Prob1 warm_ups

RTX 4090

context_length == 256

with warm-ups (5, 10)

	Size	d_model	d_ff	num_layers	num_heads	forward_avg_time	backward_avg_time
0	small	768	3072	12	12	0.0463001	0.0483374
1	medium	1024	4096	24	16	0.0744405	0.0767165
2	large	1280	5120	32	20	0.10142	0.108804

without warm-ups (0, 10)

	Size	d_model	d_ff	num_layers	num_heads	forward_avg_time	backward_avg_time
0	small	768	3072	12	12	0.109703	0.0604332
1	medium	1024	4096	24	16	0.122141	0.0848722
2	large	1280	5120	32	20	0.135347	0.120498

with short warm-ups (2, 10)

	Size	d_model	d_ff	num_layers	num_heads	forward_avg_time	backward_avg_time
0	small	768	3072	12	12	0.0489356	0.0488375
1	medium	1024	4096	24	16	0.0759389	0.0766813
2	large	1280	5120	32	20	0.101845	0.108322

当 model_size == "xl"时, 将发生 OOM

Prob2 Performance timing

RTX 4090

python standard library (s)

context_length == 128

	Size	d_model	d_ff	num_layers	num_heads	forward_avg_time	backward_avg_time
0	small	768	3072	12	12	0.0288064	0.0404039
1	medium	1024	4096	24	16	0.0692978	0.0794043
2	large	1280	5120	32	20	0.0809643	0.11068

context_length == 256

	Size	d_model	d_ff	num_layers	num_heads	forward_avg_time	backward_avg_time
0	small	768	3072	12	12	0.0368494	0.0432927
1	medium	1024	4096	24	16	0.0667282	0.0843609
2	large	1280	5120	32	20	0.0792045	0.118639

context_length == 512

	Size	d_model	d_ff	num_layers	num_heads	forward_avg_time	backward_avg_time
0	small	768	3072	12	12	0.0482759	0.0528984
1	medium	1024	4096	24	16	0.0857436	0.151867
2	large	1280	5120	32	20	0.155467	0.301169

> = 1024: OOM

(a).

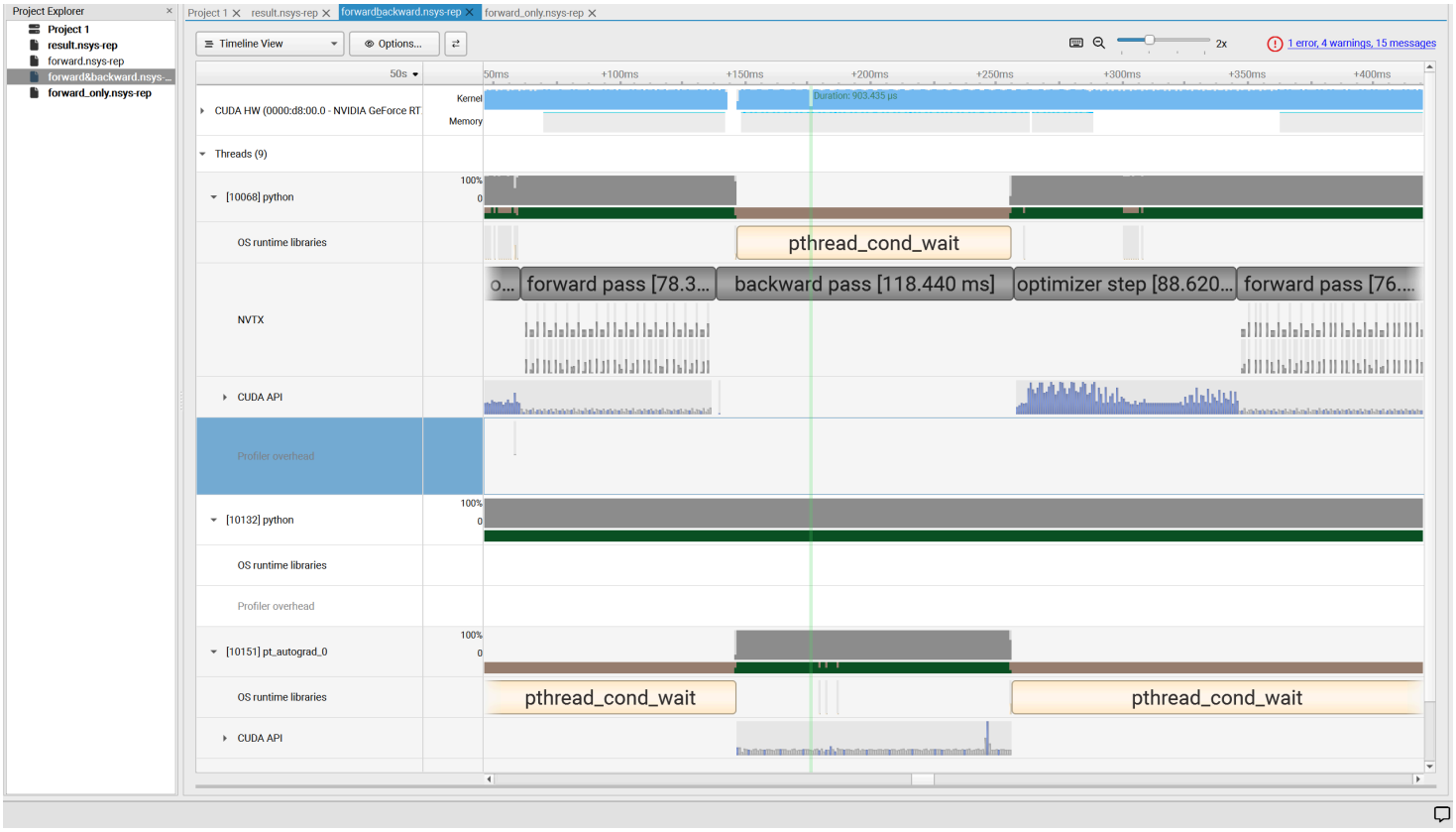
nsys profile (s)

