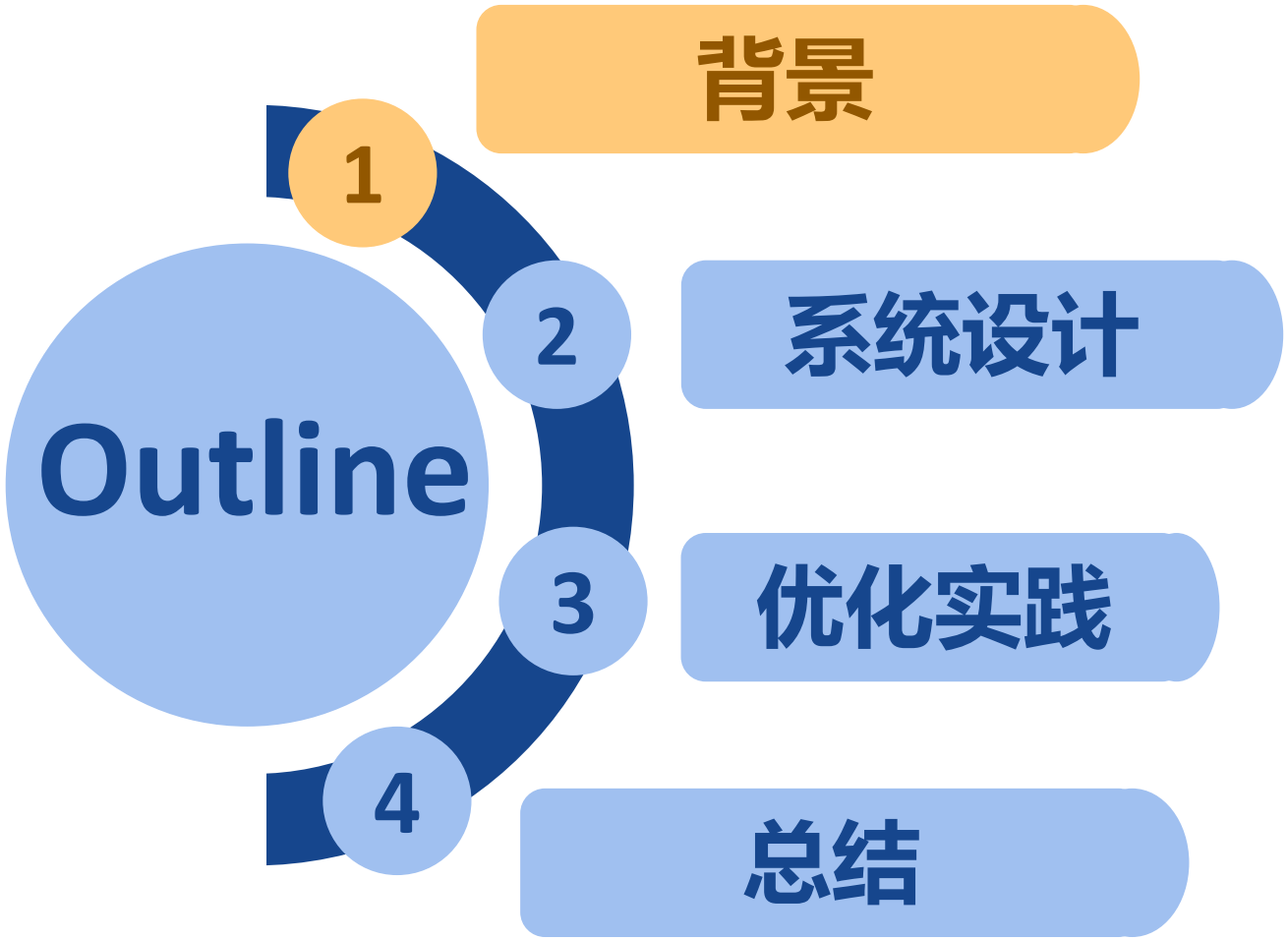
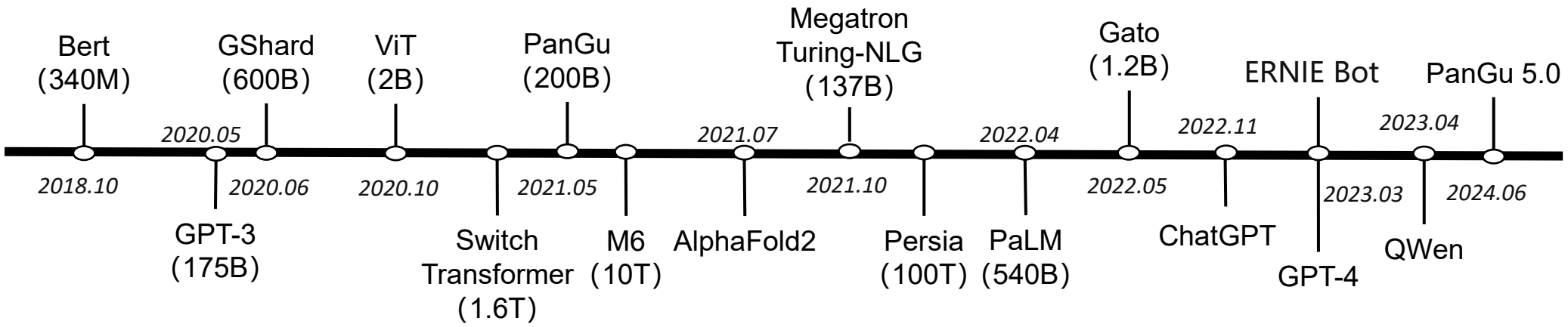
面向Ascend芯片的大模型训练性能剖析与系统优化

周宇航

南京大学


Outline
1
背景
2
系统设计
3
优化实践
4
总结

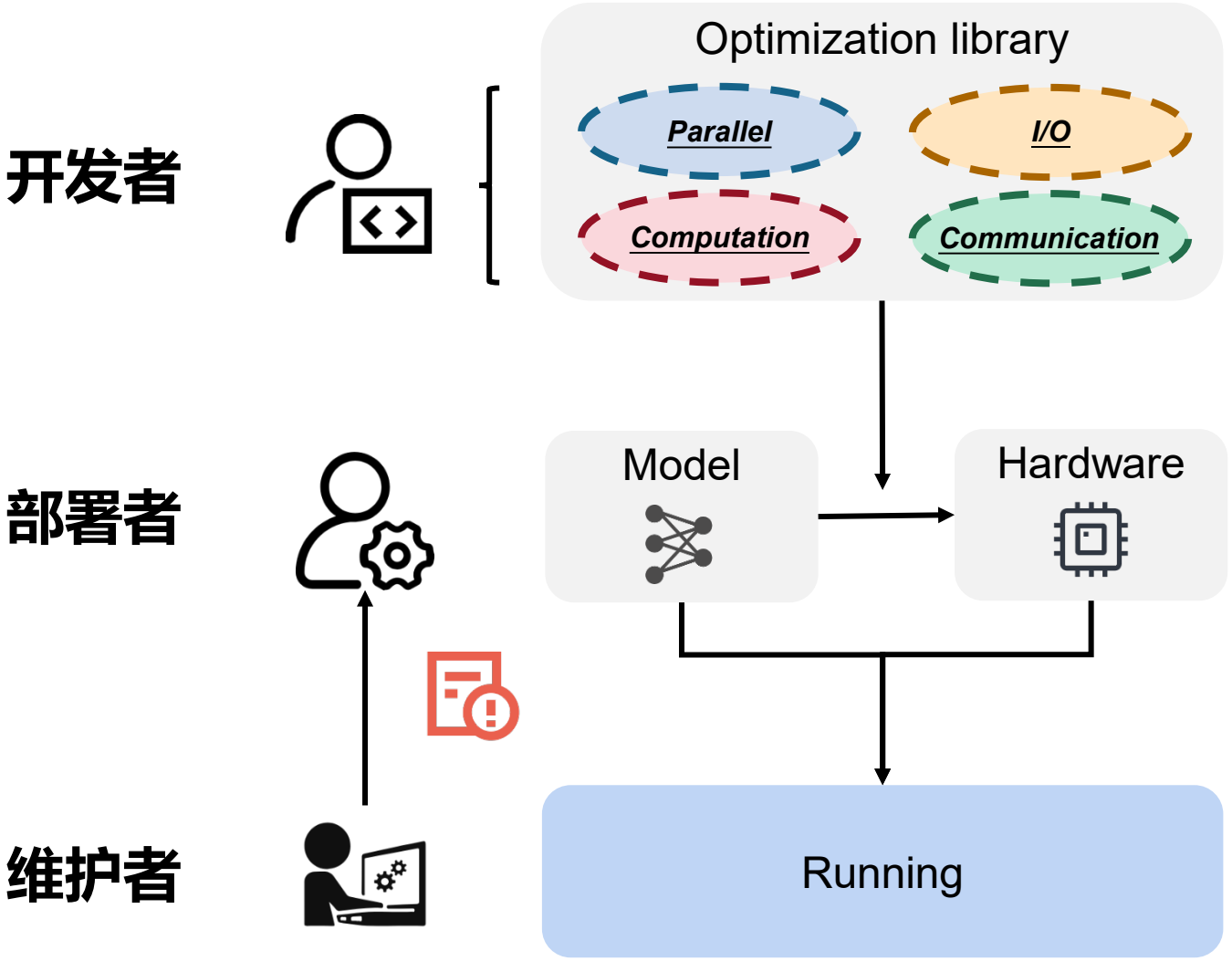
大模型训练的开销巨大



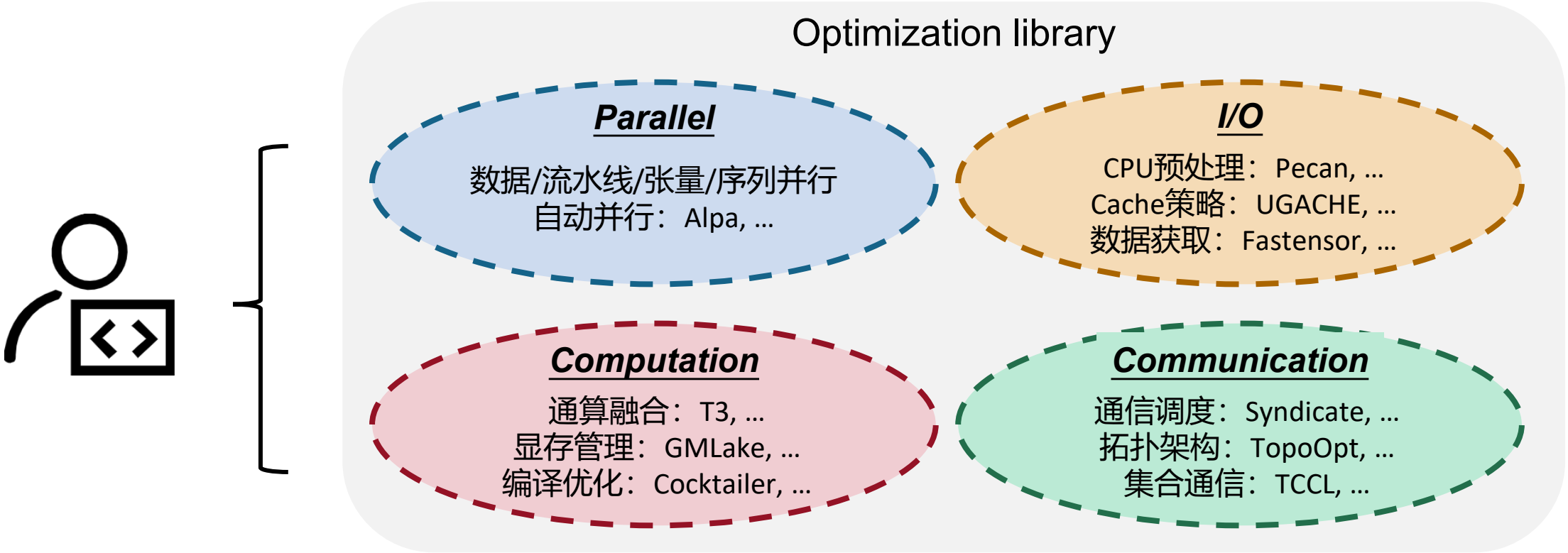
Model	Parameter	Accelerator	Time
GPT-3	100.8B	3072 x A100	84 days
BLOOM	176B	384 x A100	3.5 months
M6	10T	512 x V100	10 days
ViT	2B	N/A x TPU v3	10k core-days
V-MoE	15B	N/A x TPU v3	16.8k core-days

Question: 实践中要训好一个大模型，需要做哪些方面的工作?

训练中的角色分工


开发者
部署者
维护者
Optimization library
Parallel
I/O
Computation
Communication
Model
Hardware
Running

