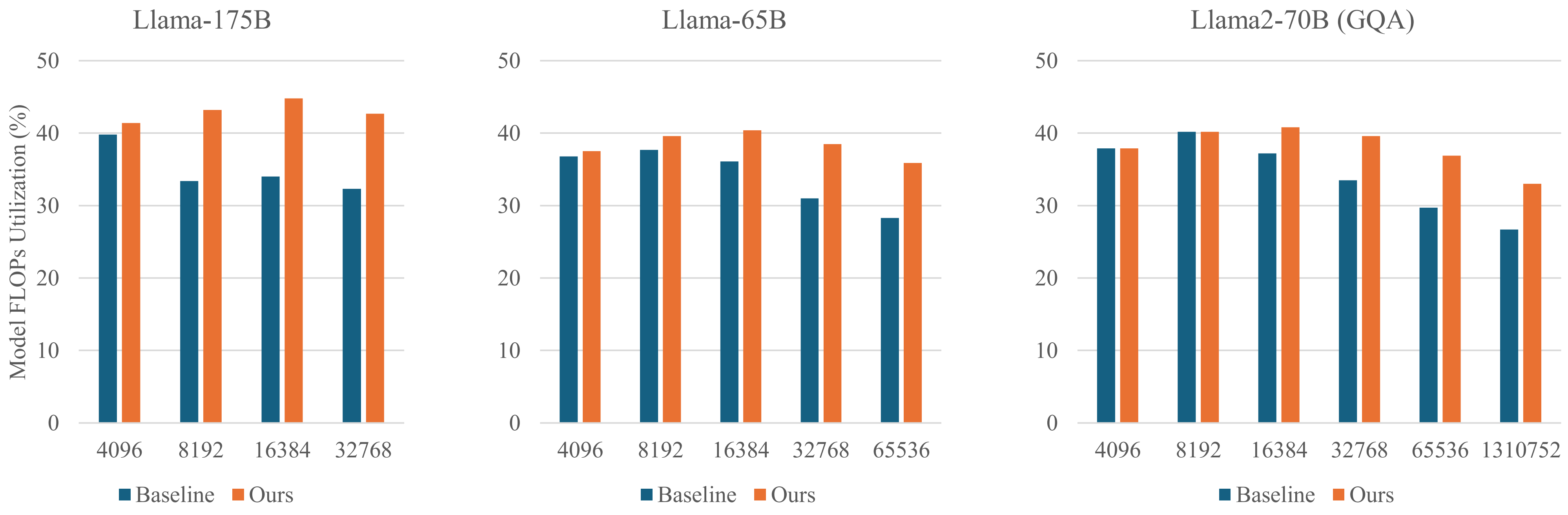
model, sequence length,

(t, c, p, l, ckpt), global batch size



3.3 Evaluation: End-to-End Performance Tuning

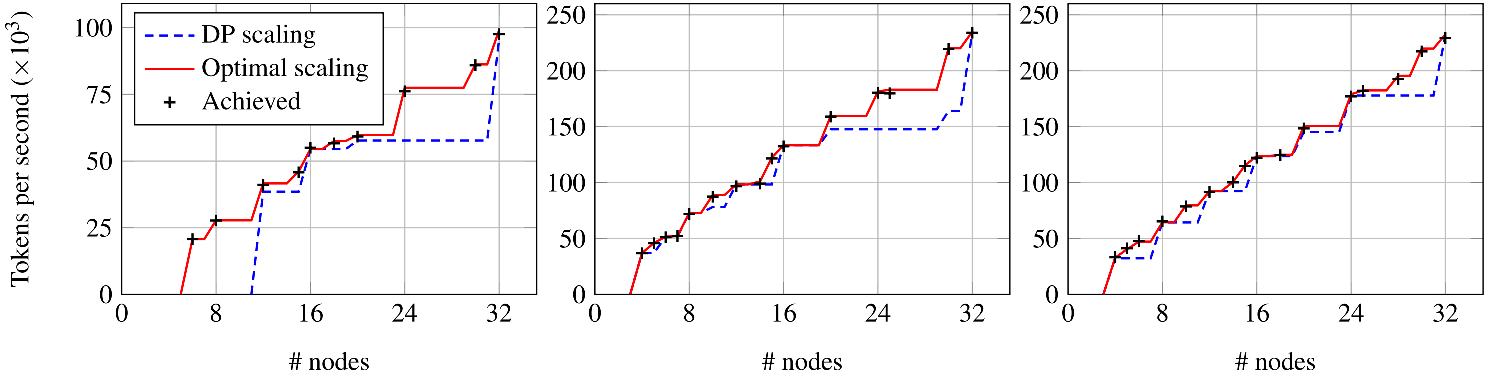
- Enhance the throughput by up to 32%
  - Both "baseline" and "ours" use the optimal hybrid parallel parameters solved by the cost model