context_length == 128

	Size	d_model	d_ff	num_layers	num_heads	forward_time_for_once	backward_time_for_once
0	small	768	3072	12	12	0.0287	0.0406
1	medium	1024	4096	24	16	0.058	0.079
2	large	1280	5120	32	20	0.075	0.115

context_length == 256

	Size	d_model	d_ff	num_layers	num_heads	forward_time_for_once	backward_time_for_once
0	small	768	3072	12	12	0.0345	0.0448
1	medium	1024	4096	24	16	0.058	0.092
2	large	1280	5120	32	20	0.076	0.117

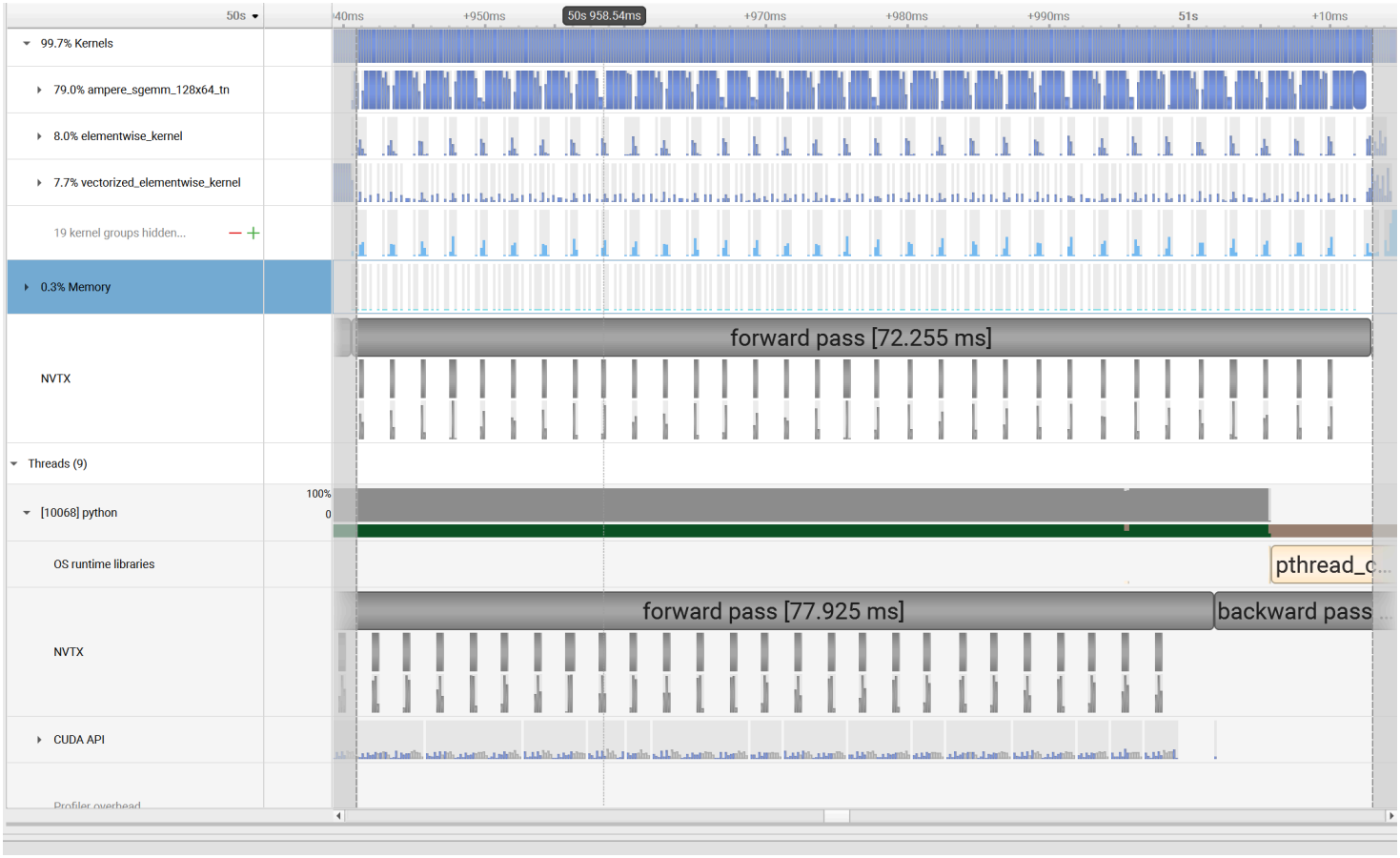


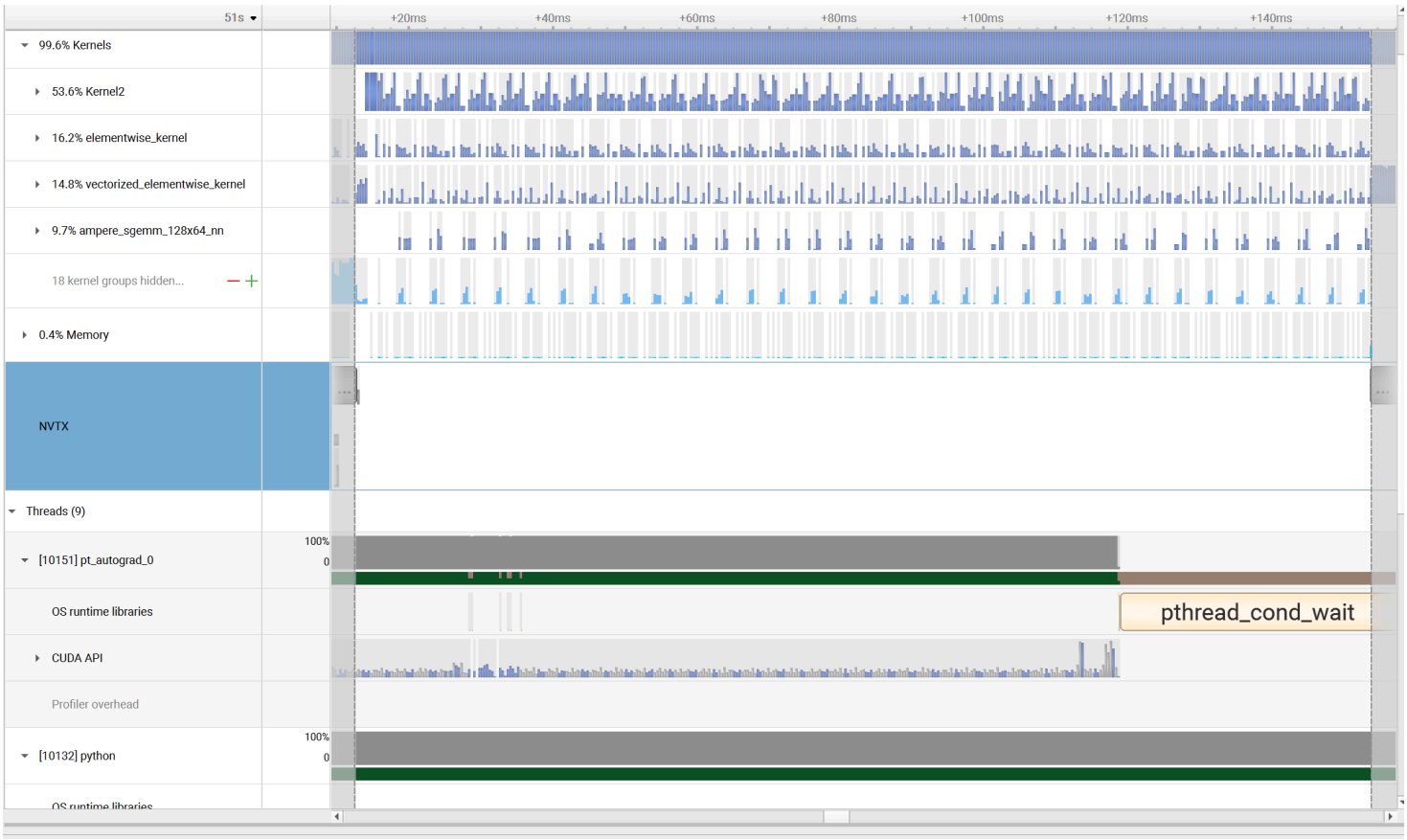
接近一致

(b).