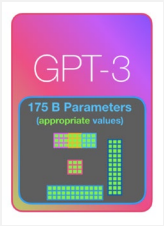
开发者



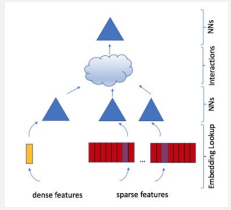
目标一：发现训练瓶颈，进而开发优化

部署者

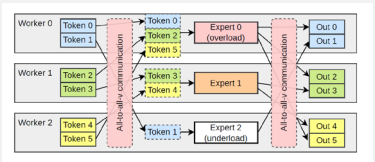
Model



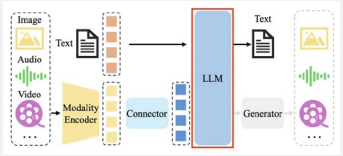
Transformer



DLRM

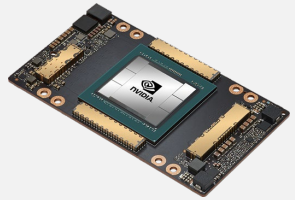


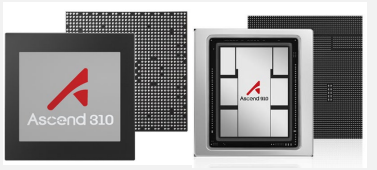
MoE



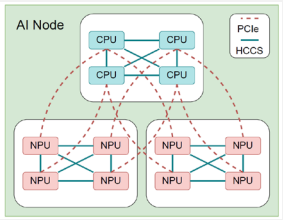
多模态

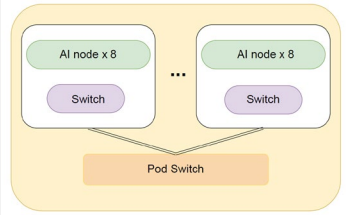
Hardware





GPU vs NPU



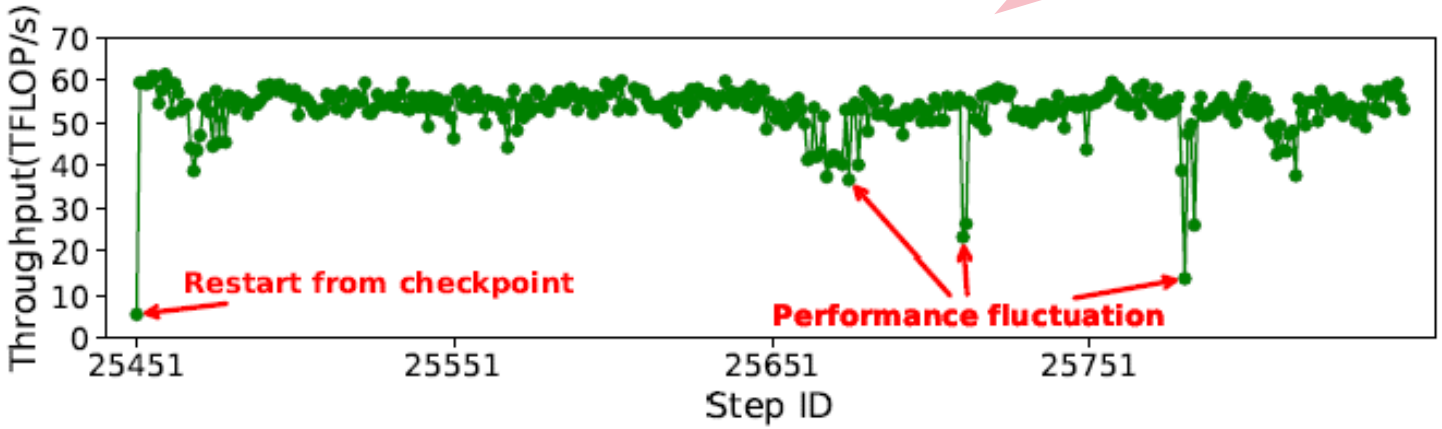


单机 vs 多机

目标二：面对变化的模型和硬件，选用合适的优化提升性能

维护者

性能波动随机出现，
难以预测



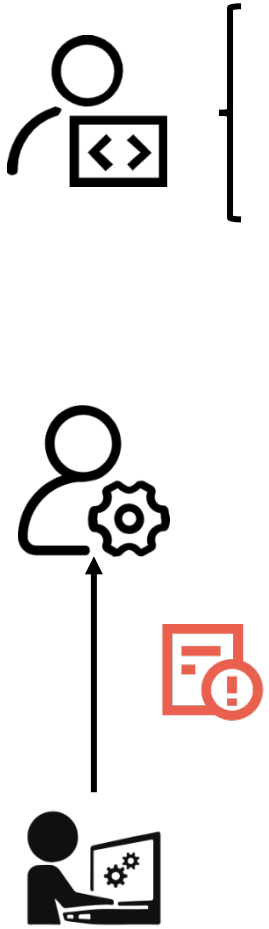
目标三：实时监控训练进程，捕捉性能波动以分析和优化

用户目标

开发者

部署者

维护者



主要目标

发现瓶颈

选取合适的优化

实时监控

用户需求

主要目标

- 发现瓶颈
- 选取合适的优化
- 实时监控

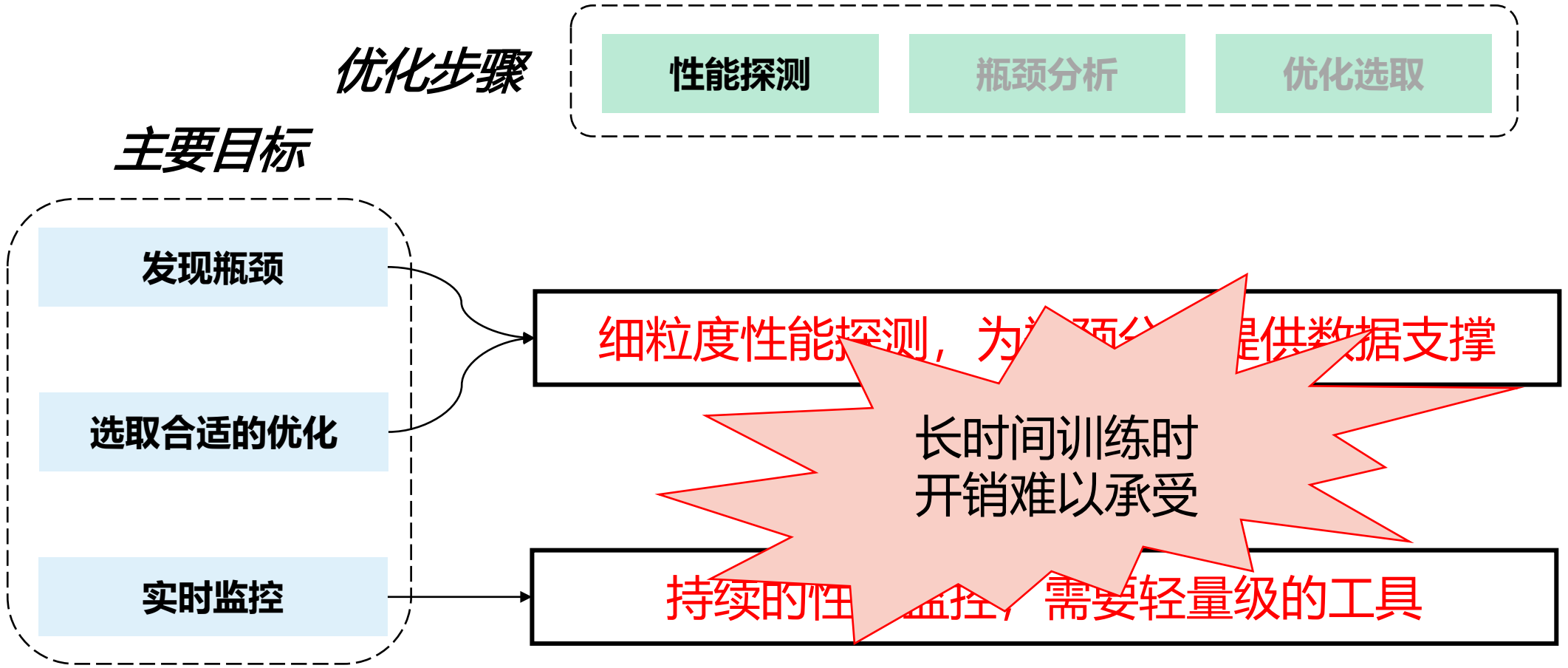
优化步骤

性能探测 → 瓶颈分析 → 优化选取

用户需求

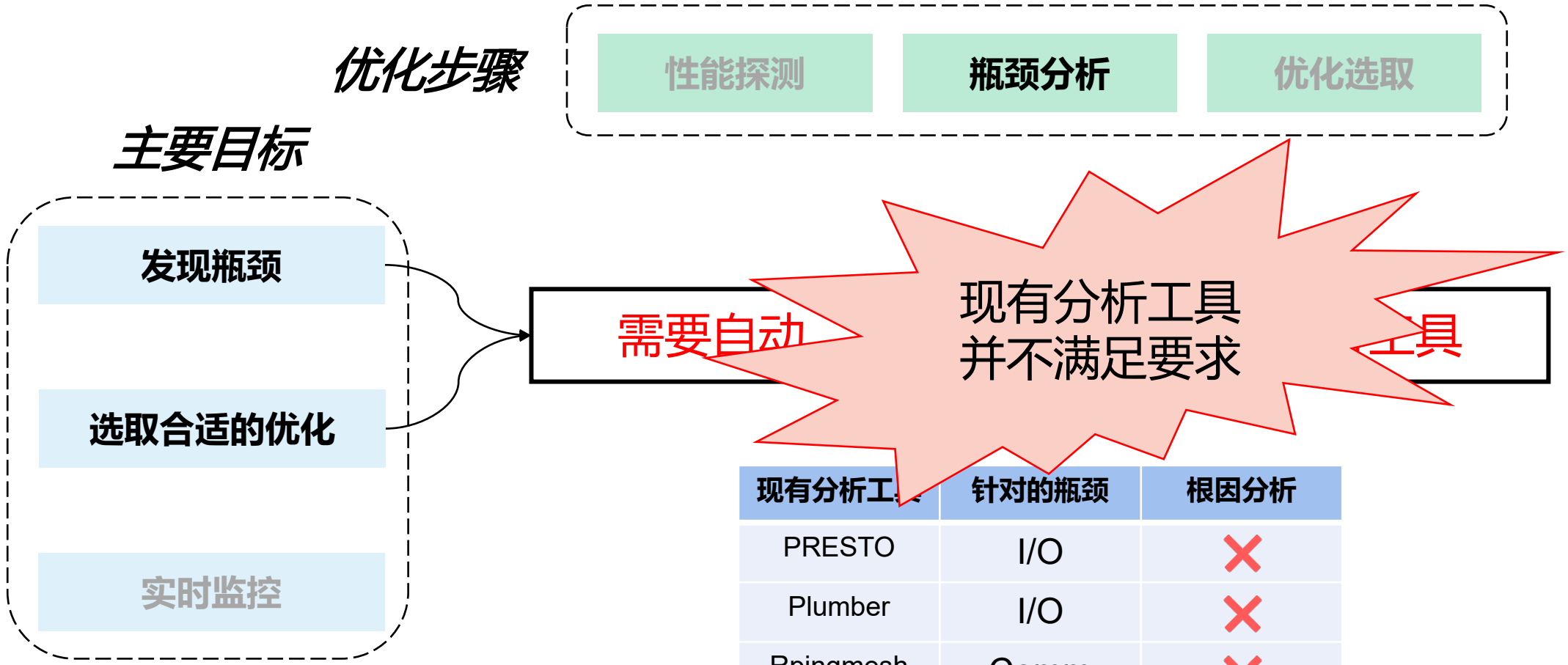
主要目标	性能探测	瓶颈分析	优化选取
发现瓶颈	✓	✓	✗
选取合适的优化	✓	✓	✓
实时监控	✓	✗	✗

