流水并行分享

汇报人：郑天宇

时间：2024. 11. 22

目录

01

广义流水并行

什么是流水线？

第一条流水线使平均每辆T型汽车的组装时间：

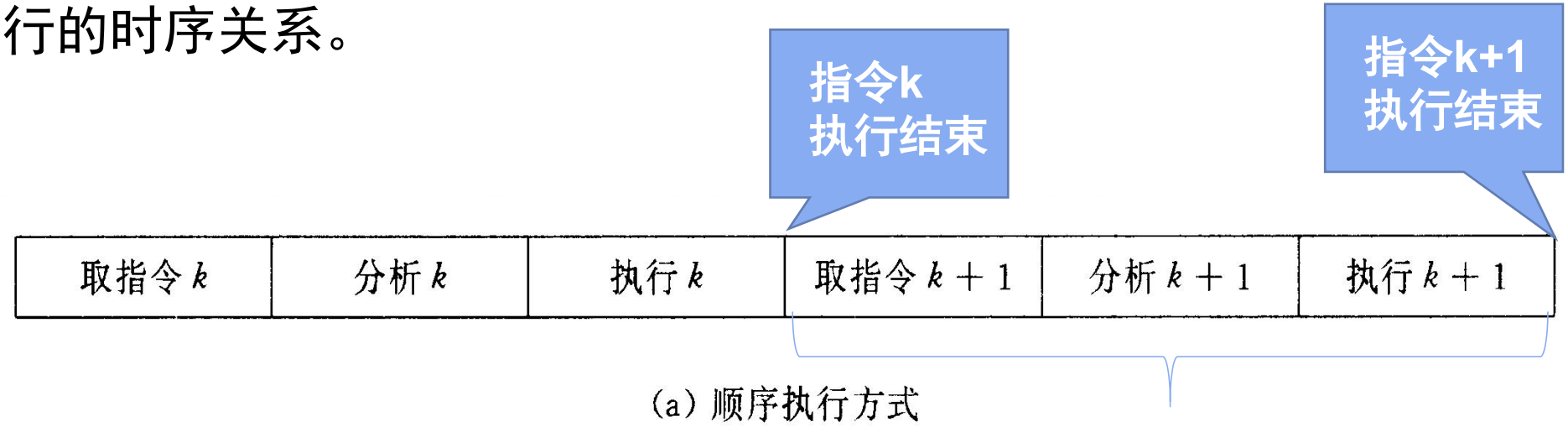
12小时28分钟 → 10秒钟

生产效率提高了4488倍！

1913年福特密歇根工厂移动装配线（84 个步骤）

操作系统中的流水并行

- 例：假定一条指令的执行分为三个阶段：取指令、分析、执行。下图是串行指令执行的时序关系。

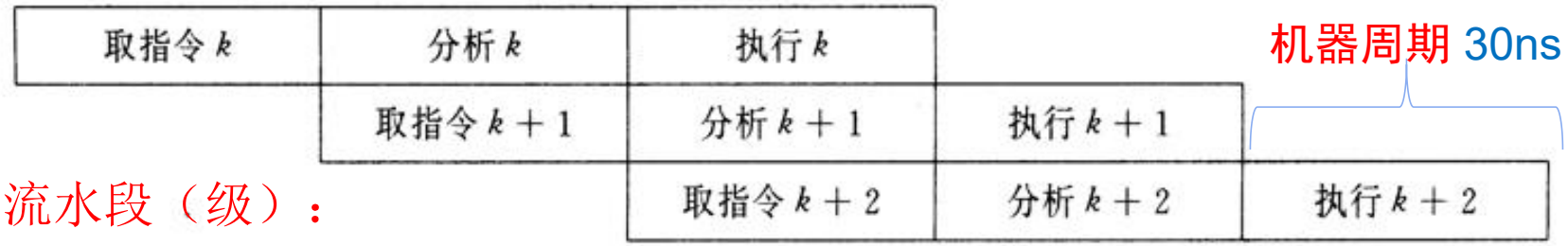


(a) 顺序执行方式

如果假设平均每条指令的执行时间是90ns，即平均每隔90ns就执行完一条指令。

3条指令执行时间：90*3= 270ns

操作系统中的流水并行



流水段（级）：

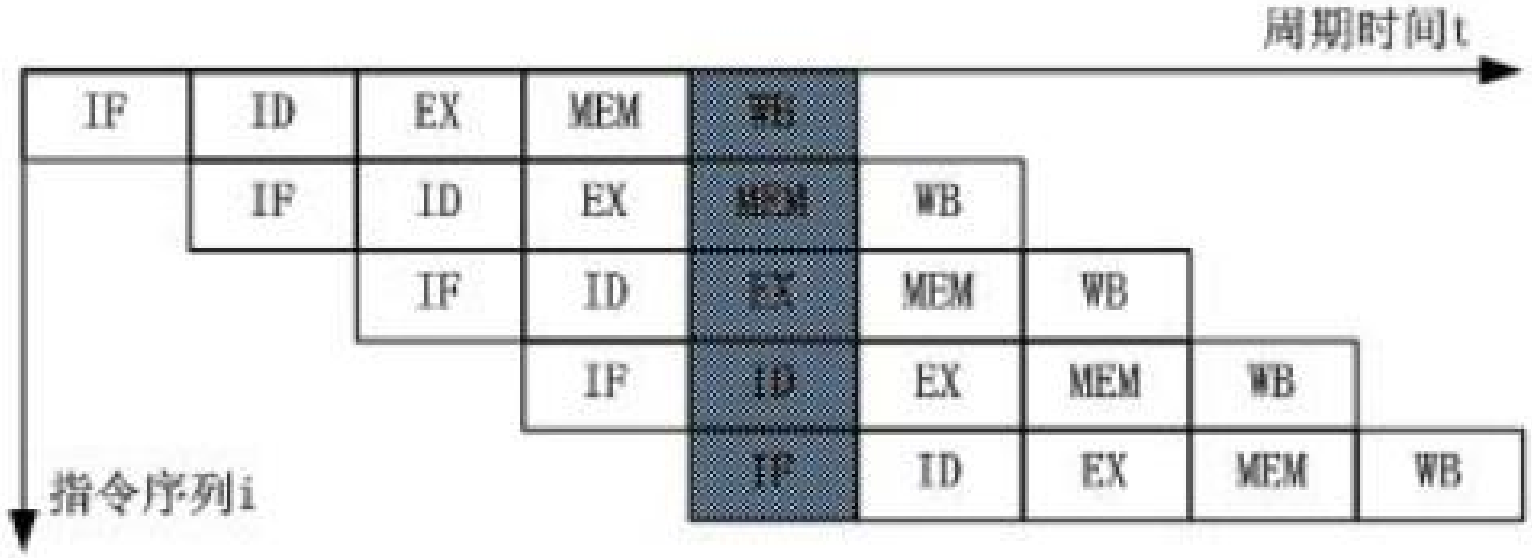
完成一条指令的

一部分操作

平均每条指令的执行时间同样是90ns，但是3条指令执行时间：30*5= 150ns

流水并行性能分析的重要工具

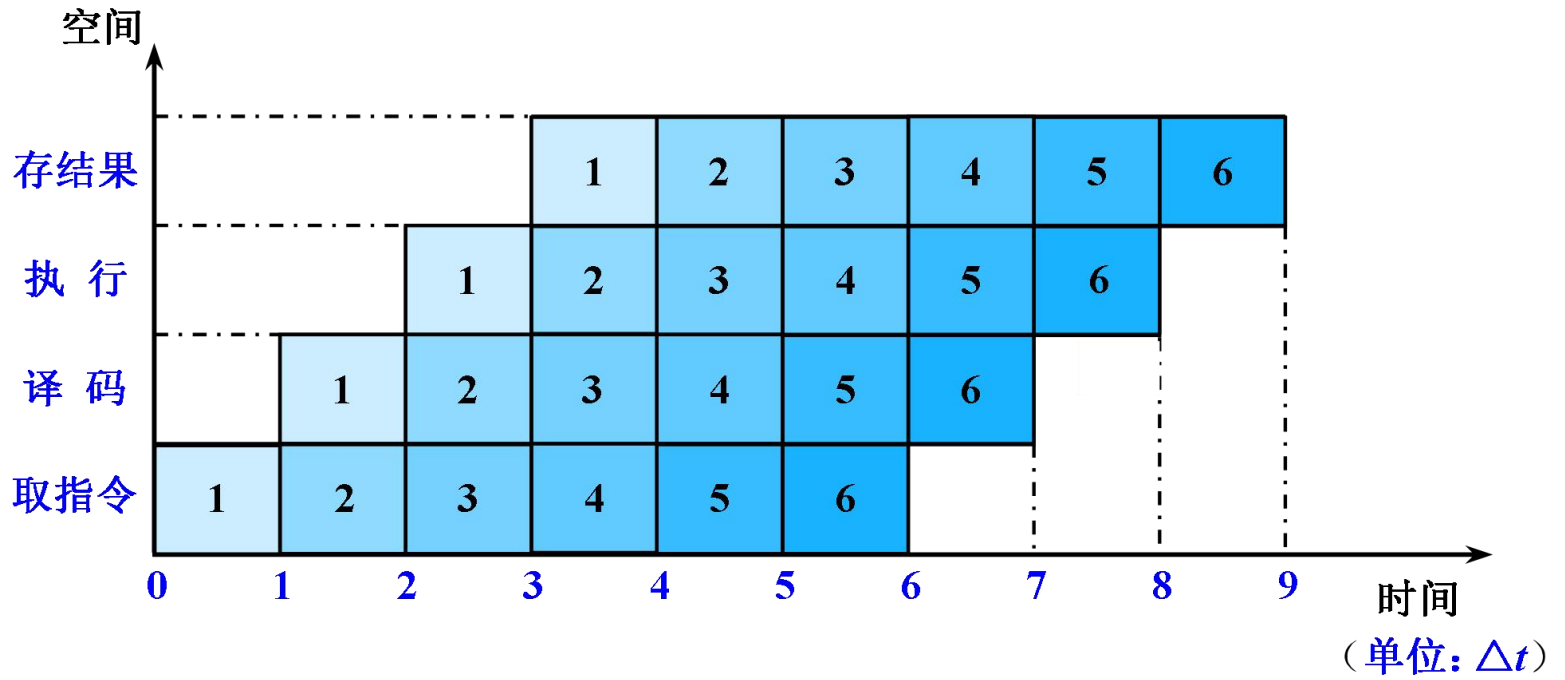
时序图



流水并行性能分析的重要工具

时空图

4段指令流水线的时空图



流水并行分类