以context_length ==256, large 为例

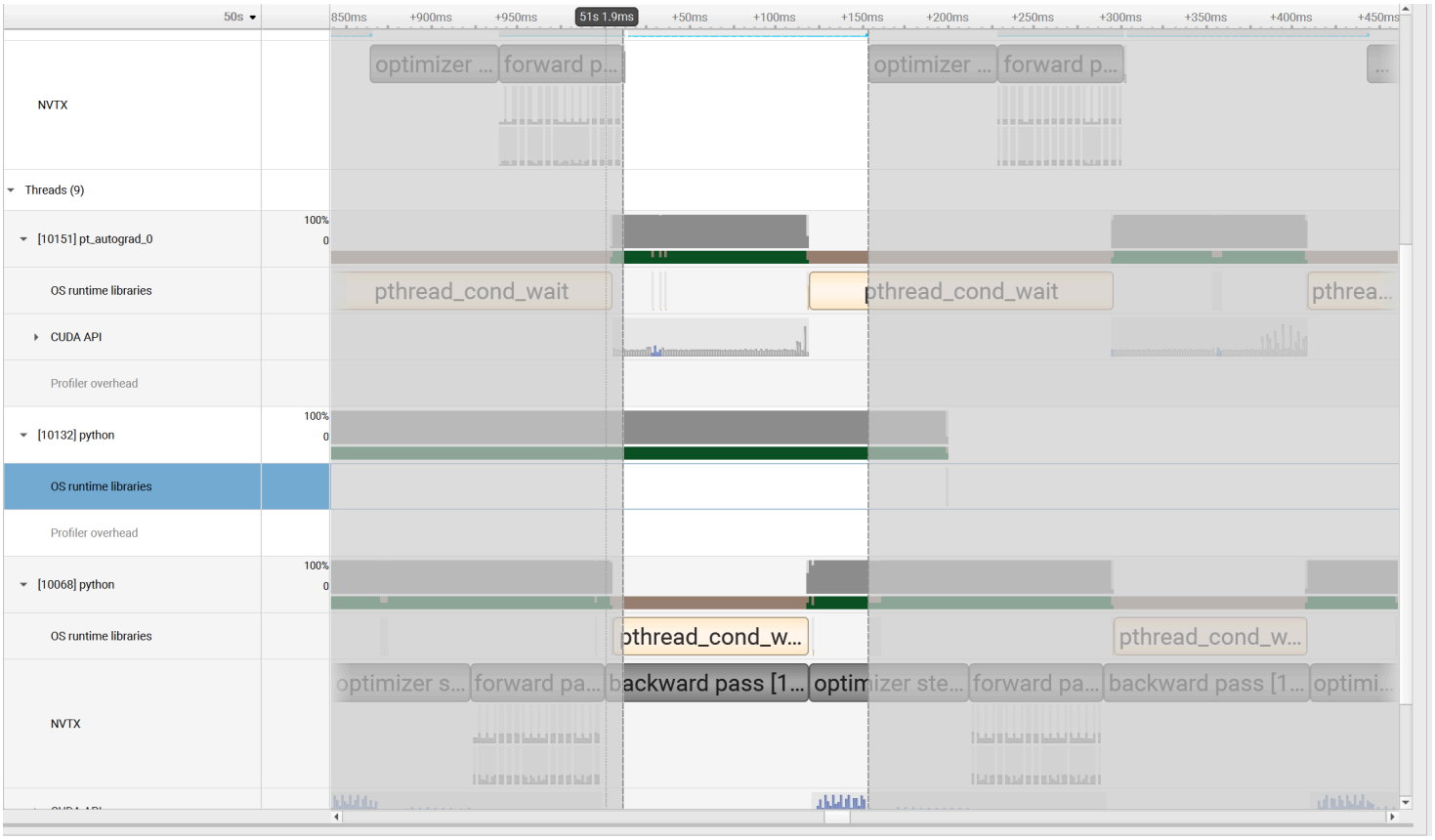
在 forward pass 中，`ampere_sgemm_128x64_tn` 大约在 78.5% 的时间内是主要运行的kernel , 调用大约 32*7次
在 backward pass 中, 占据时间最多的不是 `ampere_sgemm_128x64_nn` , 而是 Kernel2 (53.9%) 中的
`void cutlass::Kerner2<cutlass_80_simt_sgemm_128x256_8x4_nt_align1>(T1::Params)` (42.4% * 53.9%)





另外观察到一个有趣的现象: 在调用 synce(同步) 时, cpu 发送了 backward 指令后进行了较长时间的停滞, 直到 CUDA 缓慢发送完 backward 指令后, 才开始发送 optimizer 的指令.