现有工作的缺陷



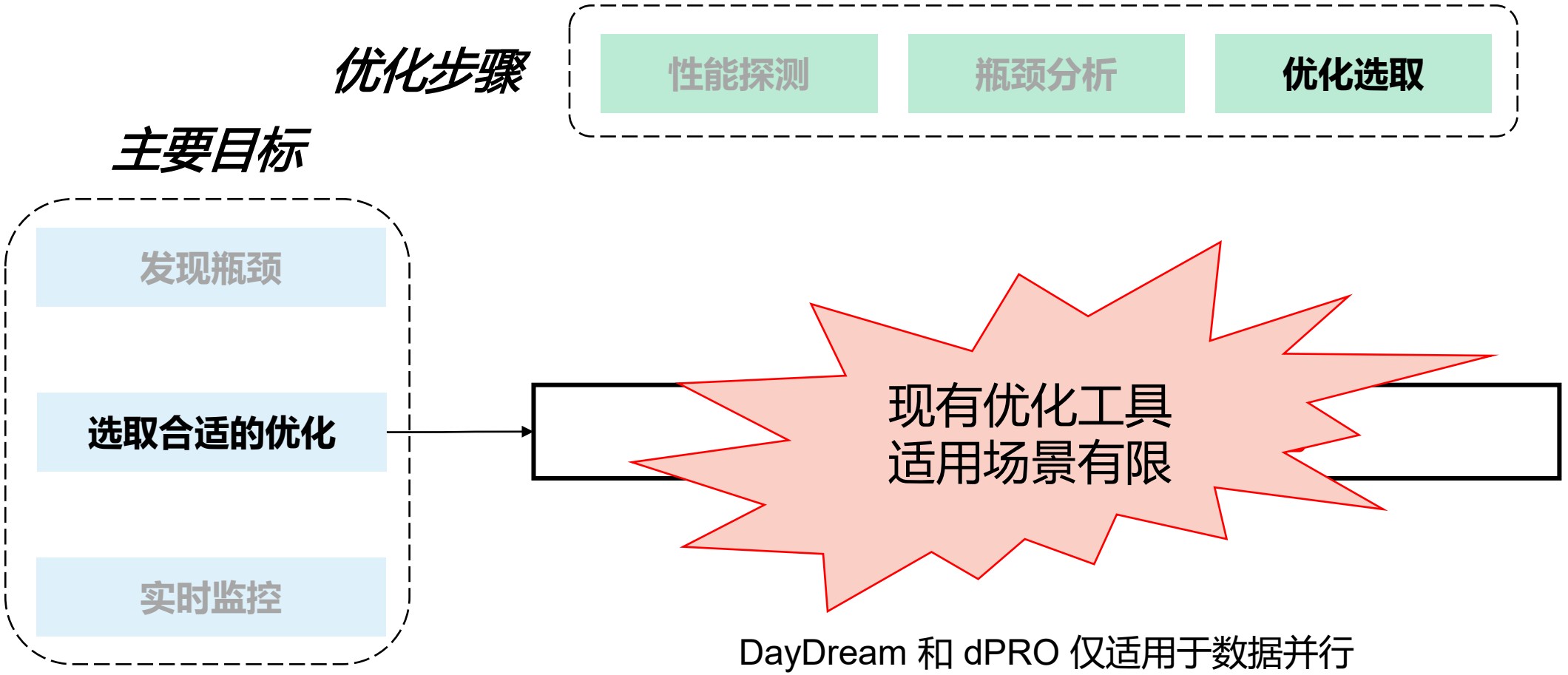
8卡 8B llama3单步训练细粒度探测开销是原本的1.77倍

现有工作的缺陷



现有分析工具	针对的瓶颈	根因分析
PRESTO	I/O	✗
Plumber	I/O	✗
Rpingmesh	Comm	✗
Meta's work	Comm	✗

现有工作的缺陷



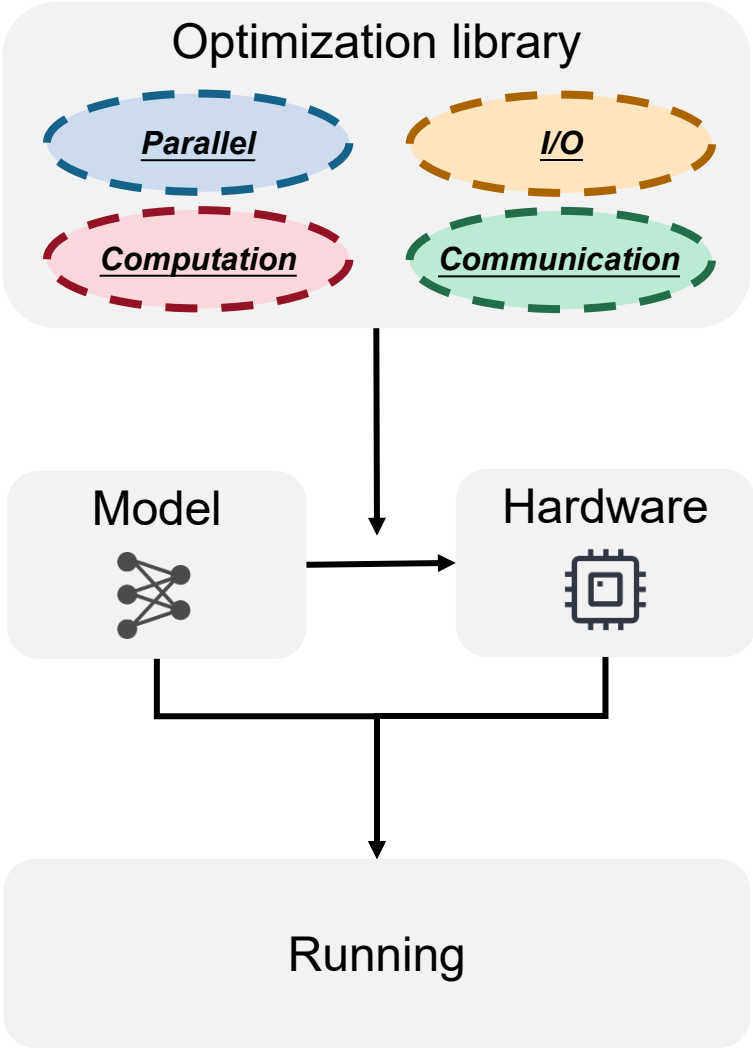
DayDream 和 dPRO 仅适用于数据并行

用户目标

开发者

部署者

维护者

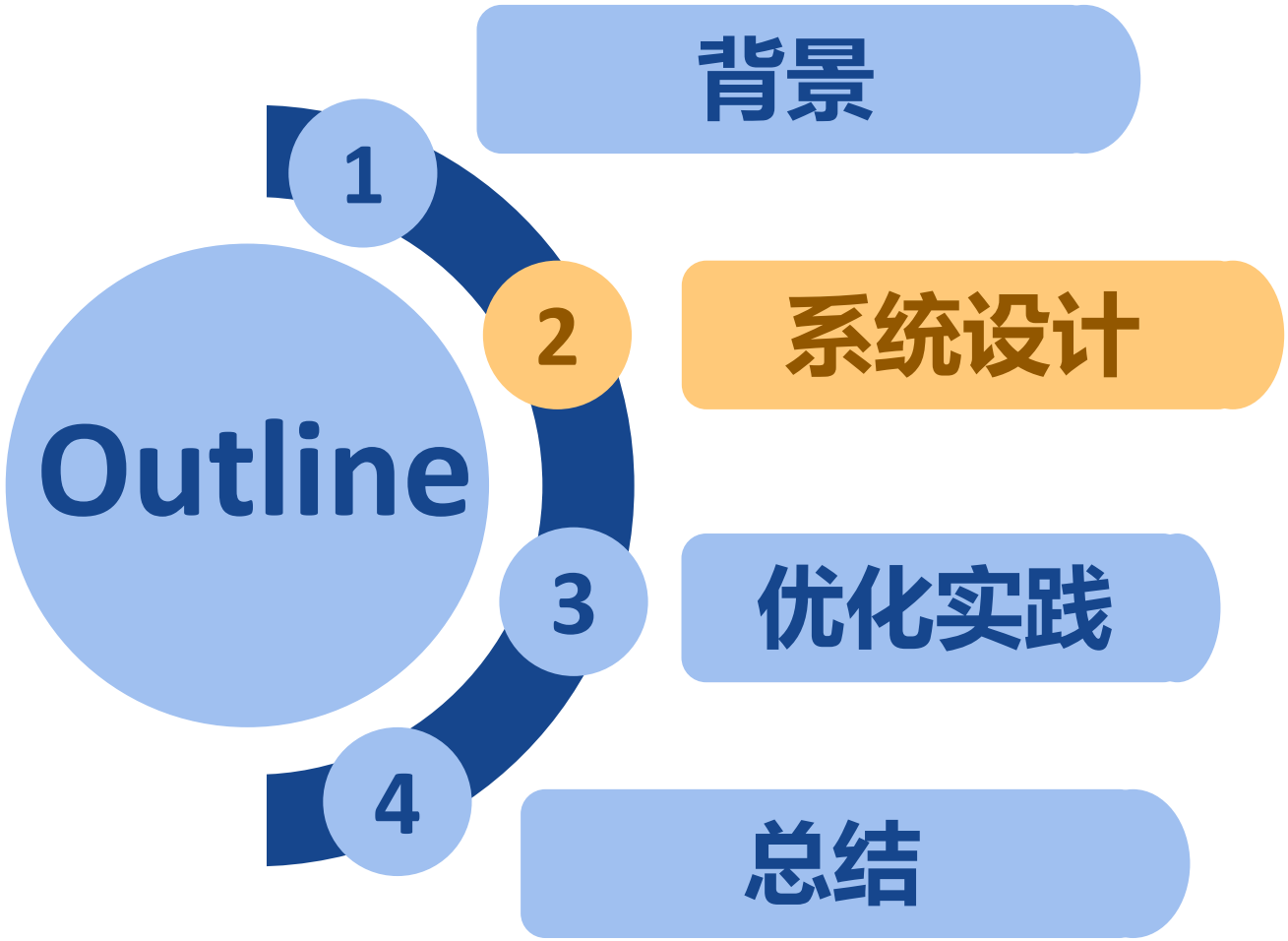


问题

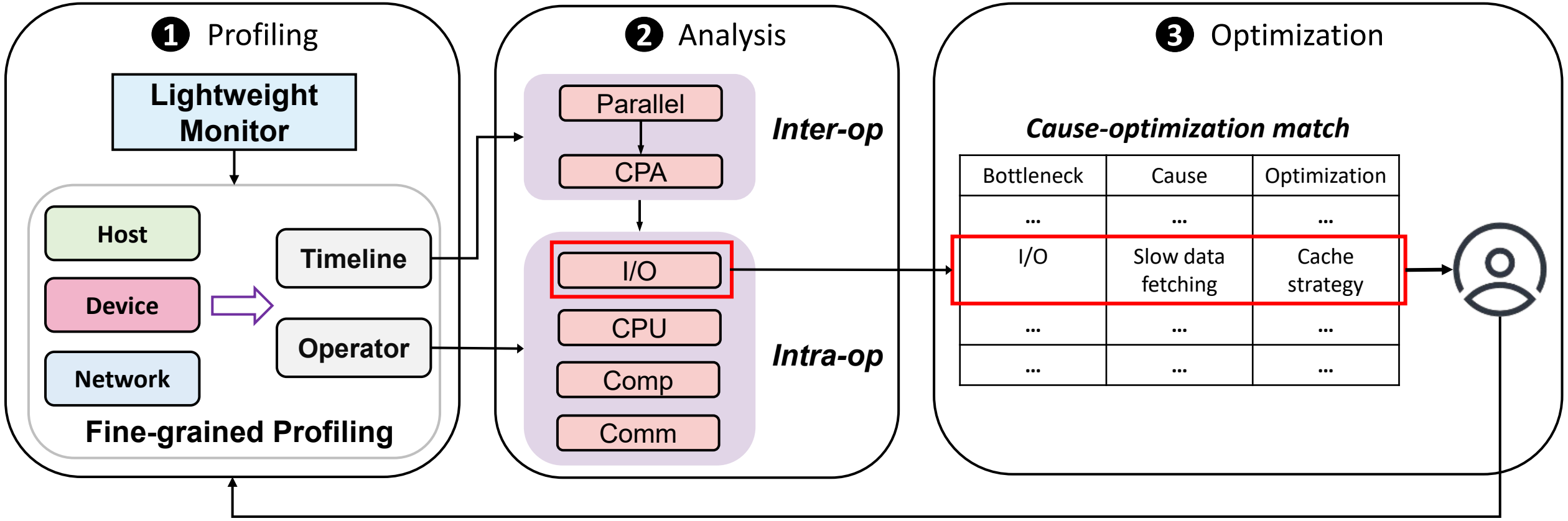
片面的瓶颈分析

受限的优化范围

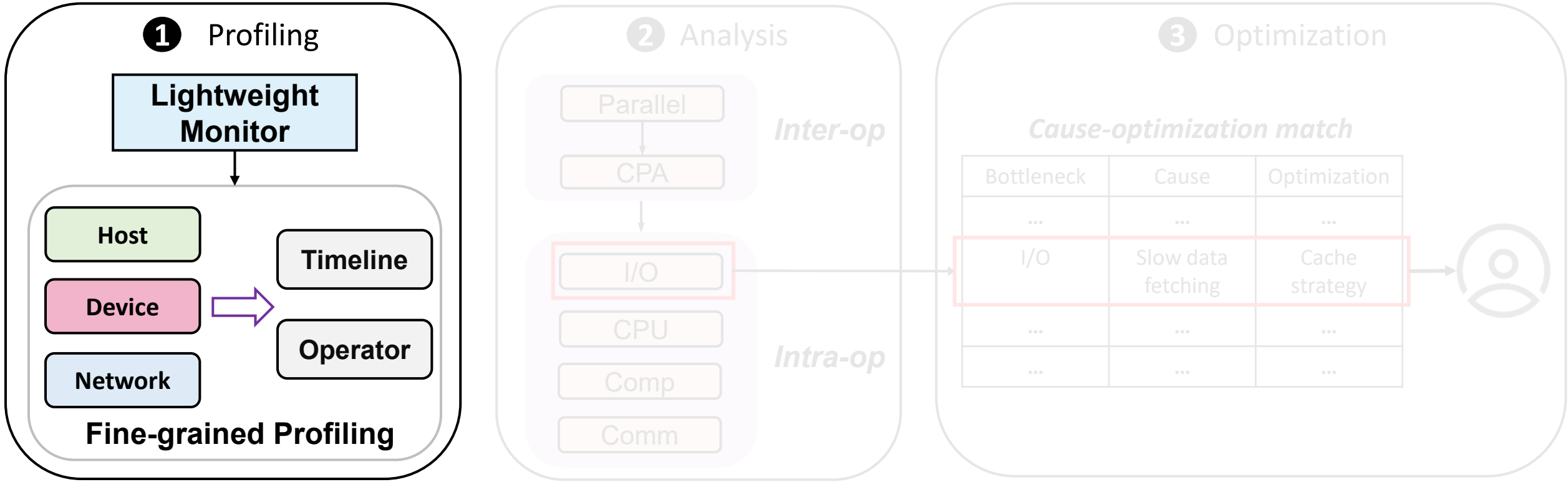
高昂的剖析成本


Outline
1
背景
2
系统设计
3
优化实践
4
总结

Hermes系统

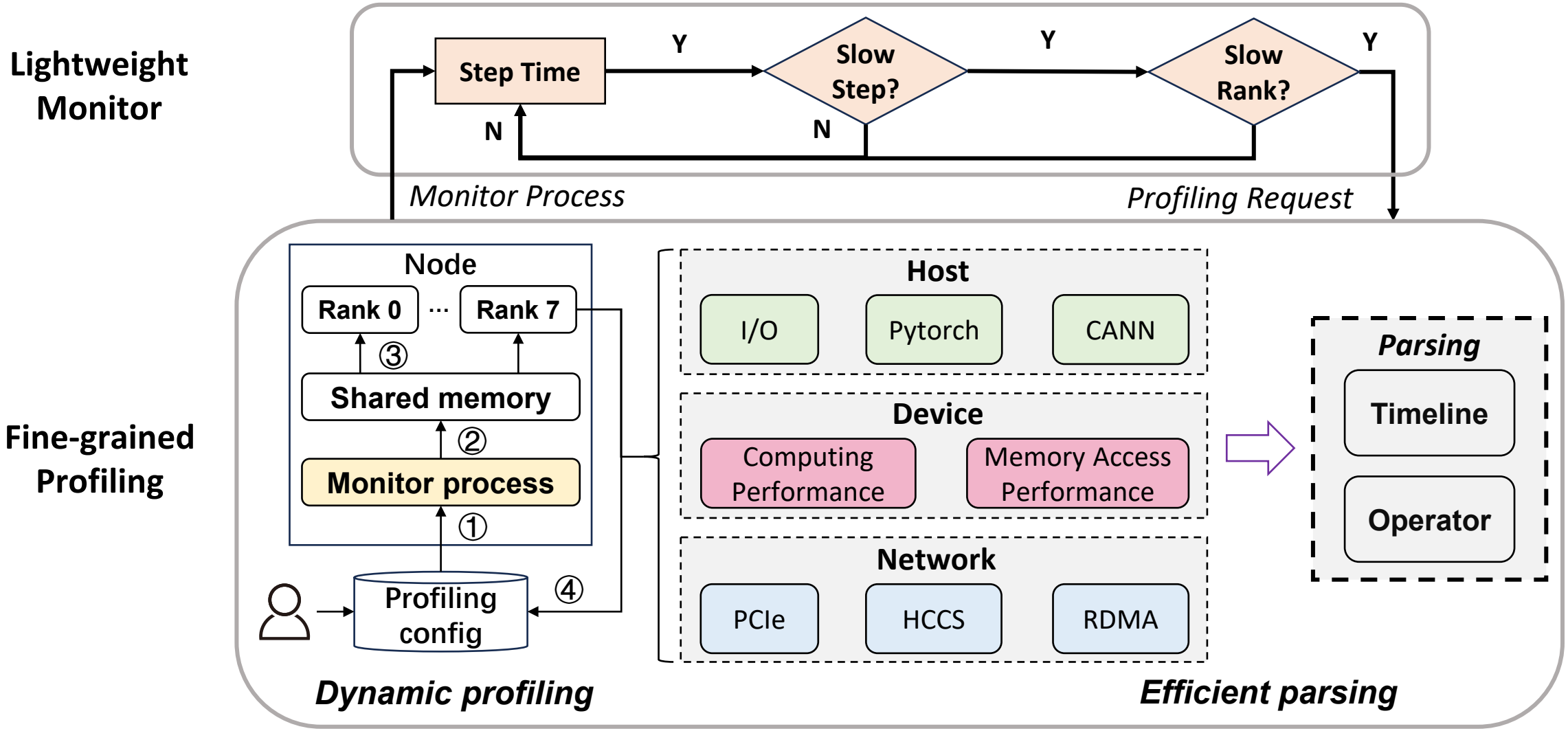

1 Profiling
Lightweight Monitor
Host
Device
Network
Timeline
Operator
Fine-grained Profiling
2 Analysis
Parallel
CPA
Inter-op
I/O
CPU
Comp
Comm
Intra-op
3 Optimization
Cause-optimization match
Bottleneck
Cause
Optimization
...
I/O
Slow data fetching
Cache strategy