流水线按各过程段用时是否相等可分为均匀流水线和非均匀流水线两种。

1） 均匀流水线指的是各过程段用时全相等的流水线

2） 非均匀流水线指的是各过程段用时不全相等的流水线

均匀流水线的性能分析

- 流水线完成n个连续任务所需要的总时间为

（假设一条k段线性流水线）

$$T_k = k\Delta t + (n-1)\Delta t = (k+n-1)\Delta t$$

- 流水线的实际吞吐率

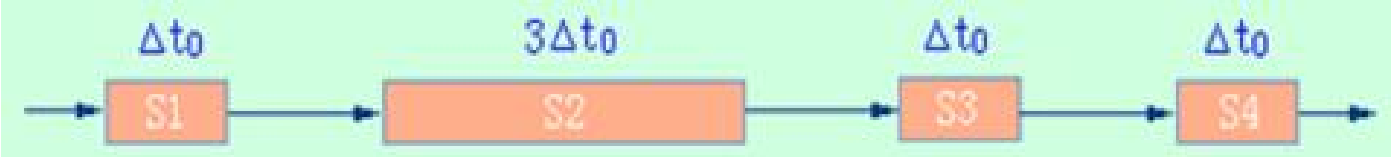
$$TP = \frac{n}{(k+n-1)\Delta t}$$

最大吞吐率

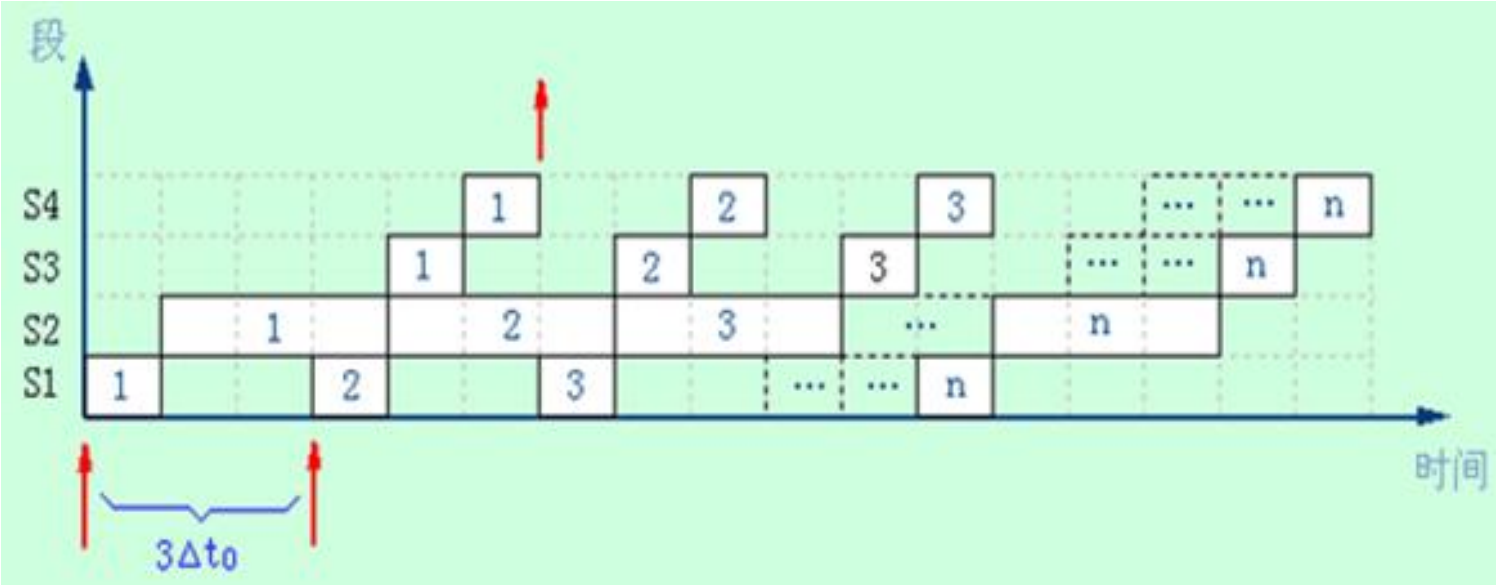
$$TP_{\max} = \lim_{n\to\infty} \frac{n}{(k+n-1)\Delta t} = \frac{1}{\Delta t}$$

非均匀流水线的性能分析

- 非均匀流水线



- 非均匀流水线时空图



非均匀流水线的性能分析

- 各段时间不等的流水线的实际吞吐率：
  - （ Δt_i为第 i段的时间，共有k个段 ）

$$TP = \frac{n}{\sum_{i=1}^{k} \Delta t_i + (n-1)\max(\Delta t_1, \Delta t_2, \cdots, \Delta t_k)}$$