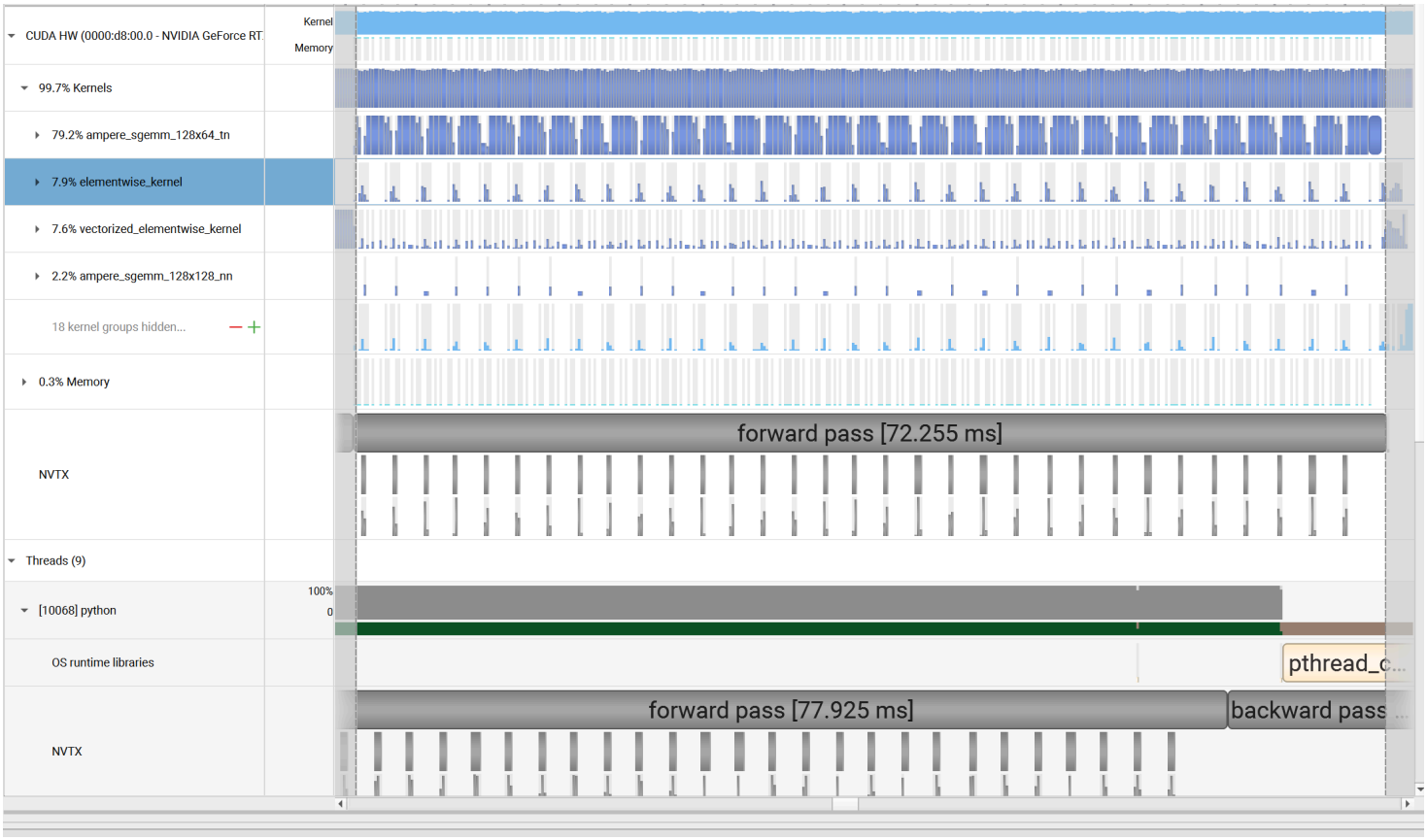
但 CUDA 发送完 backward 并不意味着已经完成任务 (?), 而是继续在 Kernel 上执行 backward 进程. 但这个时候能够观测到 CUDA 有通信 overhead, 接收 optimizer 指令了.



(c).

以context_length ==256, large 为例

- `vectorized_elementwise_lernel` : 9.0%
- `elementwise_kernel` : 7.6%
- `ampere_sgemm_128x128_tn` : 2.0%