Hermes系统

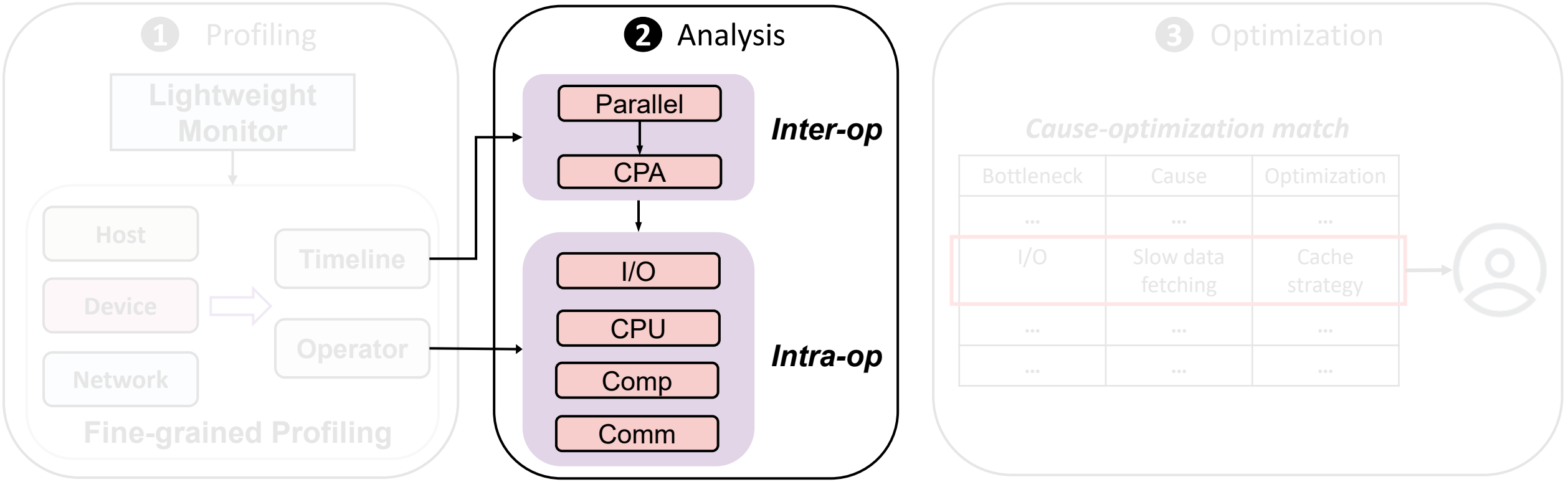


由粗到细的性能探测

由粗到细的性能探测


Lightweight Monitor
Step Time
Y
Slow Step?
Y
Slow Rank?
Y
N
N
Monitor Process
Profiling Request
Fine-grained Profiling
Node
Rank 0
···
Rank 7
③
Shared memory
②
Monitor process
①
Profiling config
④
Host
I/O
Pytorch
CANN
Device
Computing Performance
Memory Access Performance
Network
PCIe
HCCS
RDMA
Parsing
Timeline
Operator
Dynamic profiling
Efficient parsing

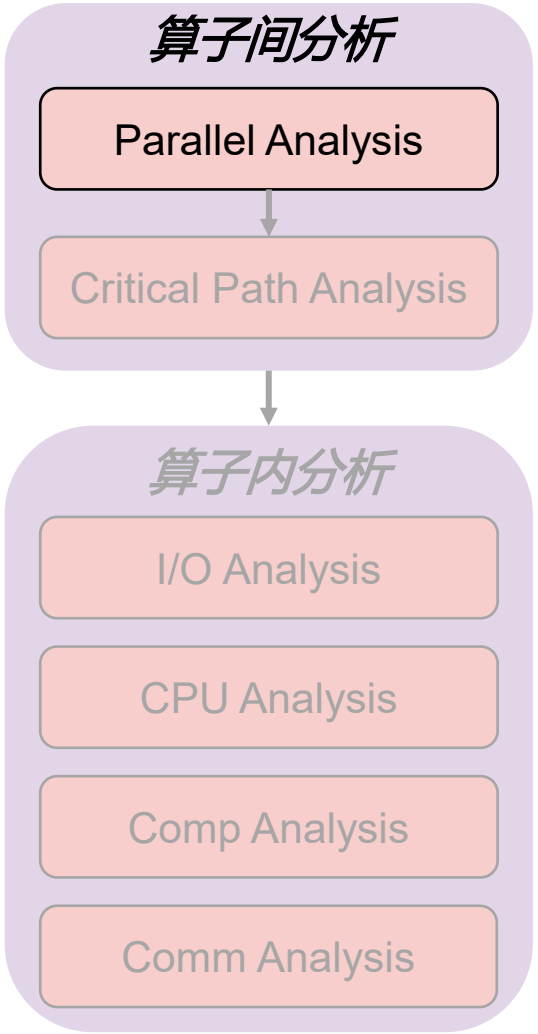
Hermes系统



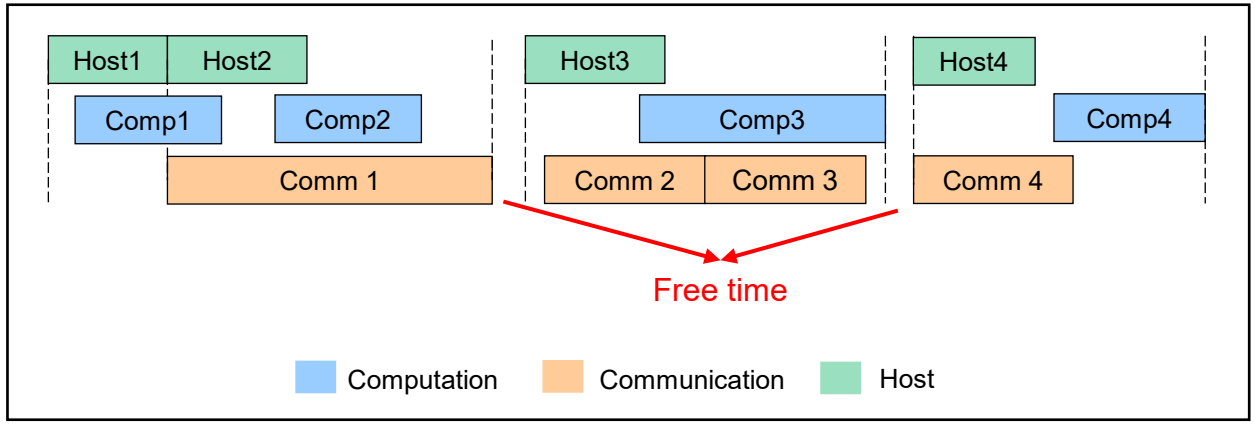
由粗到细的性能探测

层次化瓶颈分析

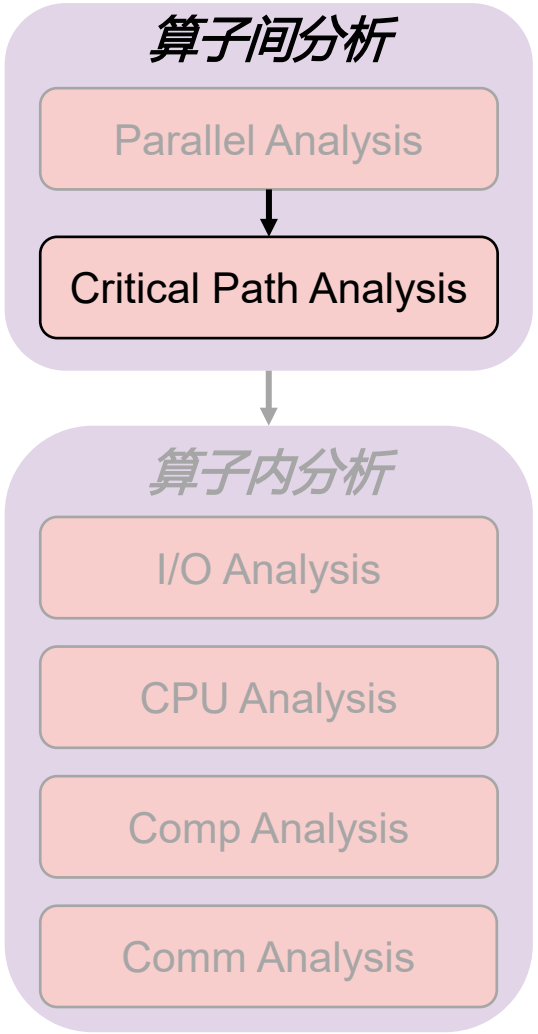
算子间分析



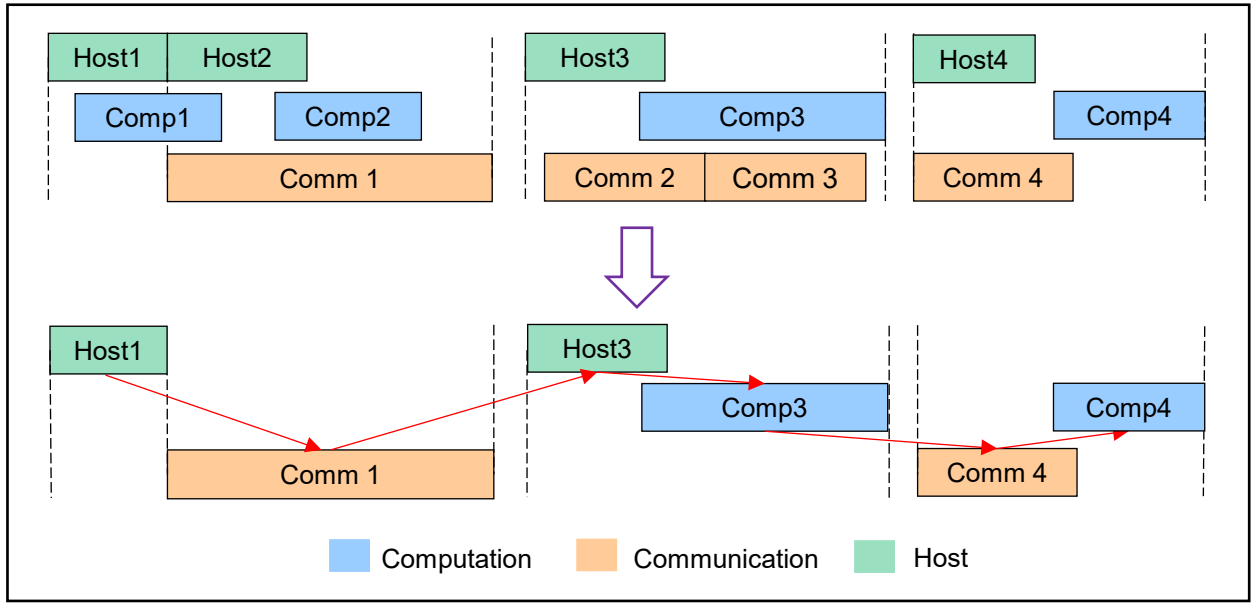
- 多组件并行分析
  - 展示计算、通信、Host任务间的并行度
  - 确定是否存在并行瓶颈