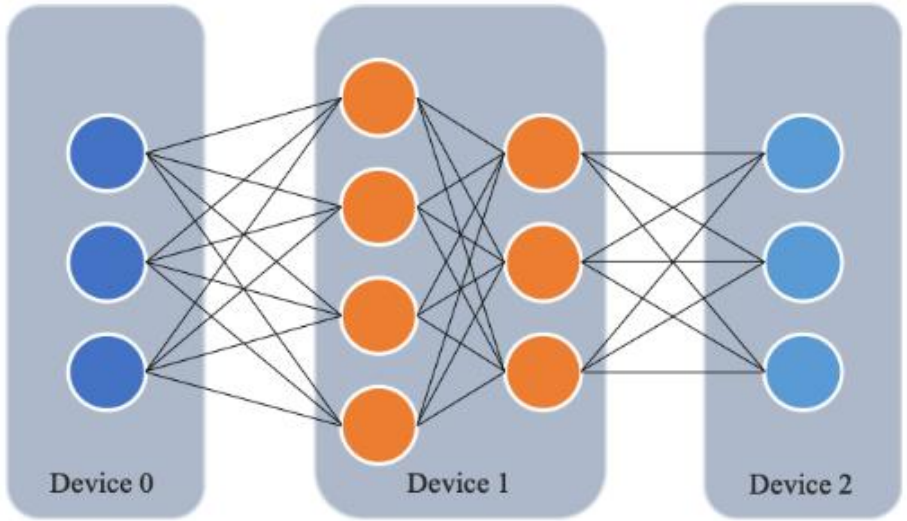
- 流水线的最大吞吐率为

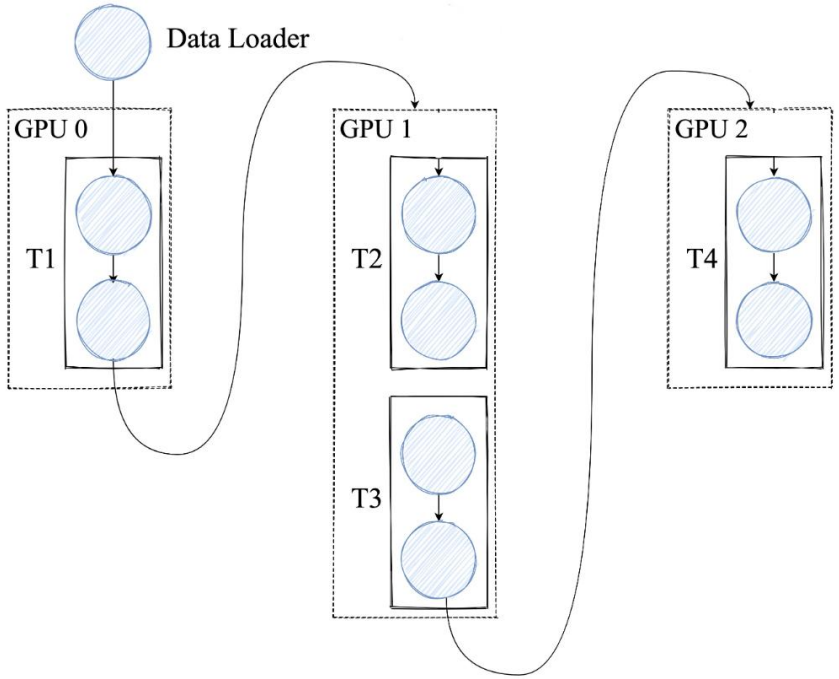
$$TP_{\max} = \frac{1}{\max(\Delta t_1, \Delta t_2, \cdots \Delta t_k)}$$

02

分布式训练流水并行

分布式训练流水并行定义



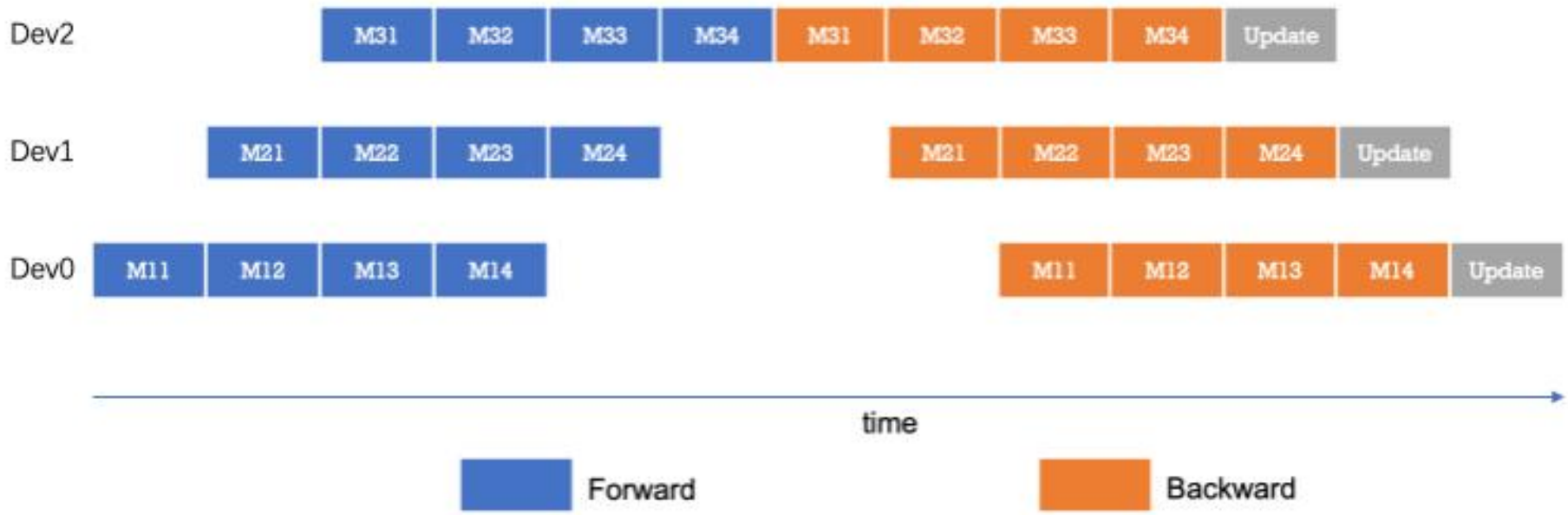


流水并行示例图

分布式流水并行——朴素流水并行

分布式流水并行——微批次流水线并行

FThenB 编排

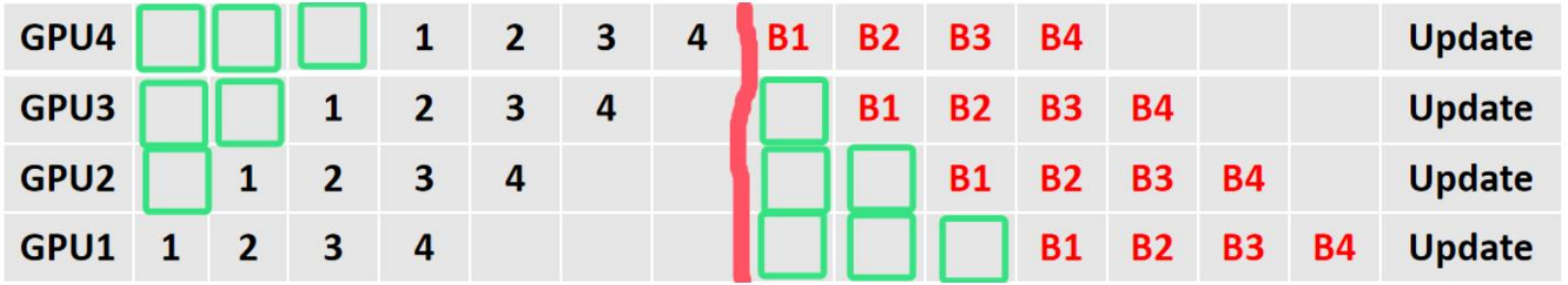


分布式流水并行的评价指标

针对流水并行（PP）假设并行的stage 切分为p 份，micro-batches 为m，每个micro-batches 的前向和后向执行时间为tf 和tb。在理想的环境下，前向和后向的合起来理想的执行时间应该是[2]：