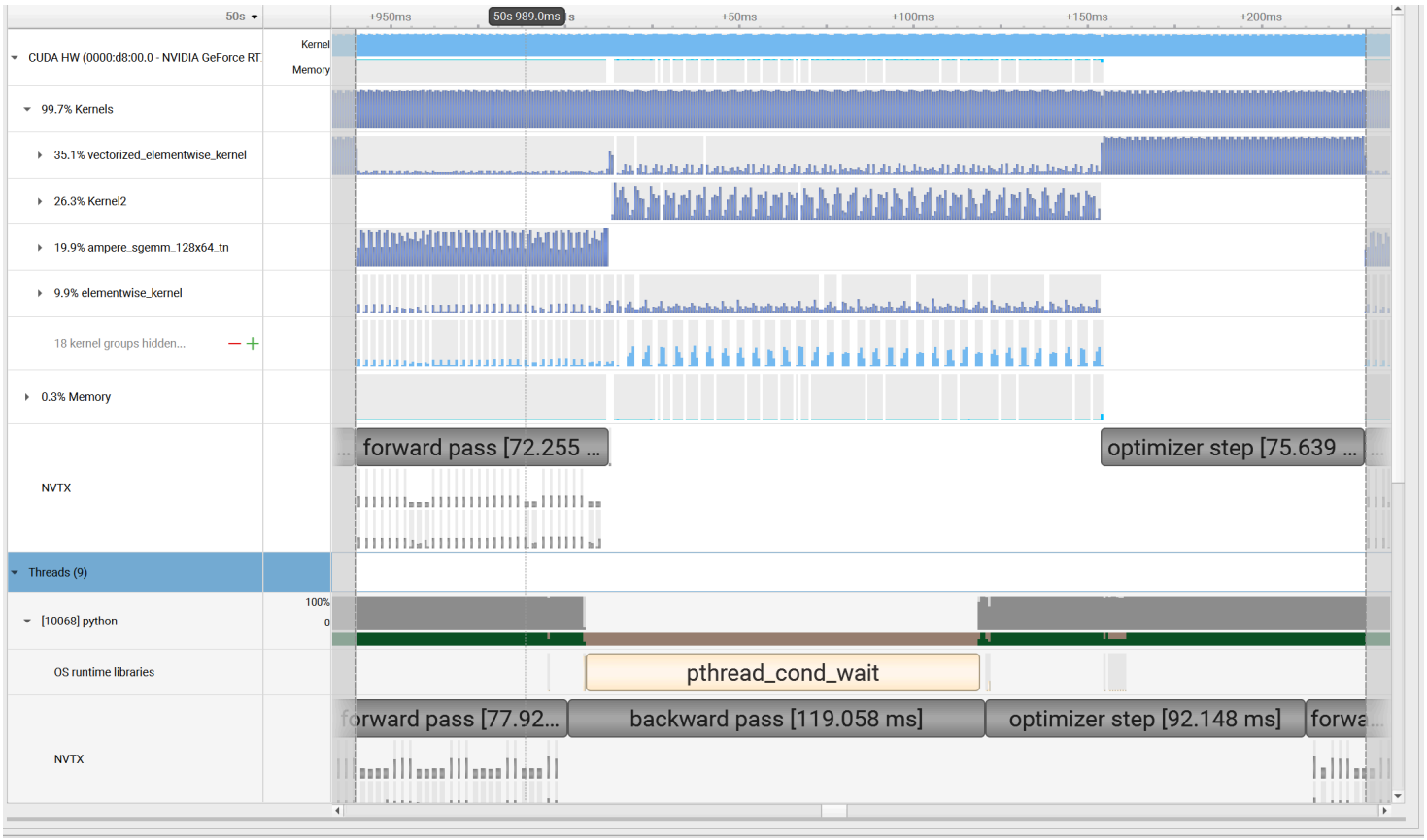


(d).

以context_length ==256, large 为例

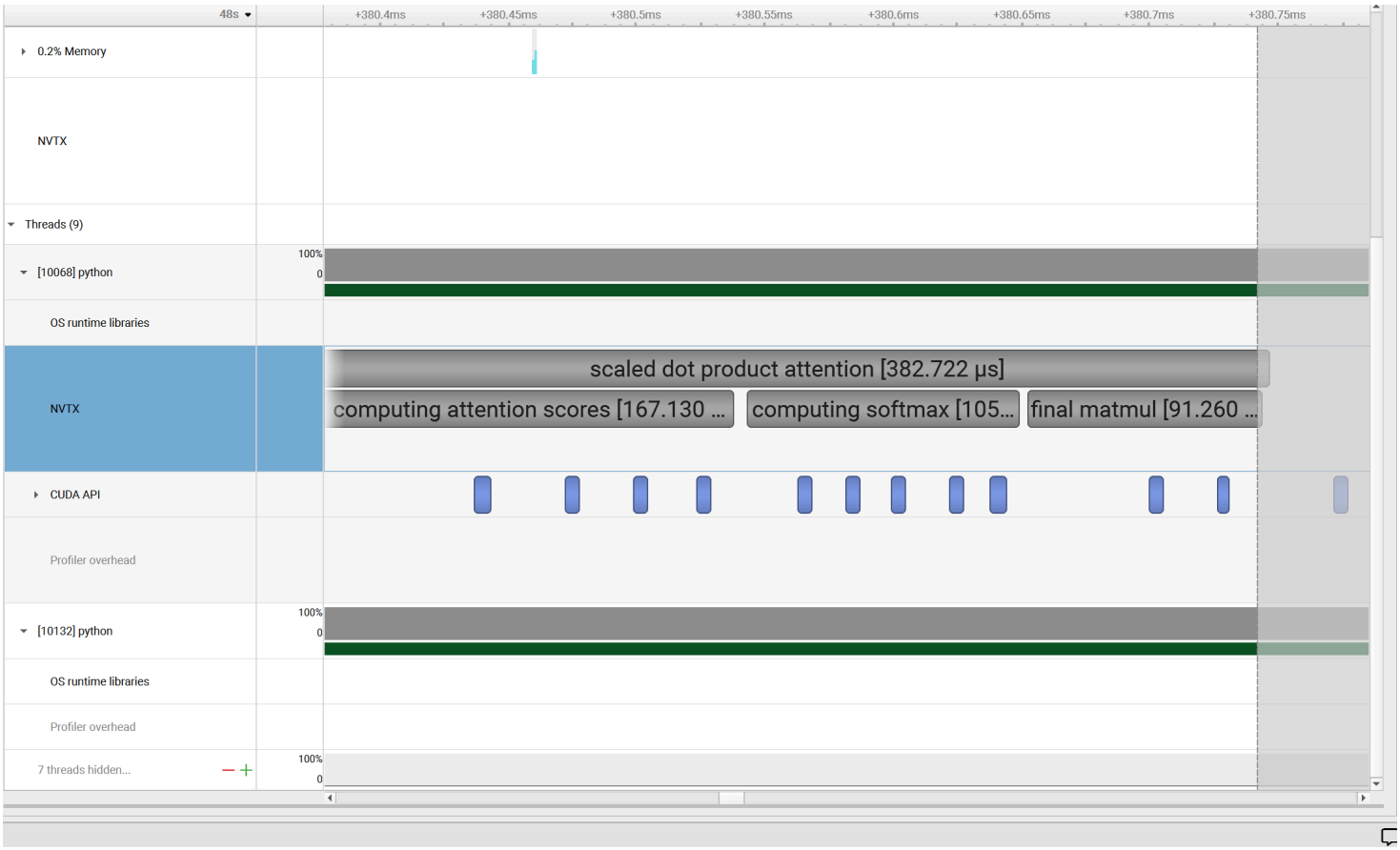
- 负责 matmul 的 `ampere_sgemm_128x64_nn` 时间由 64.0% 降低至 19.3%
- `vectorized_elementwise_lernel` 时间由 9.0% 提高至 35.4%
- `elementwise_kernel` 时间由 7.6% 提高至 10.0%
- `Kernel2` 的时间由 0 提高至 26.2%