算子间分析



- 关键路径分析
  - 确定关键路径，找出耗时最久的瓶颈算子



I/O瓶颈分析

算子间分析

Parallel Analysis

Critical Path Analysis

算子内分析

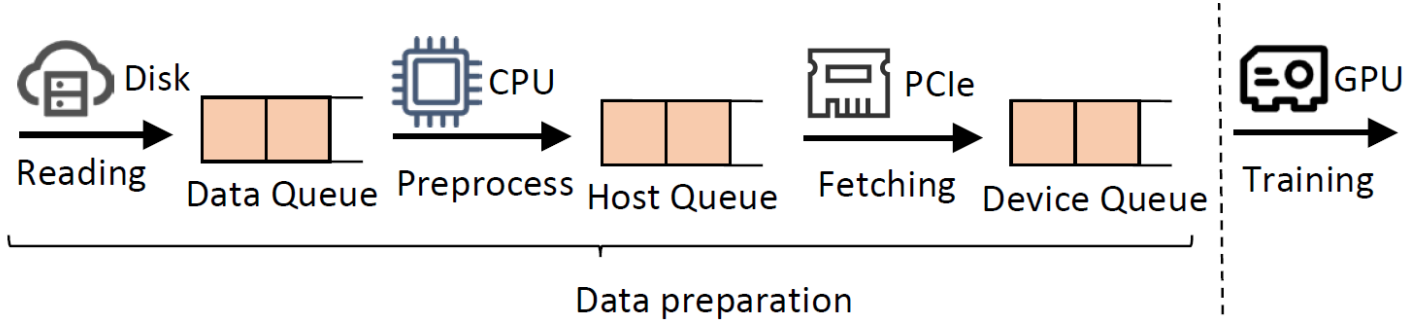
I/O Analysis

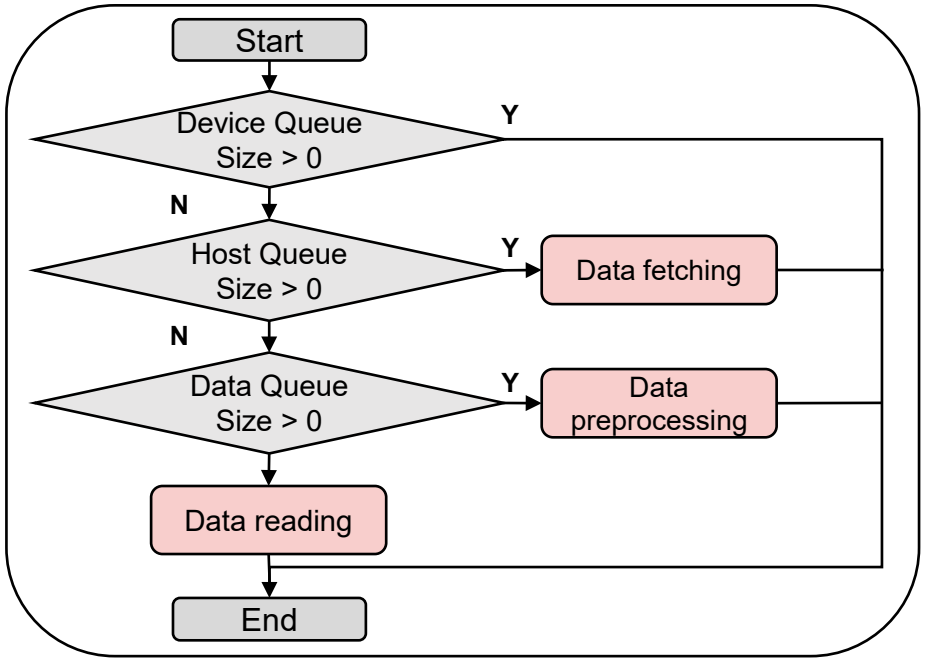
CPU Analysis

Comp Analysis

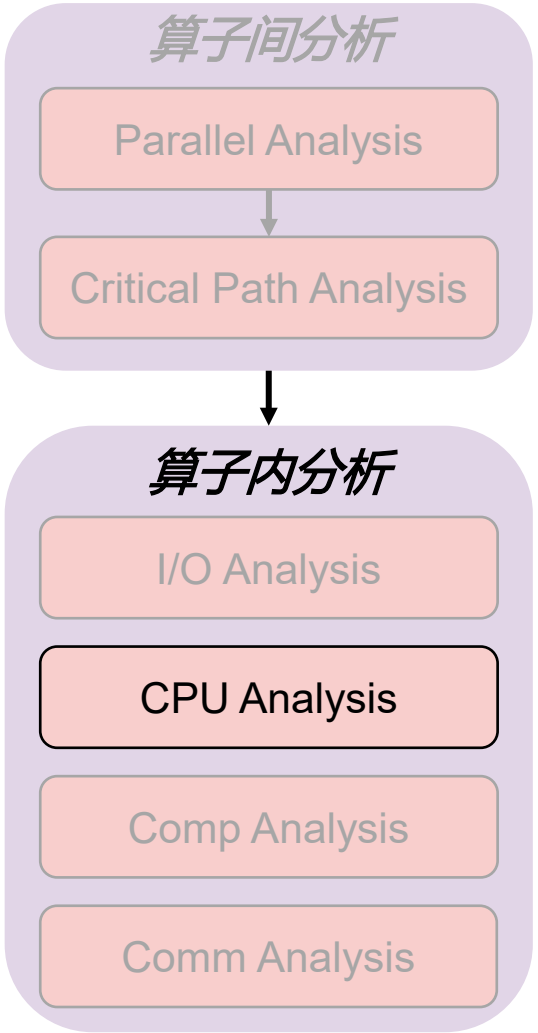
Comm Analysis

- 基于队列的 I/O 分析

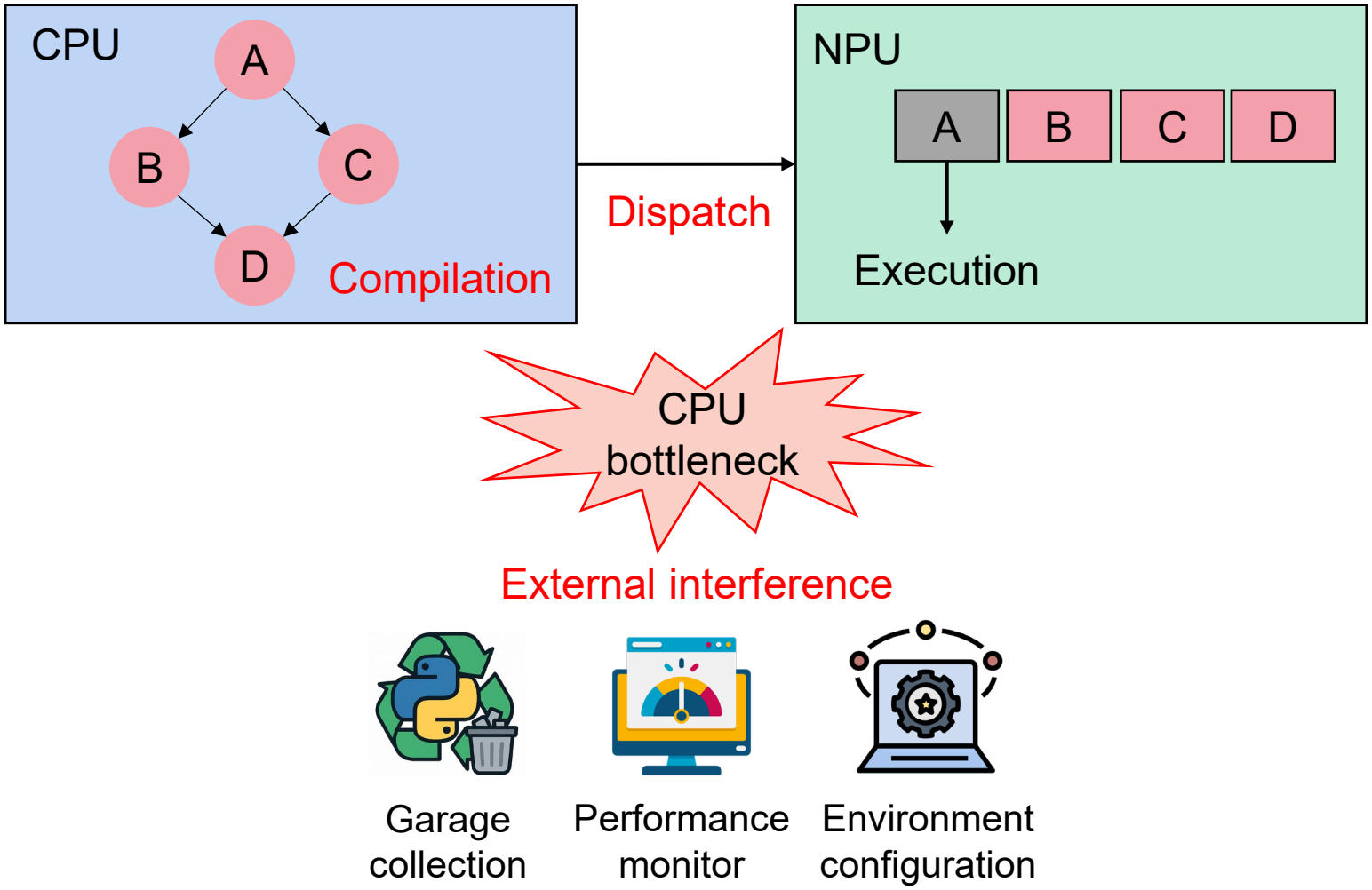




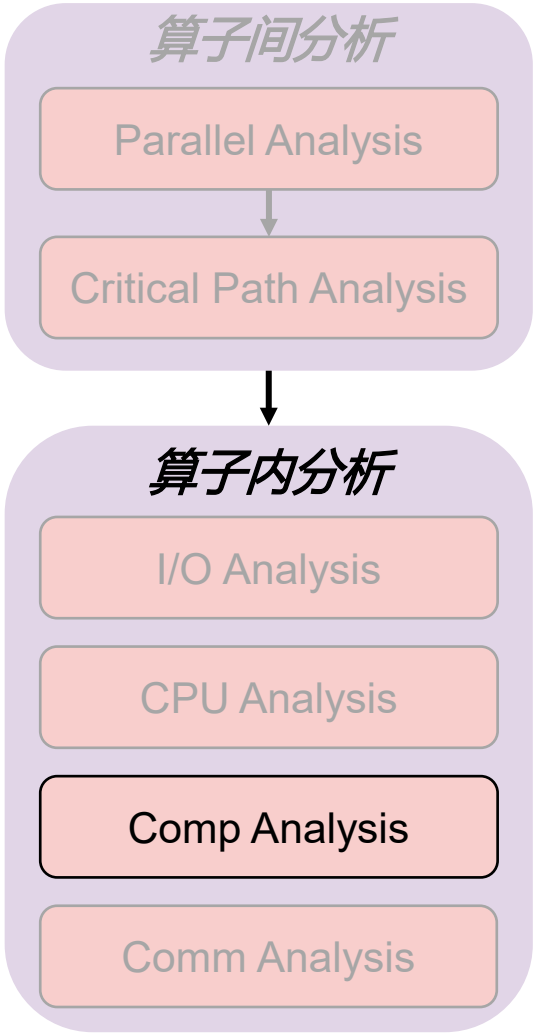
CPU瓶颈分析



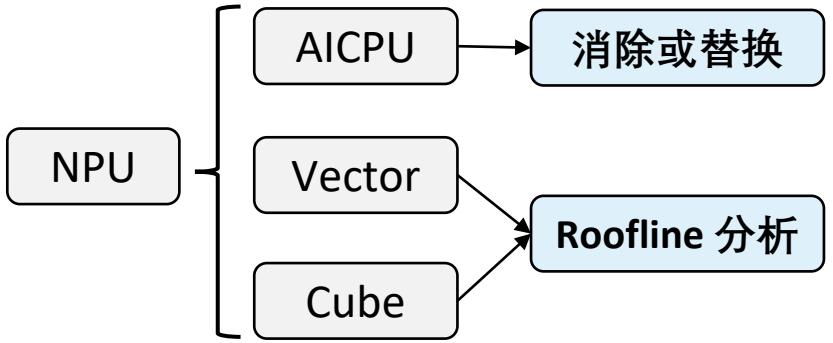
- CPU瓶颈

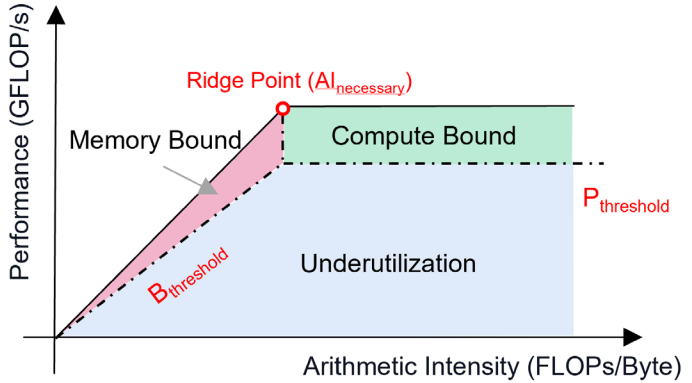


计算瓶颈分析

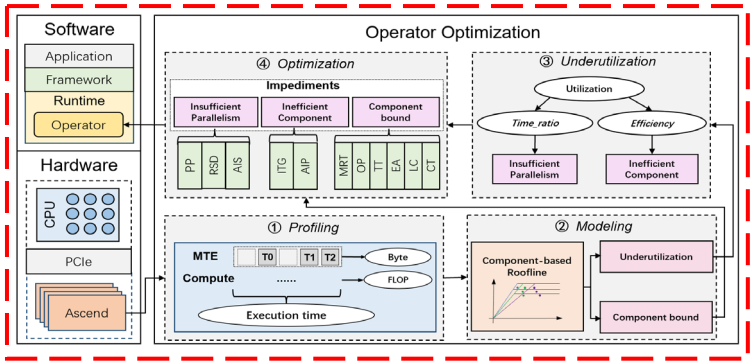


- 计算瓶颈
  - 不同的计算单元（AICPU, AICore Cube/Vector）
  - Roofline分析（arithmetic, memory）



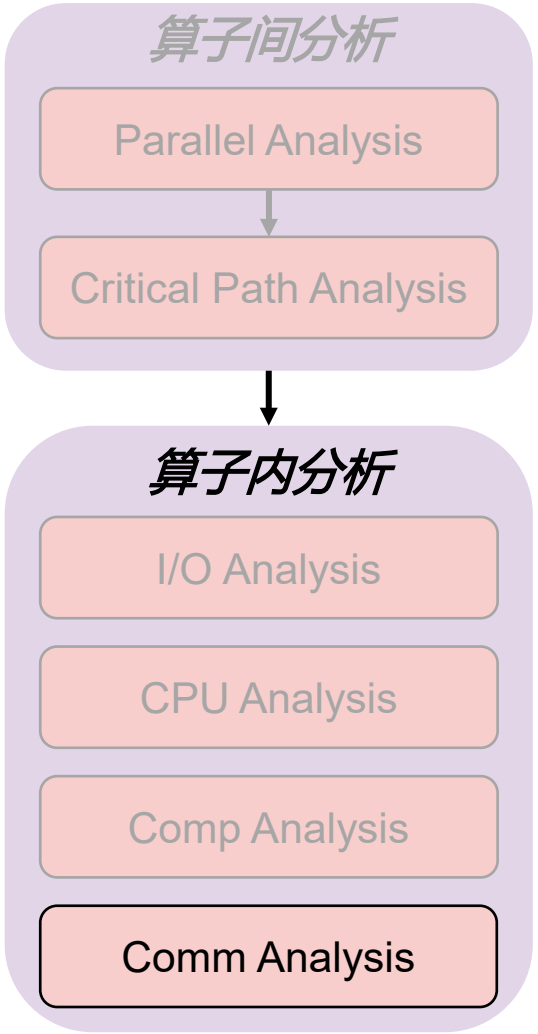


面向Ascend架构的AI算子性能建模与优化[1]
(APLOS 25)

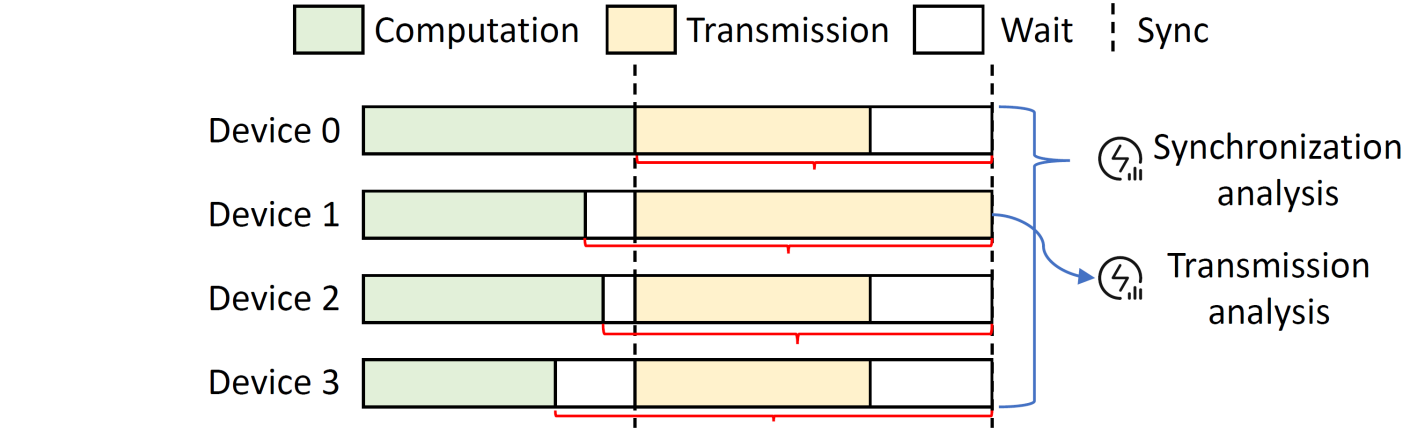


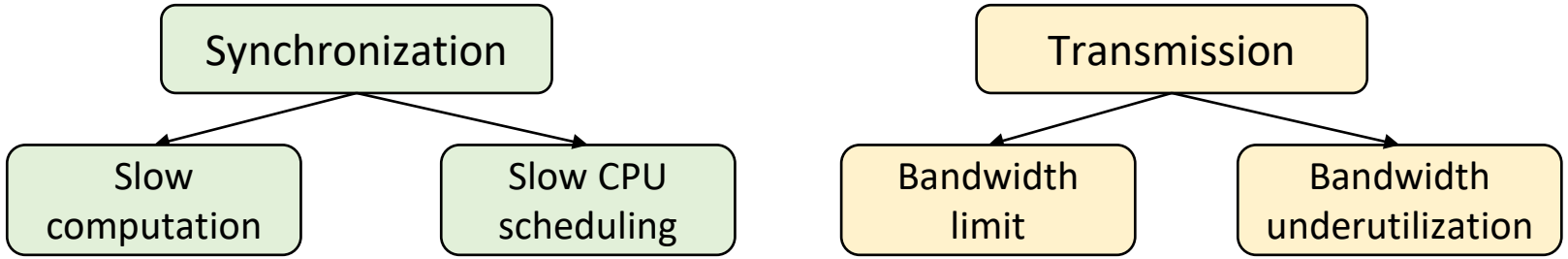
[1] Squeezing Operator Performance Potential for the Ascend Architecture. The 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '25).