$$t_{ideal} = m(tf + tb)$$

实际状态：



分布式流水并行的评价指标

总的 bubble 占用的时间跟流水并行 PP 切分策略相关

GPU4	1	2	3	1	2	3	4	B1	B2	B3	B4			Update
GPU3	1	2	1	2	3	4			B1	B2	B3	B4		Update
GPU2	1	1	2	3	4					B1	B2	B3	B4	Update
GPU1	1	2	3	4							B1	B2	B3	B4	Update

bubble占用时间：

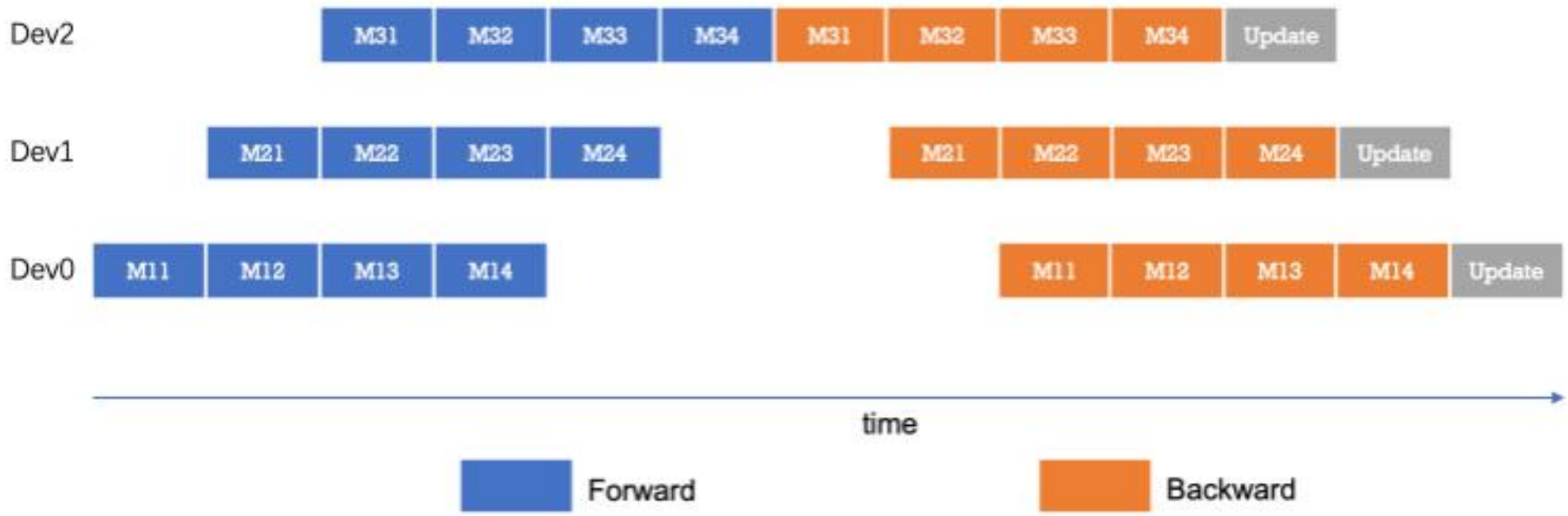
$$t_{bubble} = (p-1)(tf + tb)$$

在流水并行中，气泡花费时间占理想计算时间的比例是：

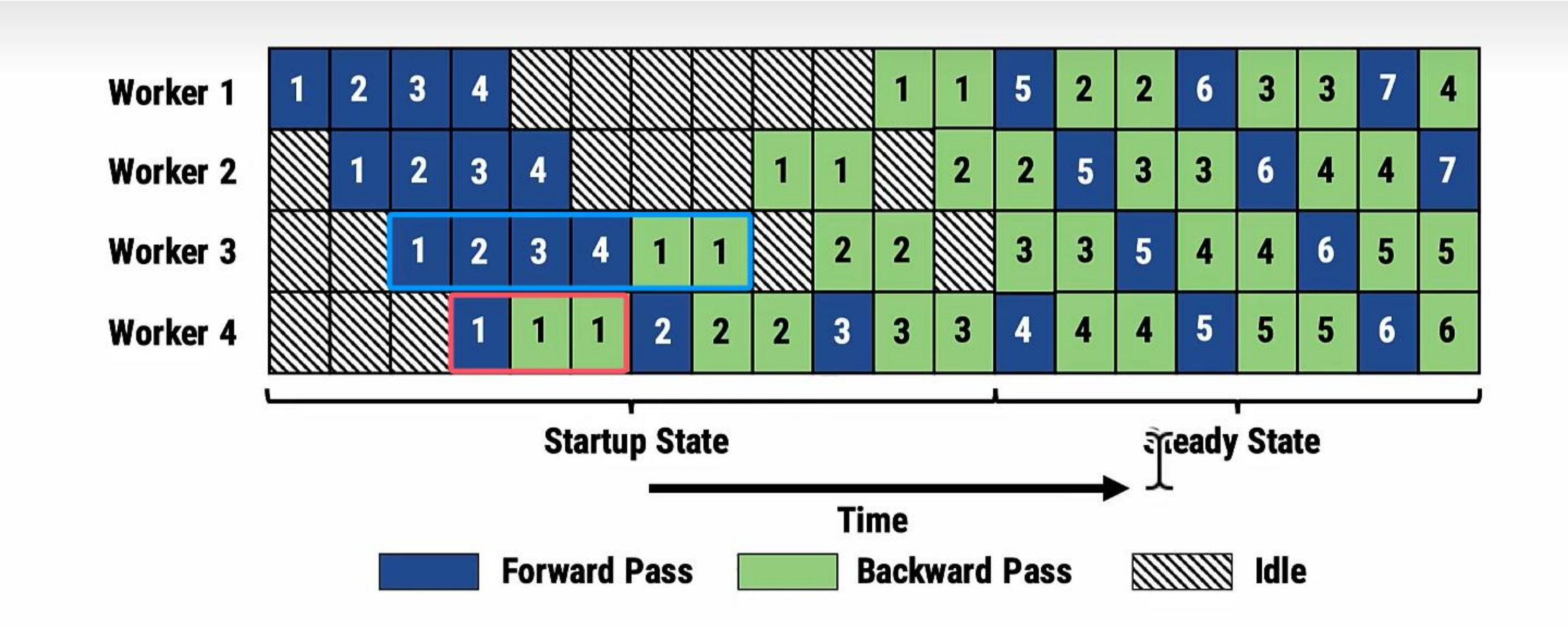
$$\text{Bubble time fraction (pipeline bubble size)} = \frac{t_{pb}}{t_{id}} = \frac{p-1}{m}.$$

分布式流水并行的评价指标

显存的实际利用率



分布式流水并行-激活重计算



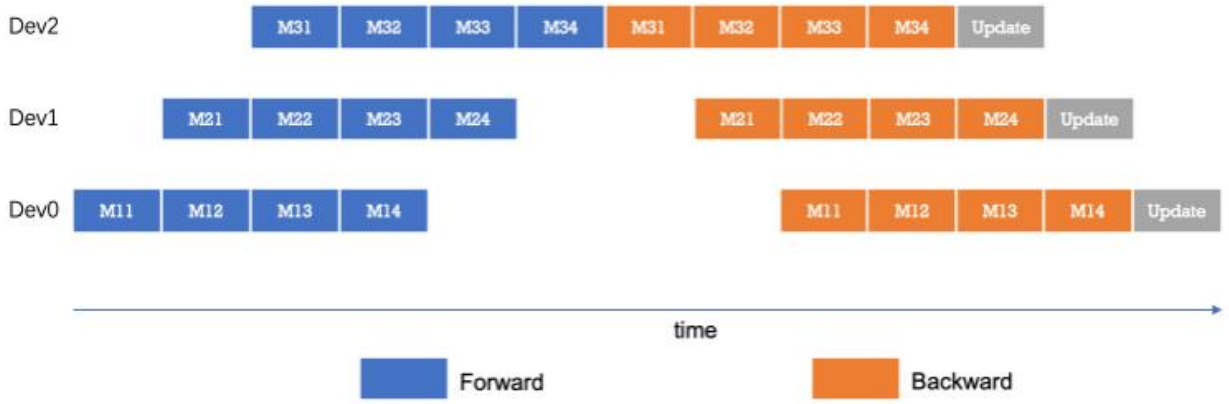
分布式流水并行

Table 1: Maximum model size of AmoebaNet supported by GPipe under different scenarios. Naive-1 refers to the sequential version without GPipe. Pipeline-k means k partitions with GPipe on k accelerators. AmoebaNet-D (L, D): AmoebaNet model with L normal cell layers and filter size D . Transformer-L: Transformer model with L layers, 2048 model and 8192 hidden dimensions. Each model parameter needs 12 bytes since we applied RMSProp during training.