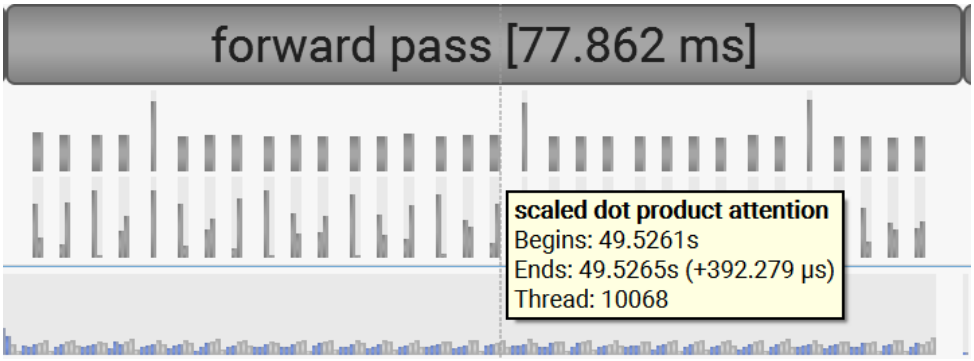
一个完整的 training loop



(e).

softmax: 105 微秒
FLOPS: b * s * (s + s + s)
matmul: 91 微秒
FLOPS: b * (2 * s * s * v)

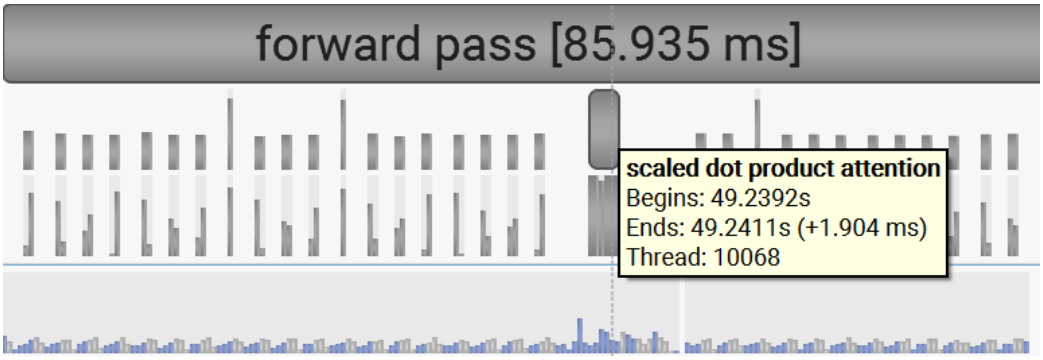




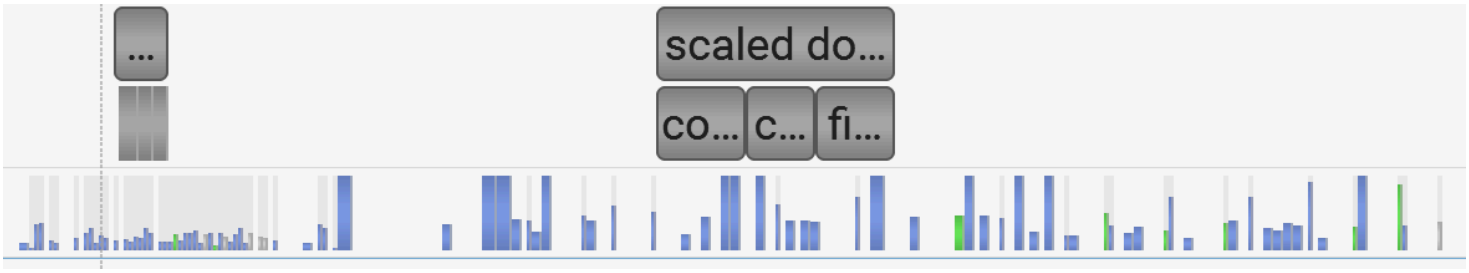
意外发现:

在少数 forward pass 中观察到耗时较长的 scale dot product Attention 过程

正常的都为 390 微秒左右, 但是下图这个超过 1900 微秒



在这个点的 Kernels 表现和其他的 Kernels 表现不一样



Prob3

```
tensor(10.0001)
tensor(9.9531, dtype=torch.float16)
tensor(10.0021)
tensor(10.0021)
```

Prob4 mixed_precision

(a).
• the model parameters within the autocast context: FP32
• the output of the first feed-forward layer (ToyModel.fc1): FP32
• the output of layer norm (ToyModel.ln): FP32
• the model's predicted logits: FP32
• the loss: FP32
• and the model's gradients: FP32

(注: 似乎题目的预期为, 除了 LayerNorm 的输出和 grad 外, 其余的精度均为 FP16, 但实测发现均为 FP32, 原因未知)

(b).