通信瓶颈分析

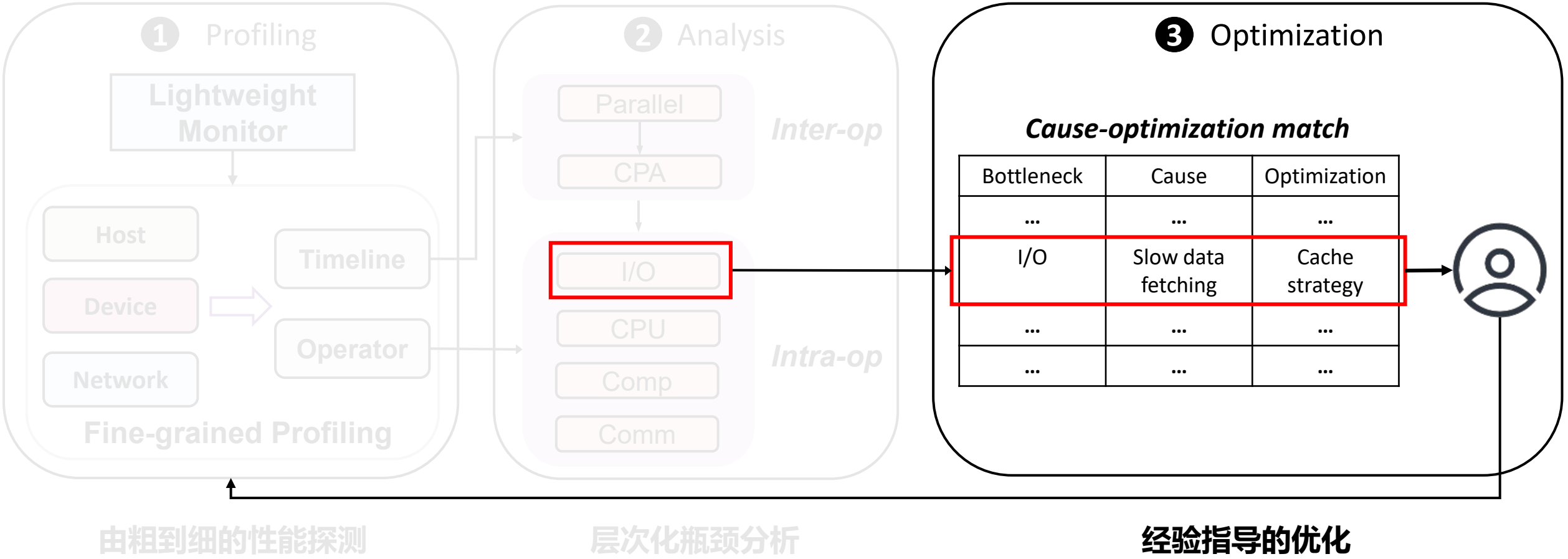


- 通信瓶颈包含同步分析和传输分析两阶段



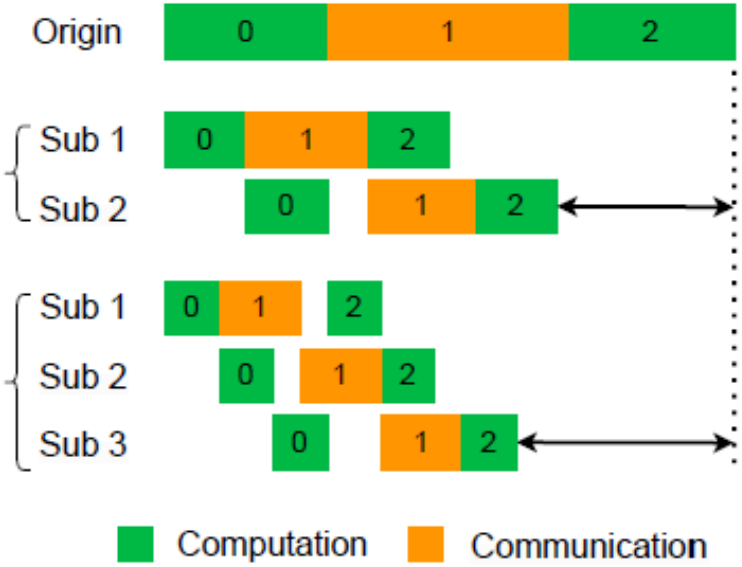


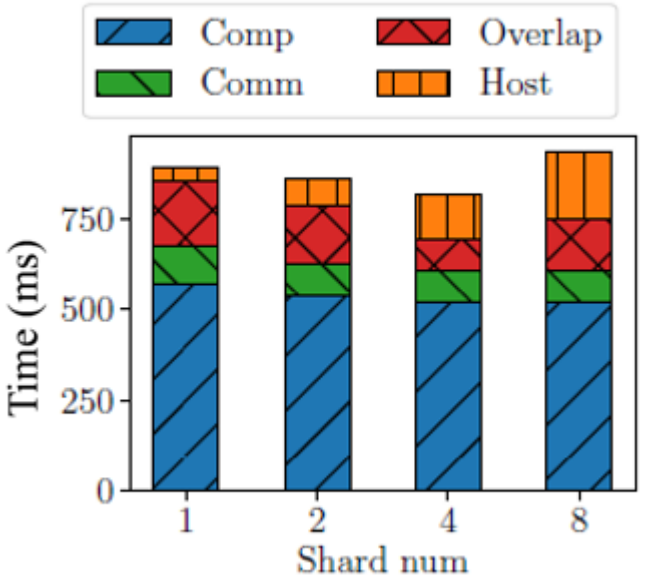
Hermes系统


1 Profiling
Lightweight Monitor
Host
Device
Network
Timeline
Operator
Fine-grained Profiling
2 Analysis
Parallel
CPA
Inter-op
I/O
CPU
Comp
Comm
Intra-op
3 Optimization
Cause-optimization match
Bottleneck
Cause
Optimization
...
I/O
Slow data fetching
Cache strategy
由粗到细的性能探测
层次化瓶颈分析
经验指导的优化

并行瓶颈优化——GPT3

- 并行瓶颈
  - 计算和通信间重叠部分仅占 4.28%。
  - 细粒度并行优化[1][2]，分片数目为 2、4、8 时，加速比分别为 1.04、1.08、0.95 倍。

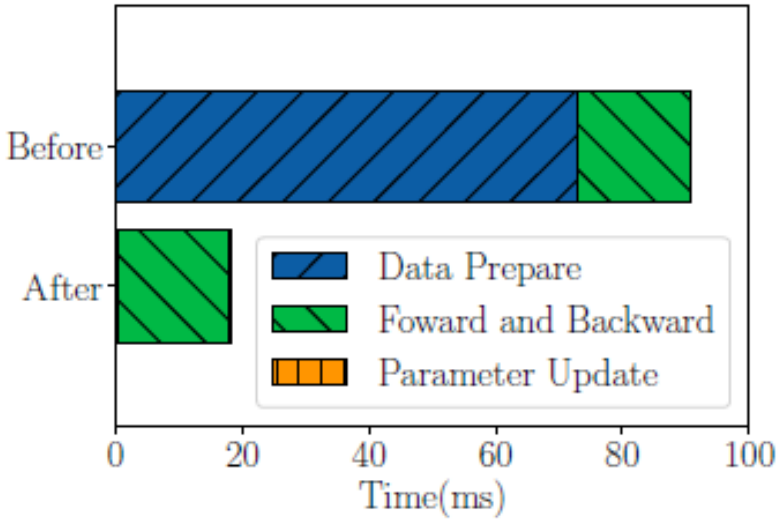




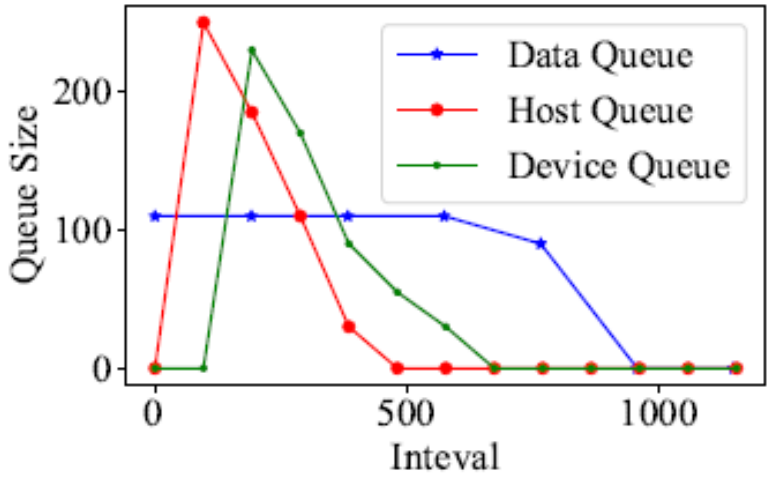
[1] Shibo Wang, etc al. Overlap communication with dependent computation via decomposition in large deep learning models, ASPLOS 2023.
[2] 多副本并行，https://www.mindspore.cn/docs/zh-CN/master/model_train/parallel/multiple_copy.html，2024.

I/O瓶颈优化——ResNet50

- I/O瓶颈
  - 数据准备占比 80.9%，队列分析表明数据预处理过慢。
  - 数据预处理的CPU线程数（num_worker）从 1 提升到 12，实现了 5.34 倍加速。



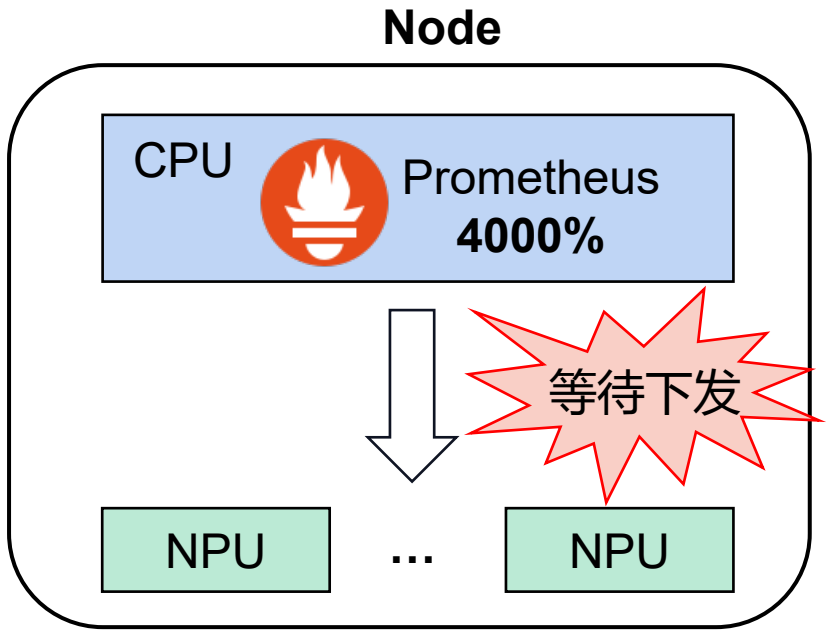
(a) Time breakdown.



(b) Queue size distribution.

CPU瓶颈优化——GPT-3

- CPU瓶颈:
  - 单机8卡GPT-3训练出现显著的性能波动，单步时间最高达到 3027.6 ms。
  - 瓶颈源于服务器性能监控的 Prometheus 插件[3]，消耗了 4000% 的 CPU 资源。



	平均单步时间	性能波动超过10%占比
优化前	444.40 ms	2.57%
优化后	374.88 ms	0.08%

[3] Prometheus. https://prometheus.io/docs/introduction/overview，2024.

计算瓶颈优化——PanGu-α

- 计算瓶颈
  - Roofline分析表明利用率不足算子达 61.48%。
  - 通过高性能算子、算子融合、减少格式转换等优化，利用率不足的算子降至 46.32%。

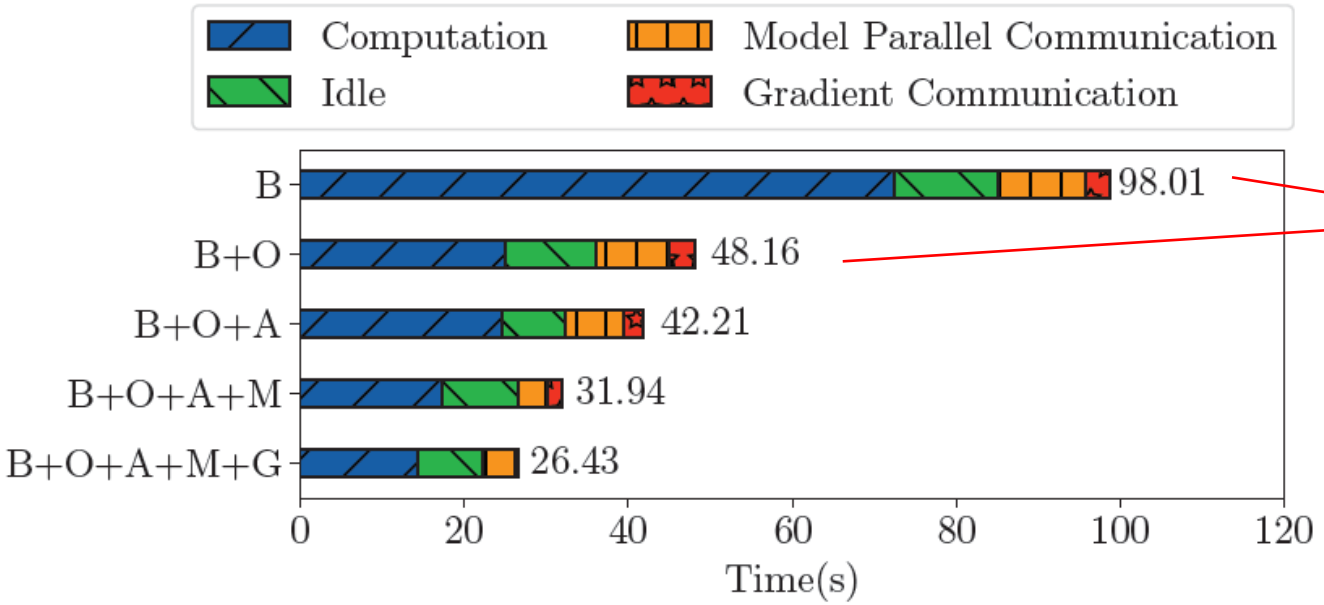


Figure 9. PanGu-α single iteration time.