NVIDIA GPUs (8GB each)	Naive-1	Pipeline-1	Pipeline-2	Pipeline-4	Pipeline-8
AmoebaNet-D (L, D)	(18, 208)	(18, 416)	(18, 544)	(36, 544)	(72, 512)
# of Model Parameters	82M	318M	542M	1.05B	1.8B
Total Model Parameter Memory	1.05GB	3.8GB	6.45GB	12.53GB	24.62GB
Peak Activation Memory	6.26GB	3.46GB	8.11GB	15.21GB	26.24GB
Cloud TPUv3 (16GB each)	**Naive-1**	**Pipeline-1**	**Pipeline-8**	**Pipeline-32**	**Pipeline-128**
Transformer-L	3	13	103	415	1663
# of Model Parameters	282.2M	785.8M	5.3B	21.0B	83.9B
Total Model Parameter Memory	11.7G	8.8G	59.5G	235.1G	937.9G
Peak Activation Memory	3.15G	6.4G	50.9G	199.9G	796.1G

分布式流水并行-1F1B

FthenB编排

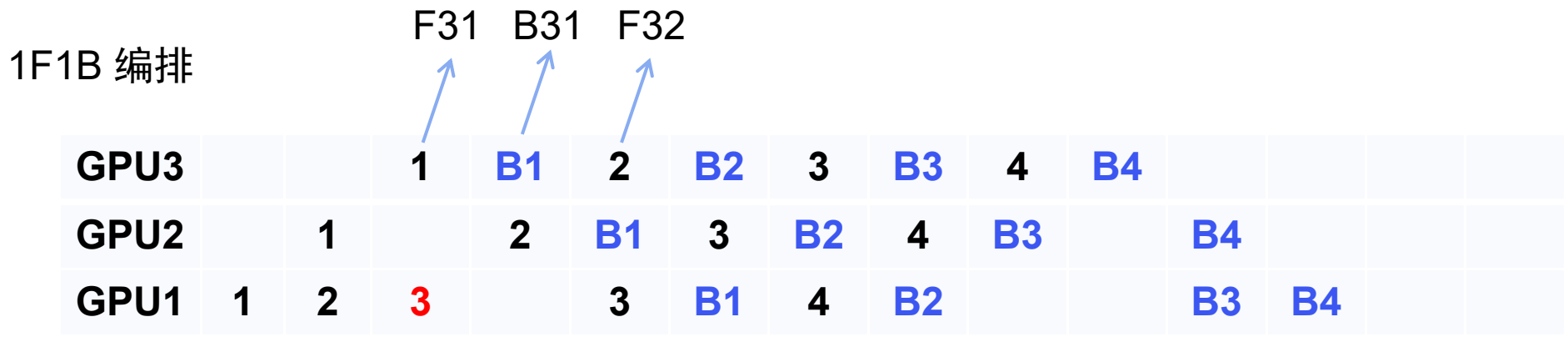


1F1B 编排

GPU3			1	B1	2	B2	3	B3	4	B4			
GPU2		1		2	B1	3	B2	4	B3		B4		
GPU1	1	2			3	B1	4	B2			B3	B4	

分布式流水并行-1F1B

1F1B 编排



1. gpu1的峰值动态内存最多只用保存3份forward的中间变量

2. gpu3的峰值动态内存最多只用保存1份forward的中间变量

分布式流水并行-1F1B编排

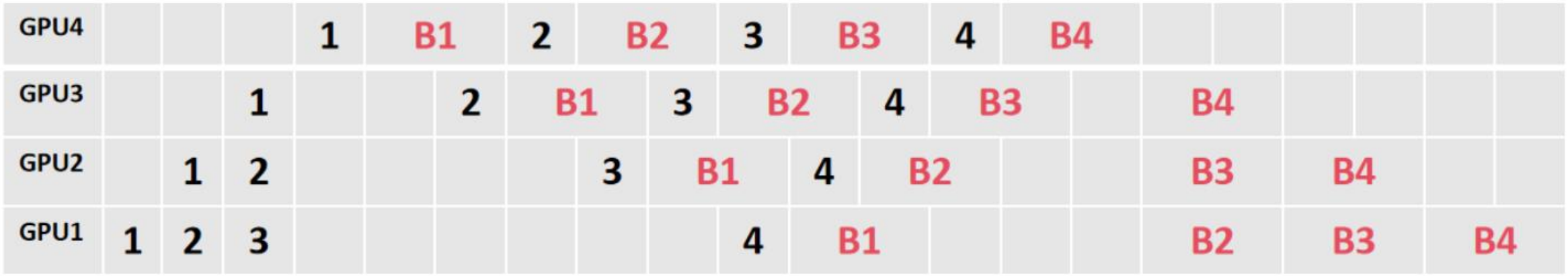
GPU4				1	B1	2	B2	3	B3	4	B4				
GPU3			1	2		B1	3	B2	4	B3		B4			
GPU2		1	2	3			B1	4	B2		B3		B4		
GPU1	1	2	3	4				B1		B2		B3		B4	

非理想时空图

GPU4				1	B1	2	B2	3	B3	4	B4	5	B5			
GPU3			1		2	B1	3	B2	4	B3	5	B4		B5		
GPU2		1	2			3	B1	4	B2	5	B3		B4		B5	
GPU1	1	2	3				4	B1	5	B2		B3		B4		B5