(256 H800 GPUs, global batch size is 256)

3.4 Evaluation: Optimal Scaling

- Vary the number of GPUs
  - DP scaling (baseline): only scale DP size when node number changes
  - Optimal scaling: solve global batch size, TP size, CP size, etc. using the cost model



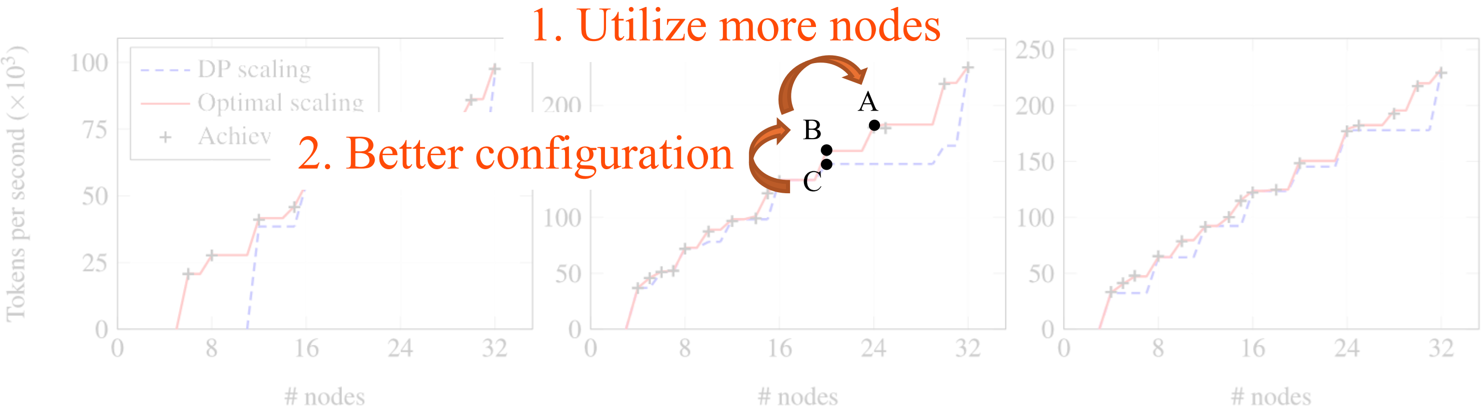
(a) Llama-175B, $s = 4096$ (b) Llama-65B, $s = 4096$ (c) Llama2-70B, $s = 8192$

- Lines are modeled throughput, marks are achieved throughput.

(Satisfying global batch size: 256±16)

3.4 Evaluation: Optimal Scaling

- Vary the number of GPUs
  - DP scaling (baseline): only scale DP size when node number changes
  - Optimal scaling: solve global batch size, TP size, CP size, etc. using the cost model



(a) Llama-175B, $s = 4096$ (b) Llama-65B, $s = 4096$ (c) Llama2-70B, $s = 8192$

  - Lines are modeled throughput, marks are achieved throughput.

(Satisfying global batch size: 256±16)

Outline

1. Background
2. Methods
   - Compute-Memory Balanced Checkpointing
   - Pipeline-Parallel-Aware Offloading
   - Hybrid Parallel Parameters Tuning
3. Evaluation
4. Contribution

4. Contribution

1. Pipeline-Parallel-Aware Offloading
   - Schedule offloading and reloading of activations, following the pipeline parallel schema, fully utilizing host memory to store activations with negligible overhead.
2. Compute-Memory Balanced Checkpointing
   - Balance memory cost and computation cost to achieve the Pareto optimality.
3. Efficient Searching Method
   - find the optimal hybrid parallelism parameters using the performance model measured from cluster-related primitive information and model-related primitive information.
4. Extensive Experiments
   - Example: Increase Model FLOPs Utilization (MFU) from 32.3% to 42.7% for a 175B Llama-like model with a context window size of 32,768 on 256 NVIDIA H800 GPUs.

- Artifact Evaluated: https://github.com/kwai/Megatron-Kwai | branch: atc24ae

Thank you!

Contact: {yuantailing,liuyuliang}@kuaishou.com