总训练时间加速 2.05 倍

通信瓶颈优化——VGG16

- 通信瓶颈
  - 未掩盖的 AllReduce 通信占比 28.4%。
  - 同步分析：瓶颈源自 Rank 7 的传输较慢。
  - 传输分析：HCCS 带宽利用率不足，由于通信粒度过小。

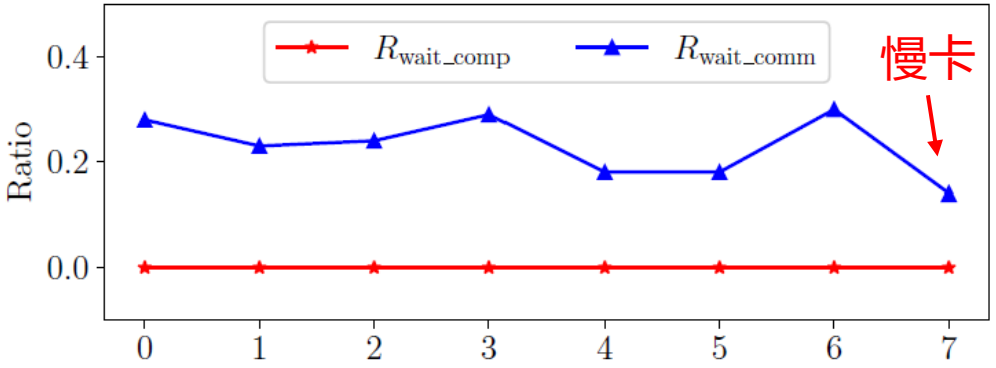


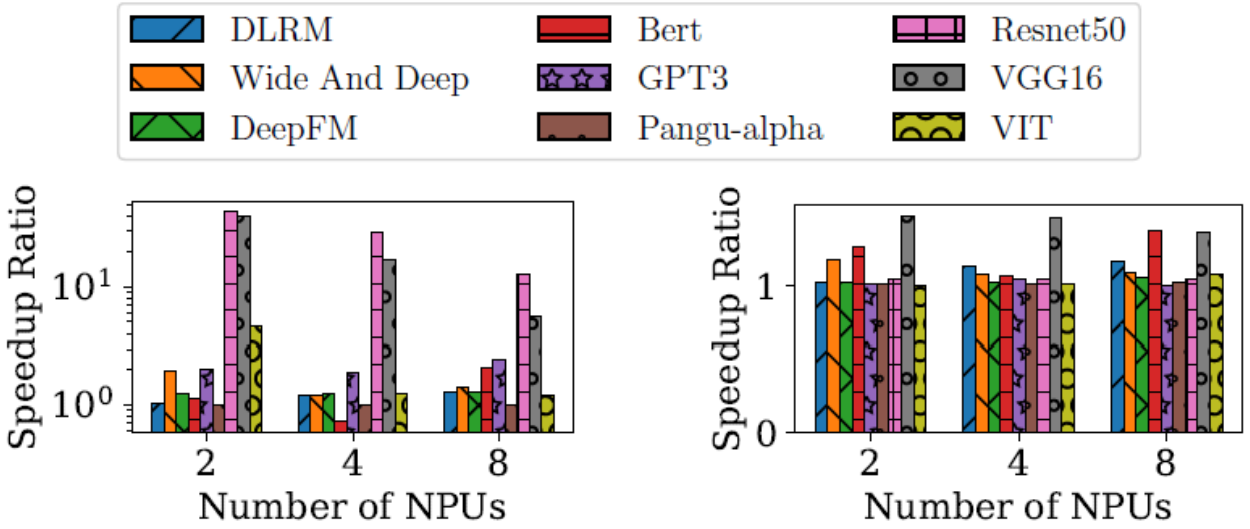
Figure 12. Synchronization analysis.

Table 4. Transmission analysis.

LinkType	Profiling data		Bandwidth (GB/s)		Packet size (MB)	
	Size (MB)	Time (ms)	Actual	Peak	Actual	Full
HCCS	179.37	18.45	9.50	18.00	12.81	32.00
PCIe	0.00	0.00	0.00	20.00	0.00	32.00
RDMA	0.00	0.00	0.00	12.50	0.00	0.50

通信瓶颈优化——VGG16

- 通信优化
  - 经过梯度融合优化[4]，VGG16的单步时间加速了 1.35倍。
  - 如果 AllReduce 通信不是瓶颈，梯度融合优化效果有限。



(a) Non-overlapping communication. (b) Overall of one iteration.

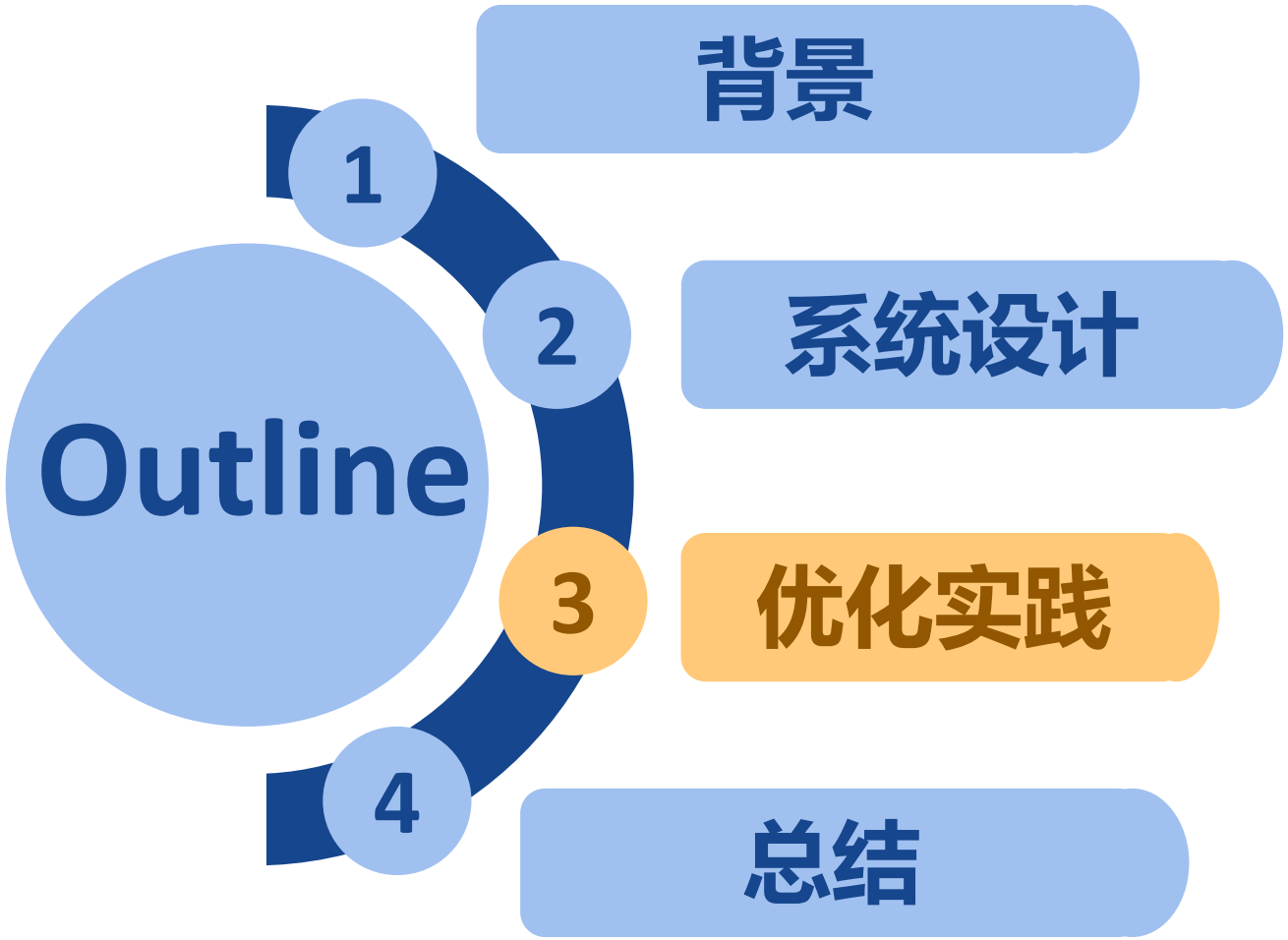
Figure 13. Speedup of communication optimization.

[4] Horovod. Tensor fusion. https://horovod.readthedocs.io/en/stable/tensor-fusion_include.html，2024.

135个实际案例的启示

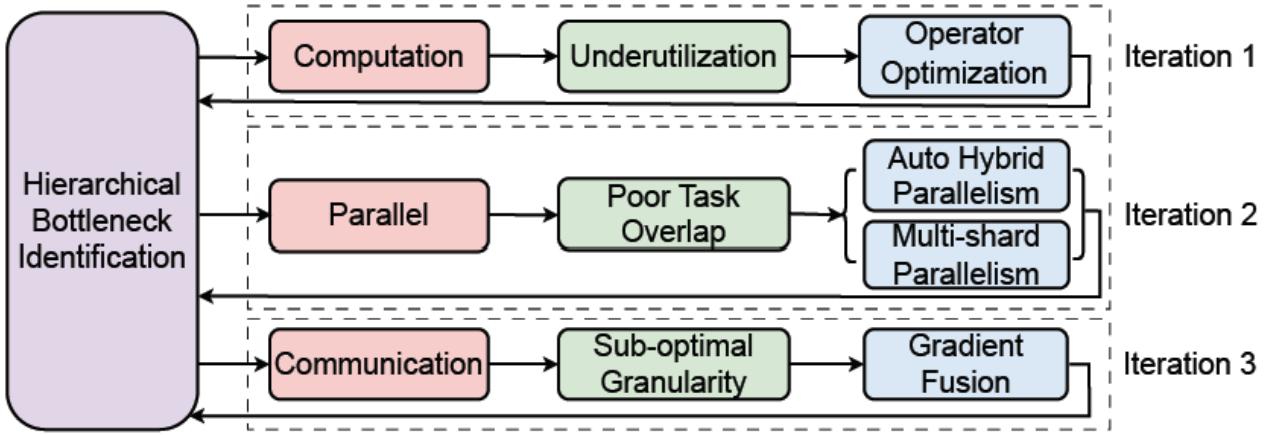
Bottleneck	Cause	Optimization	Ratio
Parallel	Poor Parallelism	Multi-shard Parallel/Auto Hybrid Parallel [58]	5.2%
I/O	Slow Data Reading	Increase I/O bandwidth/Remote to local storage	22.1%
	Slow Data Preprocessing	Improve CPU parallelism (num_workers)	
		Avoid compression formats (zip, tar)	
		Cancel the taskset process binding [28]	
	Slow Data Fetching	Cache strategy (pin_memory, data prefetcher) [19, 57]	
CPU	Operator Complication	Disable JIT compilation/Replace affinity API	44.1%
	Operator Dispatch	Eliminate synchronization stream (sync operators or functions)	
	Garbage Collection	Disable gc/increase gc threshold	
	CPU Resources Contention	Disable other CPU process	
	Environment Configuration	Align software versions/reduce log level	
Computation	Compute Bound	Avoid decreasing computing frequency/isolate slow nodes	18.2%
	Memory Bound	Adjust memory environment variables/reduce batch_size/ZeRO [46, 47]	
	Underutilization	Replace high-performance operator	
		Operator fusion[38, 59]	
		Eliminate dynamic shape operator/forbid private format	
Communication	Bandwidth Contention	Avoid bandwidth contention between comp and comm	10.4%
	RDMA Retransmission	Adjust communication library environment variable	
	Small Granularity	Tensor fusion/partition [21, 41]	
	Network Configuration	Check switch configuration/address UDP port hashing collision	

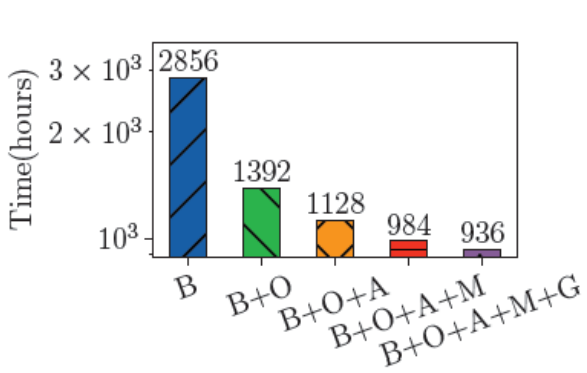
模型部署中CPU瓶颈占据主流！


Outline
1
背景
2
系统设计
3
优化实践
4
总结

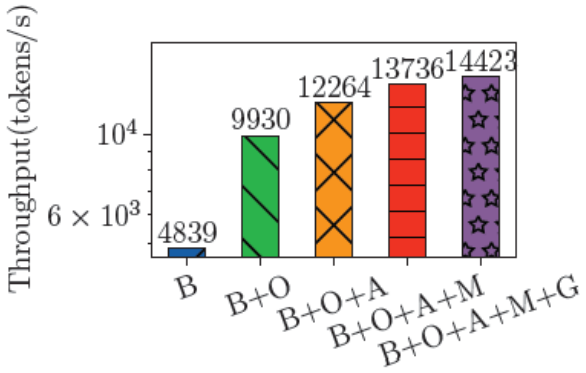
开发者：PanGu-α的迭代式分析和优化

- 100B PanGu-α 模型经过长达1年的多轮迭代优化





(a) Overall training time.



(b) Overall training throughput.