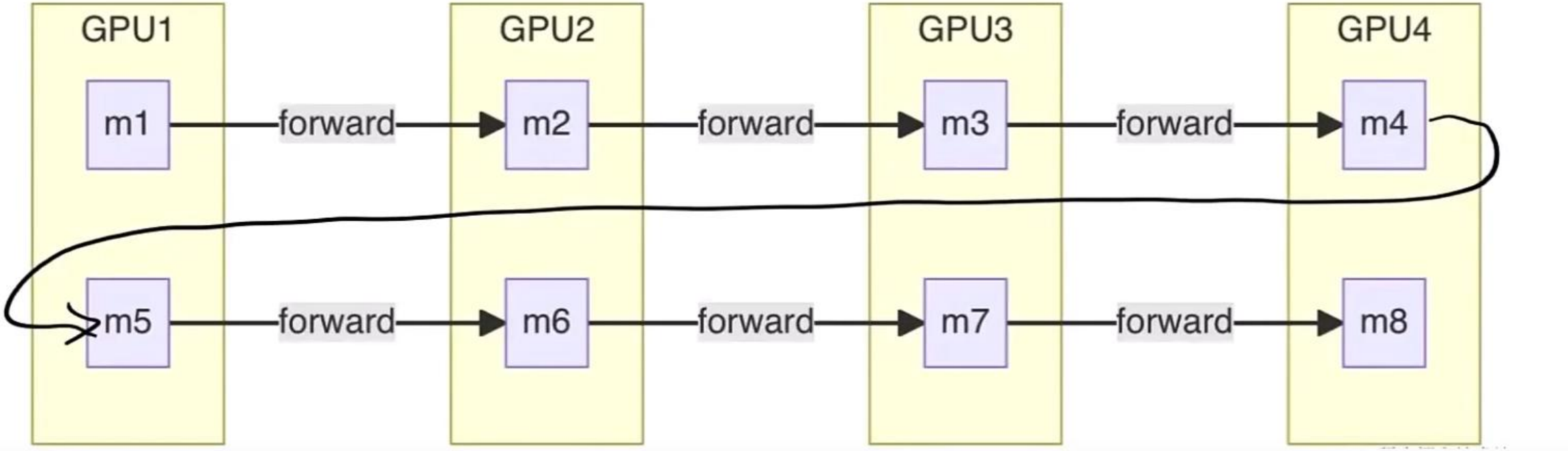
num_micro_batches>pp_degree时空图

分布式流水并行-1F1B

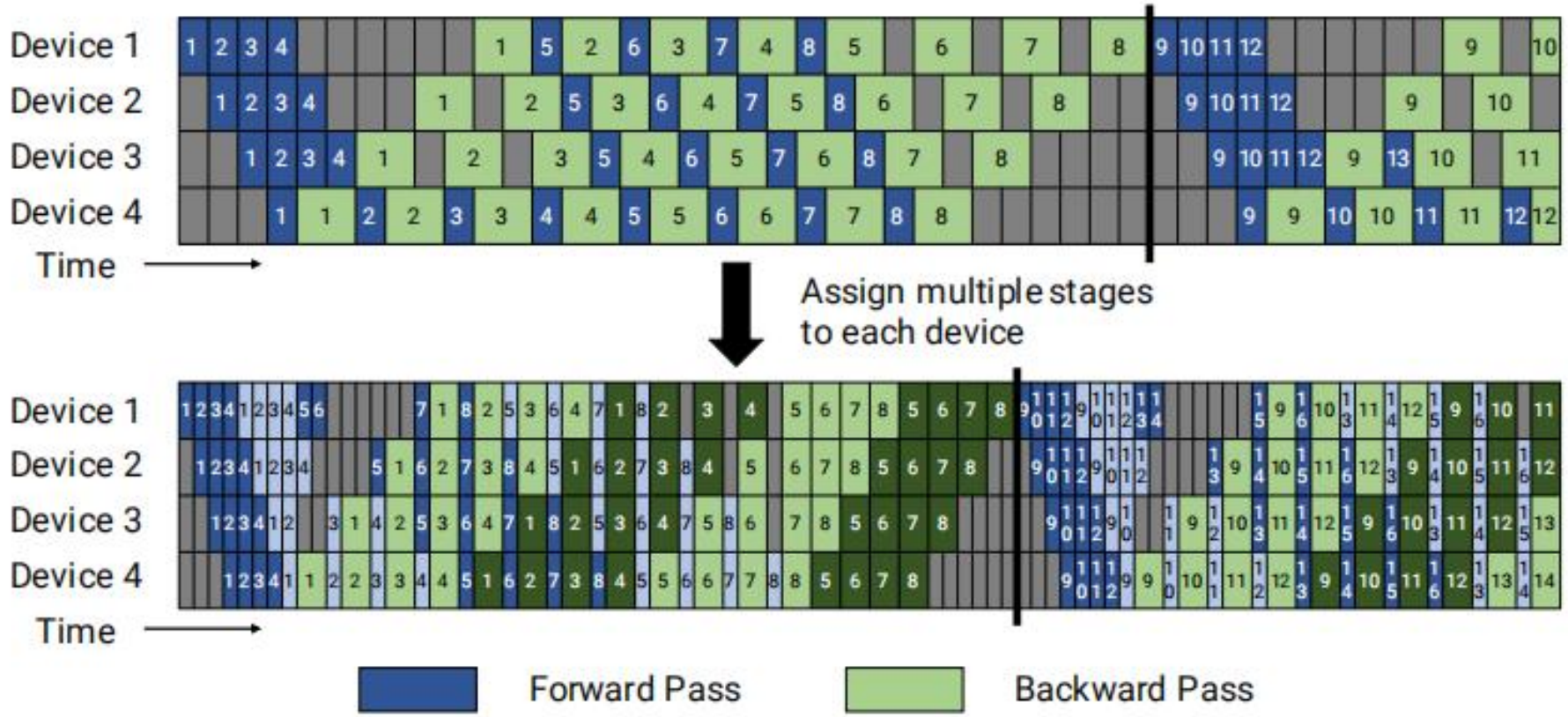


backward和forward时间不相等

分布式流水并行-VPP


GPU1
GPU2
GPU3
GPU4
m1
forward
m2
forward
m3
forward
m4
m5
forward
m6
forward
m7
forward
m8

分布式流水并行-VPP


Device 1
Device 2
Device 3
Device 4
Time
Assign multiple stages
to each device
Device 1
Device 2
Device 3
Device 4
Time
Forward Pass
Backward Pass

分布式流水并行-VPP

假设原来的每个设备的forward时间与backward时间保持一致，分别为tf和tb，则将其分为v份后，现在的时间为：tf/v和tb/v，因此由bubble产生的原因可以推导出，此时bubble时间为：

$$t_{pb}^{\text{int.}} = \frac{(p-1)\cdot(t_f+t_b)}{v},$$

在流水并行中，气泡花费时间占理想计算时间的比例是：

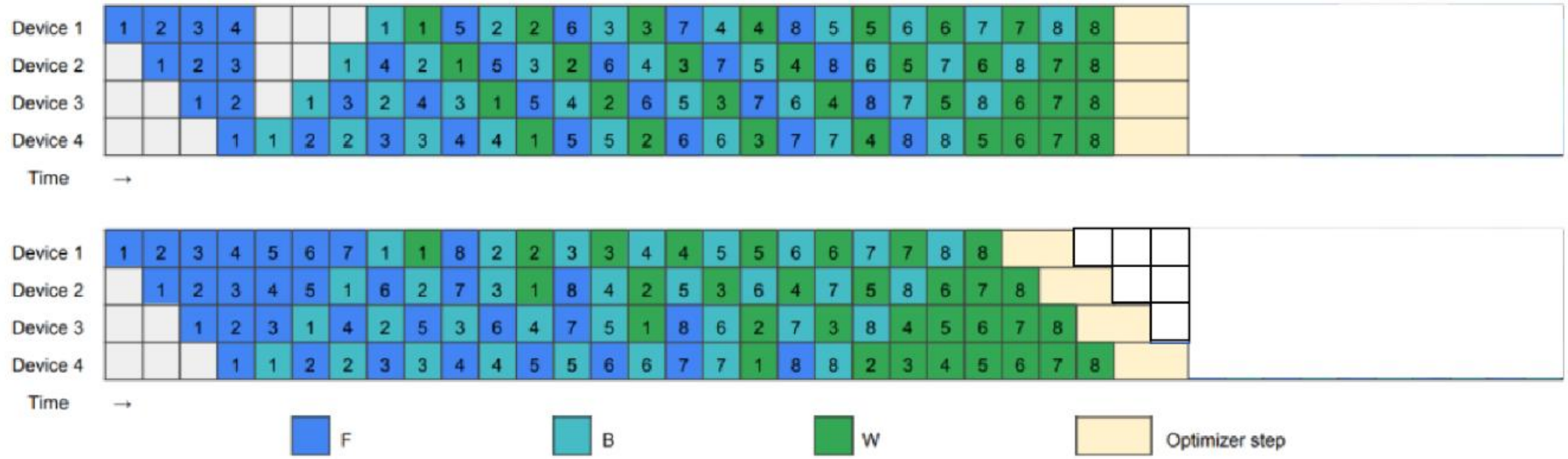
$$\text{Bubble time fraction (pipeline bubble size)} = \frac{t_{pb}^{\text{int.}}}{t_{id}} = \frac{1}{v}\cdot\frac{p-1}{m}.$$