总训练时间加速 3.05 倍

部署者：从GPU到NPU的模型部署优化

- 模型部署优化
  - 8 卡训练 MobileNetV1-SSD，Ascend 吞吐量仅为 A800 的 43%，优化CPU瓶颈后达到 90%。
  - 完整结果：

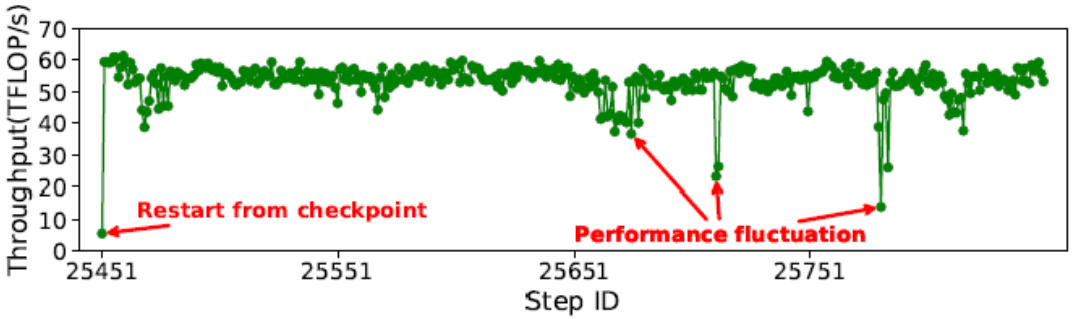
Table 5. Model deployment optimization results

Type	Model	Parameter	Optimization Speedup (-: not optimizable)						# of NPUs	Dataset
			I/O	CPU	Para.	Compu.	Comm.	Total		
Vision	ResNet50	25.6M	5.03	-	-	1.02	1.04	5.34	8	ImageNet2012
	VGG16	138.4M	-	-	-	1.08	1.35	1.46		
	MobileNetV1-SSD	4.2M	-	1.37	-	-	-	1.37	1	VOC2012
			1.08	1.91	-	-	-	2.07	8	
NLP	Bert-Large	330M	-	-	-	1.63	1.38	2.49	8	Wiki
	PanGu-α	1.3B	-	-	-	1.18	1.02	1.20		
	GPT3-13B	13B	-	-	1.08	-	-	1.08		
Recommend	DeepFM	16.5M	-	-	-	-	1.08	1.08	8	Criteo
	DLRM	540M	-	-	-	-	1.17	1.17		

1.08到5.34倍训练加速

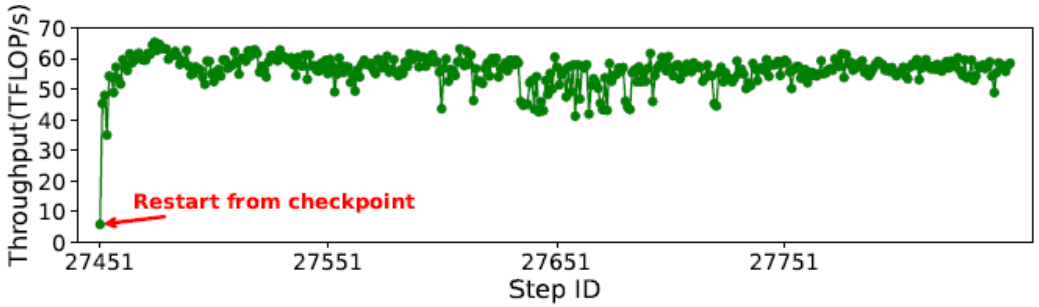
维护者：训练性能波动优化

- 9千卡 MoE 模型训练CPU瓶颈



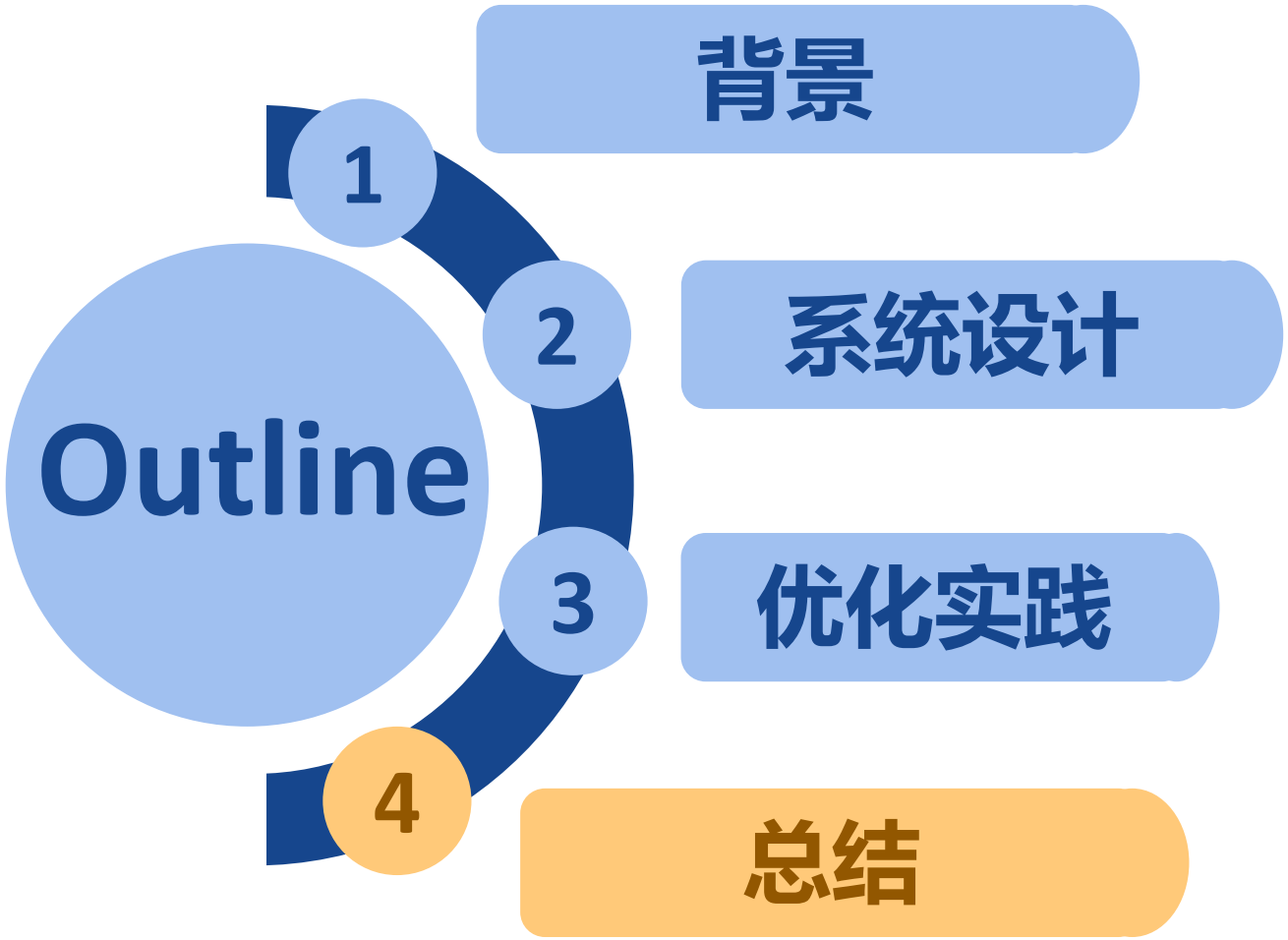
(a) Performance before optimization.

适当提升Python垃圾回收阈值
在保存检查点时主动进行垃圾回收

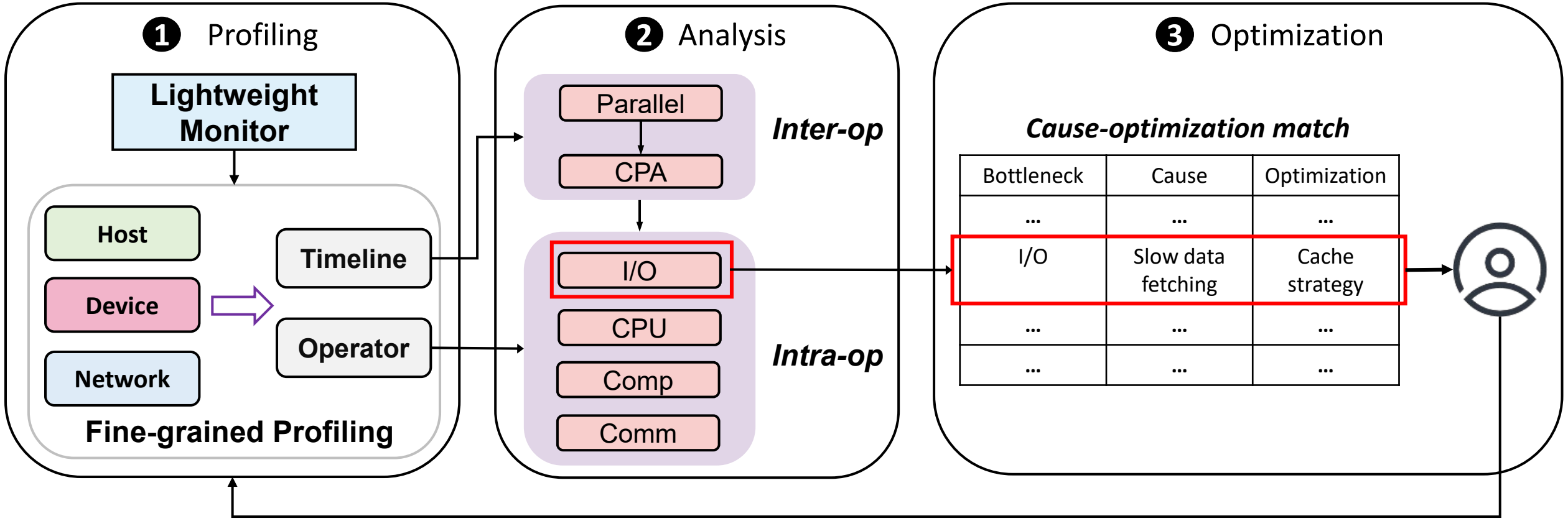


(b) Performance after optimization.

训练时间加速 1.06 倍
平均吞吐量提升 1.05 倍


Outline
1
背景
2
系统设计
3
优化实践
4
总结

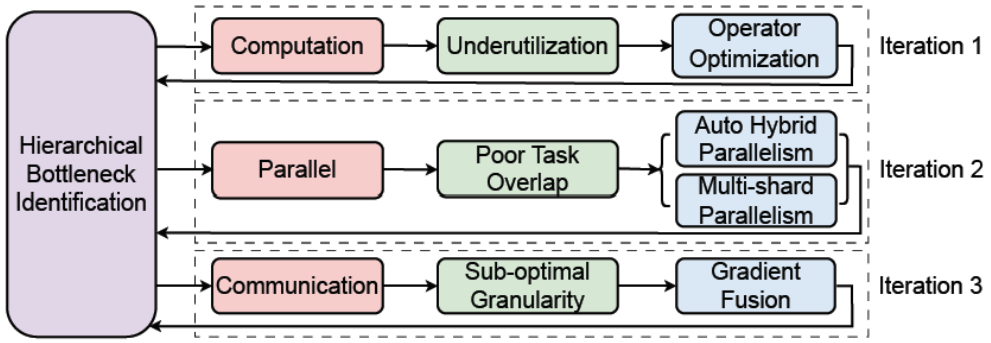
Hermes系统


1 Profiling
Lightweight Monitor
Host
Device
Network
Timeline
Operator
Fine-grained Profiling
2 Analysis
Parallel
CPA
Inter-op
I/O
CPU
Comp
Comm
Intra-op
3 Optimization
Cause-optimization match
Bottleneck
Cause
Optimization
...
I/O
Slow data fetching
Cache strategy

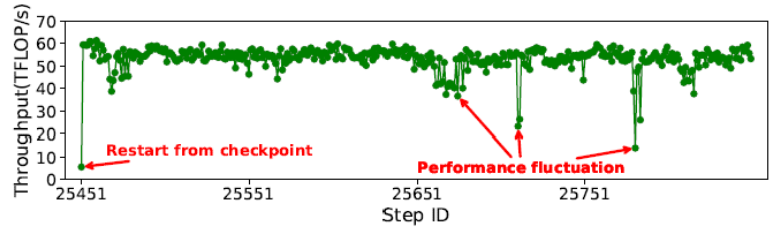
125个实际案例总结

Bottleneck	Cause	Optimization	Ratio
Parallel	Poor Parallelism	Multi-shard Parallel/Auto Hybrid Parallel [58]	5.2%
I/O	Slow Data Reading	Increase I/O bandwidth/Remote to local storage	22.1%
	Slow Data Preprocessing	Improve CPU parallelism (num_workers)	
		Avoid compression formats (zip, tar)	
		Cancel the taskset process binding [28]	
	Slow Data Fetching	Cache strategy (pin_memory, data prefetcher) [19, 57]	
CPU	Operator Complication	Disable JIT compilation/Replace affinity API	44.1%
	Operator Dispatch	Eliminate synchronization stream (sync operators or functions)	
	Garbage Collection	Disable gc/increase gc threshold	
	CPU Resources Contention	Disable other CPU process	
	Environment Configuration	Align software versions/reduce log level	
Computation	Compute Bound	Avoid decreasing computing frequency/isolate slow nodes	18.2%
	Memory Bound	Adjust memory environment variables/reduce batch_size/ZeRO [46, 47]	
	Underutilization	Replace high-performance operator	
		Operator fusion[38, 59]	
		Eliminate dynamic shape operator/forbid private format	
Communication	Bandwidth Contention	Avoid bandwidth contention between comp and comm	10.4%
	RDMA Retransmission	Adjust communication library environment variable	
	Small Granularity	Tensor fusion/partition [21, 41]	
	Network Configuration	Check switch configuration/address UDP port hashing collision	

不同角色的优化实践

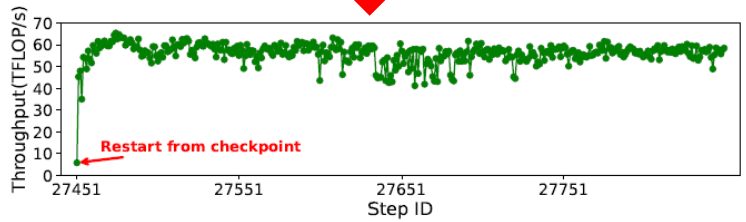


总训练时间加速 3.05 倍



(a) Performance before optimization.

训练时间加速 1.06 倍
平均吞吐量提升 1.05 倍



(b) Performance after optimization.

Table 5. Model deployment optimization results

Type	Model	Parameter	Optimization Speedup (-: not optimizable)						# of NPUs	Dataset
			I/O	CPU	Para.	Compu.	Comm.	Total		
Vision	ResNet50	25.6M	5.03	-	-	1.02	1.04	5.34	8	ImageNet2012
	VGG16	138.4M	-	-	-	1.08	1.35	1.46		
	MobileNetV1-SSD	4.2M	-	1.37	-	-	-	1.37	1	VOC2012
			1.08	1.91	-	-	-	2.07	8	
NLP	Bert-Large	330M	-	-	-	1.63	1.38	2.49	8	Wiki
	PanGu-α	1.3B	-	-	-	1.18	1.02	1.20		
	GPT3-13B	13B	-	-	1.08	-	-	1.08		
Recommend	DeepFM	16.5M	-	-	-	-	1.08	1.08	8	Criteo
	DLRM	540M	-	-	-	-	1.17	1.17		

1.08到5.34倍训练加速

大模型推理性能建模与自动调优

大模型推理部署过程中面临以下性能挑战：

- **优化目标多样且动态**：
  - 推理系统需在延迟与吞吐之间权衡，不同业务场景下的优化目标各异，缺乏通用方案。
- **影响因素复杂且高度耦合**：推理性能受多方面影响，难以分析定位性能瓶颈：
  - 负载特征（如输入/输出长度分布、请求数量等）
  - 模型与量化（模型结构、参数规模、量化精度等）
  - 系统与硬件配置（加速器类型与数量、内存、带宽等）
  - 软件与算法策略（并行、批处理、KV缓存、算子融合、分离部署等）
- **现有调优方法低效且成本高**：
  - 业界多依赖专家经验手动调优，参数组合空间巨大，验证周期长，难以适应动态变化。

适应多场景需求、自动识别性能瓶颈、智能搜索最优配置。

已有工作

- **推理性能建模**
  - Roofline模型或ML-based预测
  - 硬件无关的性能预测
  - 特定硬件建模（Tensor Core）
- **LLM推理性能优化**
  - 并行策略：DP/PP/TP，MoE专家并行
  - KV缓存管理：Paged Attention
  - 批处理调度：Chunked Prefill
  - 分离式部署：PD分离、AF分离

性能分析不够全面准确，且不通用

优化种类繁多，用户难以选择

Thanks!

Q&A

yuhangzhou@smail.nju.edu.cn