问题

NPU 间点对点通信次数为 virtual pipeline stage 倍。

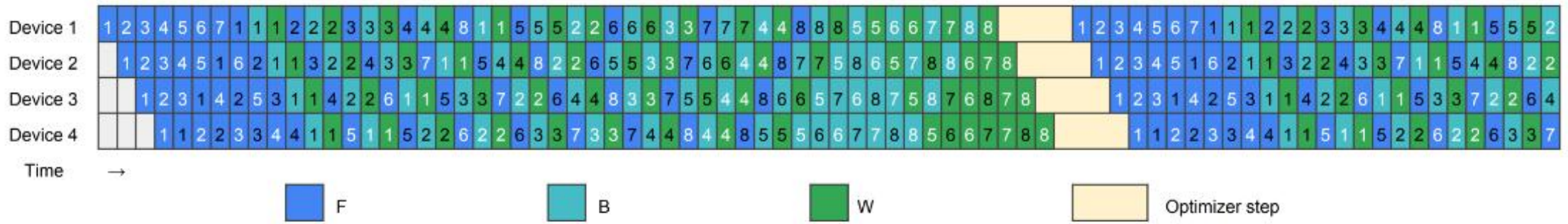
分布式流水并行-其它流水并行策略

Zero-Bubble：



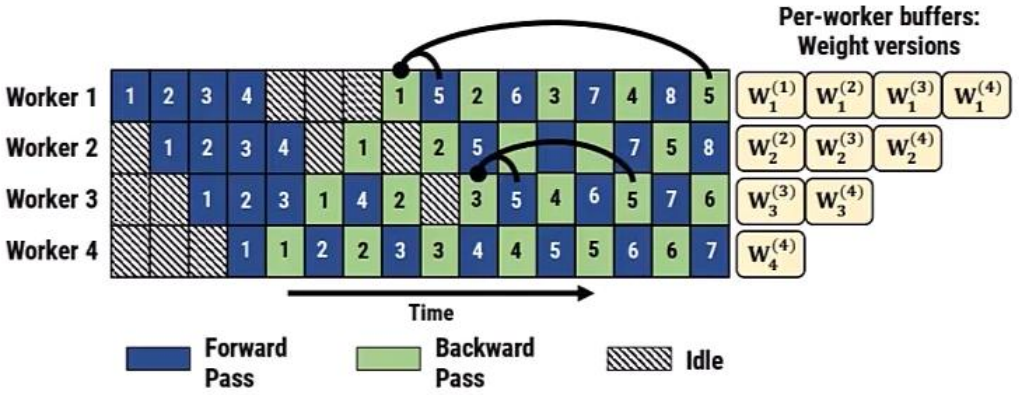
分布式流水并行-其它流水并行策略

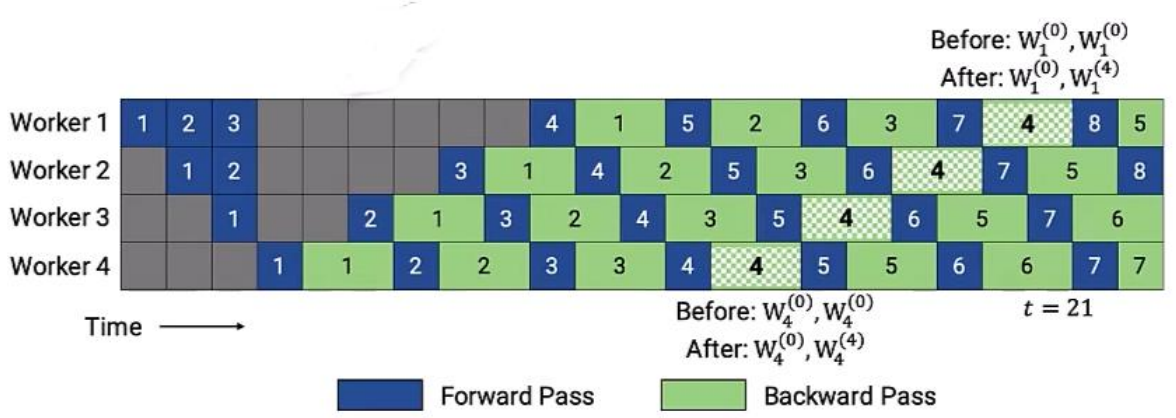
ZB-vpp：



PP 分类

- 可细分为同步流水线并行和异步流水线并行：
  1. Sync-PP 同步流水：代表 GPipe，PipeDream-flush 等；
  2. Async-PP 异步流水：代表 PipeDream，PipeDream-2BW 等





03

流水并行代码解读

分布式流水并行-程序主入口

Paddle / python / paddle / base / executor.py

Code | Blame | Executable File · 3349 lines (2937 loc) · 126 KB | Raw

```
1024                use_fetch_v2=True,
1025            )
1026
1027            new_program = program.clone()
1028            if (
1029                new_program._pipeline_opt
1030                and "standalone_opt" in new_program._pipeline_opt
1031            ):
1032                from paddle.distributed.passes.pipeline_scheduler_pass import (
1033                    apply_pass,
1034                )
1035
1036                standalone_opt = new_program._pipeline_opt["standalone_opt"]
1037                pass_name = standalone_opt["schedule_mode"]
1038                plan = apply_pass(
1039                    new_program, new_program, pass_name, standalone_opt
1040                )
```

分布式流水并行-程序主入口

Paddle / python / paddle / distributed / passes / pipeline_scheduler_pass / __init__.py ↑ Top

Code Blame 52 lines (45 loc) · 1.95 KB Raw

```python
)

__all__ = []


def apply_pass(main_program, startup_program, pass_name, pass_attr={}):
    assert pass_name in [
        "FThenB",
        "1F1B",
        "Eager1F1B",
        "VPP",
        "ZBH1",
        "ZBVPP",
    ], f"pipeline scheduler only support FThenB, 1F1B, Eager1F1B, VPP and ZBH1, but receive {pass_name}"

    if pass_name == "1F1B":
        # TODO(Ruibiao): Move FLAGS_1f1b_backward_forward_overlap and
        # FLAGS_mp_async_allreduce_in_backward to auto parallel Strategy
        # after these two optimizations are available.
        pass_attr["enable_backward_forward_overlap"] = int(
            os.environ.get("FLAGS_1f1b_backward_forward_overlap", 0)
        )

    pipeline_pass = new_pass("pipeline_scheduler_" + pass_name, pass_attr)
    pass_context = PassContext()
    pipeline_pass.apply([main_program], [startup_program], pass_context)
    plan = pass_context.get_attr("plan")
    return plan
```

```
PassBase - PipelinePassBase - PipelineFThenBPass
                            - Pipeline1F1BPass
```

分布式流水并行-FthenB对应代码

_partial_programs

Paddle / python / paddle / distributed / passes / pipeline_scheduler_pass / pipeline_fthenb.py

Code | Blame | 75 lines (61 loc) · 2.51 KB | Raw

```
35  class PipelineFThenBPass(PipelinePassBase):
49          for i in range(num_micro_batches):
50              backward_job = core.Job(BACKWARD)
51              backward_job.set_micro_batch_id(i)
52              job_list.append(backward_job)
53
54          opt_job = core.Job(OPT)
55          opt_job.set_micro_batch_id(0)
56          job_list.append(opt_job)
57          return job_list
58
59      def _partial_programs(self, program):
60          # NOTE: The flag "enable_send_recv_overlap" may increase the reserved memory of GPUs.
61          enable_send_recv_overlap = self.get_attr("enable_send_recv_overlap")
62          types = [FORWARD, BACKWARD, OPT]
63          sub_program_list = _program_for_fthenb_and_1f1b(
64              program, enable_send_recv_overlap
65          )
66          return types, sub_program_list
67
```

Paddle / python / paddle / distributed / passes / pass_utils.py

Paddlepythonpaddle分布式通道

Code 法典 | Blame 怪 | 2209 lines (1903 loc) · 80.3 KB | 2209 行 (1903 loc) · 80.3 KB

```
634
635
636 def _program_for_fthenb_and_1f1b(program, enable_send_recv_overlap=False):
637     """
638     This implementation is for fthenb and 1f1b programs and is called in partial_programs function.
639     """
640     if enable_send_recv_overlap:
641         _overlap_send_recv(program)
642     else:
643         _insert_sync_for_fthenb_1f1b(program)
644
645     fwd_prog = Program()
646     bwd_prog = Program()
647     opt_prog = Program()
648
649     # split the program based on the op_role
650     def _split_ops(block):
651         fwd_ops = []
652         bwd_ops = []
653         opt_ops = []
654         fetch_ops = []
655         for op in block.ops:
656             if _is_fetch_op(op):
657                 fetch_ops.append(op)
658             elif is_forward_op(op):
659                 fwd_ops.append(op)
660             elif is_backward_op(op):
661                 bwd_ops.append(op)
662             elif is_optimize_op(op):
663                 opt_ops.append(op)
664             else:
665                 raise ValueError(
666                     "The op role: "
667                     + str(op.attr('op_role'))
668                     + " isn't one of Forward, Backward or Optimizer."
669                 )
670         return fwd_ops, bwd_ops, opt_ops, fetch_ops
671
```

分布式流水并行-FthenB对应代码

```python
for idx, src_block in enumerate(program.blocks):
    fwd_ops, bwd_ops, opt_ops, fetch_ops = _split_ops(src_block)
    if idx == 0:
        fwd_block = fwd_prog.block(0)
        _add_ops_into_block(src_block, fwd_block, fwd_ops)

        bwd_block = bwd_prog.block(0)
        _add_ops_into_block(src_block, bwd_block, bwd_ops)

        opt_block = opt_prog.block(0)
        _add_ops_into_block(src_block, opt_block, opt_ops)
    else:
        if len(fwd_ops):
            fwd_block = fwd_prog._create_block(
                parent_idx=src_block.parent_idx
            )
            fwd_block._set_forward_block_idx(src_block.forward_block_idx)
            _add_ops_into_block(src_block, fwd_block, fwd_ops)

        if len(bwd_ops):
            bwd_block = bwd_prog._create_block(
                parent_idx=src_block.parent_idx
            )
            bwd_block._set_forward_block_idx(src_block.forward_block_idx)
            _add_ops_into_block(src_block, bwd_block, bwd_ops)

        if len(opt_ops):
            opt_block = opt_prog._create_block(
                parent_idx=src_block.parent_idx
            )
            opt_block._set_forward_block_idx(src_block.forward_block_idx)
            _add_ops_into_block(src_block, opt_block, opt_ops)
```

```python
    for fetch_op in fetch_ops:
        in_name = fetch_op.input_arg_names[0]
        dst_block = None
        for block in [fwd_block, bwd_block, opt_block]:
            if block._find_var_recursive(in_name):
                dst_block = block
                break
        if dst_block:
            _create_program(src_block, dst_block, fetch_op)
```

```python
fwd_prog._sync_with_cpp()
bwd_prog._sync_with_cpp()
opt_prog._sync_with_cpp()

fwd_prog._roll_to_global_block()
bwd_prog._roll_to_global_block()
opt_prog._roll_to_global_block()

# It MUST return in this order
return [fwd_prog, bwd_prog, opt_prog]
```

```python
def _create_program(src_block, dst_block, src_op, force_create=False):
    dst_op_desc = dst_block.desc.append_op()
    dst_op_desc.copy_from(src_op.desc)
    for input_varname in src_op.input_arg_names:
        if src_block.has_var(input_varname) or (
            force_create and src_block._find_var_recursive(input_varname)
        ):
            _create_var(src_block, dst_block, input_varname, force_create)
    for output_varname in src_op.output_arg_names:
        if src_block.has_var(output_varname) or (
            force_create and src_block._find_var_recursive(output_varname)
        ):
            _create_var(src_block, dst_block, output_varname, force_create)
```

分布式流水并行-FthenB对应代码

```python
def _create_job_list(self):
    num_micro_batches = self.get_attr("num_micro_batches")

    job_list = []

    for i in range(num_micro_batches):
        forward_job = core.Job(FORWARD)
        forward_job.set_micro_batch_id(i)
        job_list.append(forward_job)

    for i in range(num_micro_batches):
        backward_job = core.Job(BACKWARD)
        backward_job.set_micro_batch_id(i)
        job_list.append(backward_job)

    opt_job = core.Job(OPT)
    opt_job.set_micro_batch_id(0)
    job_list.append(opt_job)
    return job_list
```

分布式流水并行-FthenB对应代码

```
def _create_job_list(self):
    num_micro_batches = self.get_attr("num_micro_batches")

    job_list = []

    for i in range(num_micro_batches):
        forward_job = core.Job(FORWARD)
        forward_job.set_micro_batch_id(i)
        job_list.append(forward_job)

    for i in range(num_micro_batches):
        backward_job = core.Job(BACKWARD)
        backward_job.set_micro_batch_id(i)
        job_list.append(backward_job)

    opt_job = core.Job(OPT)
    opt_job.set_micro_batch_id(0)
    job_list.append(opt_job)
    return job_list
```

分布式流水并行-1F1B对应代码

```
def _create_job_list(self):
    num_micro_batches = self.get_attr("num_micro_batches")
    pp_stage = self.get_attr("pp_stage")
    pp_degree = self.get_attr("pp_degree")

    job_list = []
    assert (
        pp_degree <= num_micro_batches
    ), "Num of micro batches should larger than or equal to pp degree."

    micro_batch_in_warmup = pp_degree - pp_stage
    micro_batch_in_1f1b = num_micro_batches - micro_batch_in_warmup

    forward_micro_batch_id = 0
    for i in range(micro_batch_in_warmup):
        forward_job = core.Job(FORWARD)
        forward_job.set_micro_batch_id(forward_micro_batch_id)
        job_list.append(forward_job)
        forward_micro_batch_id += 1

    backward_micro_batch_id = 0
    for i in range(micro_batch_in_1f1b):
        for job_type in self.jobs_in_stable_phase:
            job = core.Job(job_type)
            micro_batch_id = (
                forward_micro_batch_id
                if job_type.startswith(FORWARD)
                else backward_micro_batch_id
            )
            job.set_micro_batch_id(micro_batch_id)
            job_list.append(job)
        forward_micro_batch_id += 1
        backward_micro_batch_id += 1

    for i in range(micro_batch_in_warmup):
        backward_job = core.Job(BACKWARD)
        backward_job.set_micro_batch_id(backward_micro_batch_id)
        job_list.append(backward_job)
        backward_micro_batch_id += 1

    opt_job = core.Job(OPT)
    opt_job.set_micro_batch_id(0)
    job_list.append(opt_job)
    return job_list
```

```
self.jobs_in_stable_phase = [BACKWARD, FORWARD]
```

GPU4				1	B1	2	B2	3	B3	4	B4			
GPU3			1	2		B1	3	B2	4		B3	B4		
GPU2		1	2	3			B1	4			B2	B3	B4	
GPU1	1	2	3	4							B1	B2	B3	B4

GPU4				1	B1	2	B2	3	B3	4	B4			
GPU3			1		2	B1	3	B2	4	B3		B4		
GPU2		1	2			3	B1	4	B2			B3	B4	
GPU1	1	2	3				4	B1				B2	B3	B4

THANK YOU

汇报人：郑天宇

时间：2024.11.17