当数字变小时, 方差 Var[x] 趋近于0, 容易产生 NaN
bf16 虽然精度低, 指数位多, 更容易表示更小的数

(c).

with mixed precision:

	Size	d_model	d_ff	num_layers	num_heads	forward_avg_time	backward_avg_time
0	small	768	3072	12	12	0.045378	0.0293128
1	medium	1024	4096	24	16	0.0649713	0.0651703
2	large	1280	5120	32	20	0.0671802	0.0857588

without mixed precision:

	Size	d_model	d_ff	num_layers	num_heads	forward_avg_time	backward_avg_time
0	small	768	3072	12	12	0.0249202	0.0310806
1	medium	1024	4096	24	16	0.0598995	0.0583871
2	large	1280	5120	32	20	0.0588404	0.13528

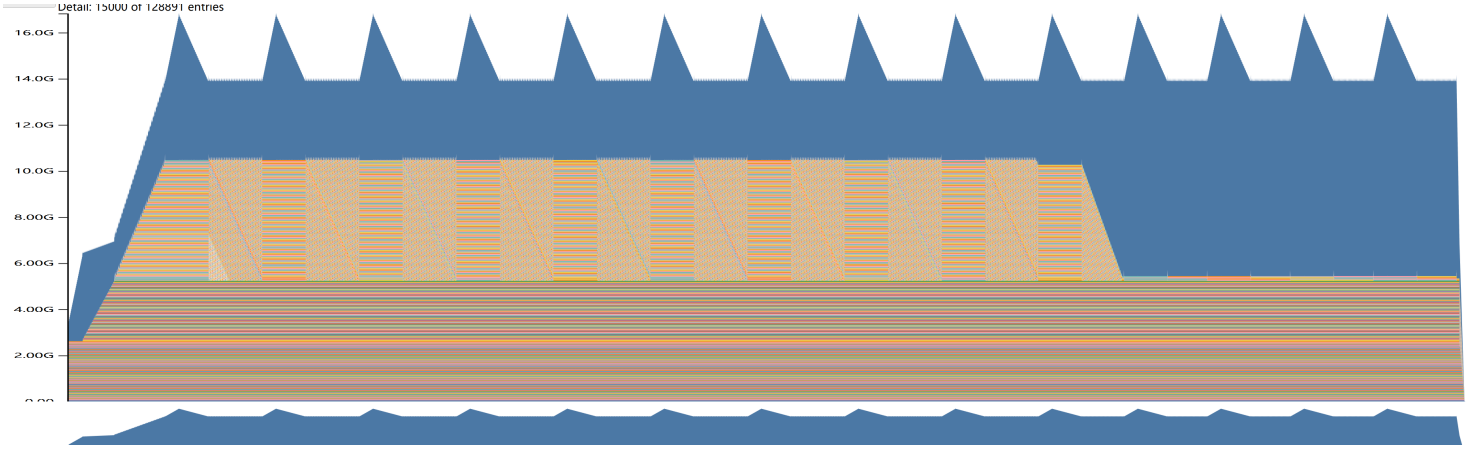
实测并未发现 mixed precision 对模型训练速度有显著影响, 可能是参数量过小 / 实验设置问题

Prob5 Mixed precision profile

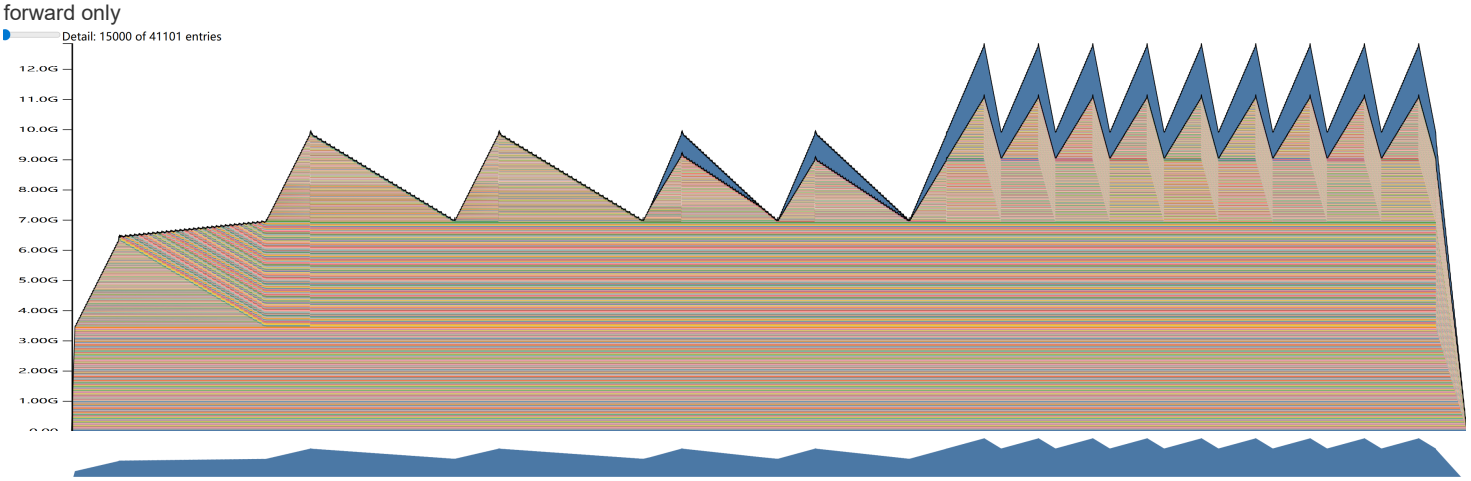
(a).

without mixed precision

forward & backward & optimizer



forward only



peaks 应该都是刚刚结束所有的 forward 和 backward, 存储了 Activation, grad 堆积
optimizer state 也会堆积很多, 不过不是周期性变化的

(b).

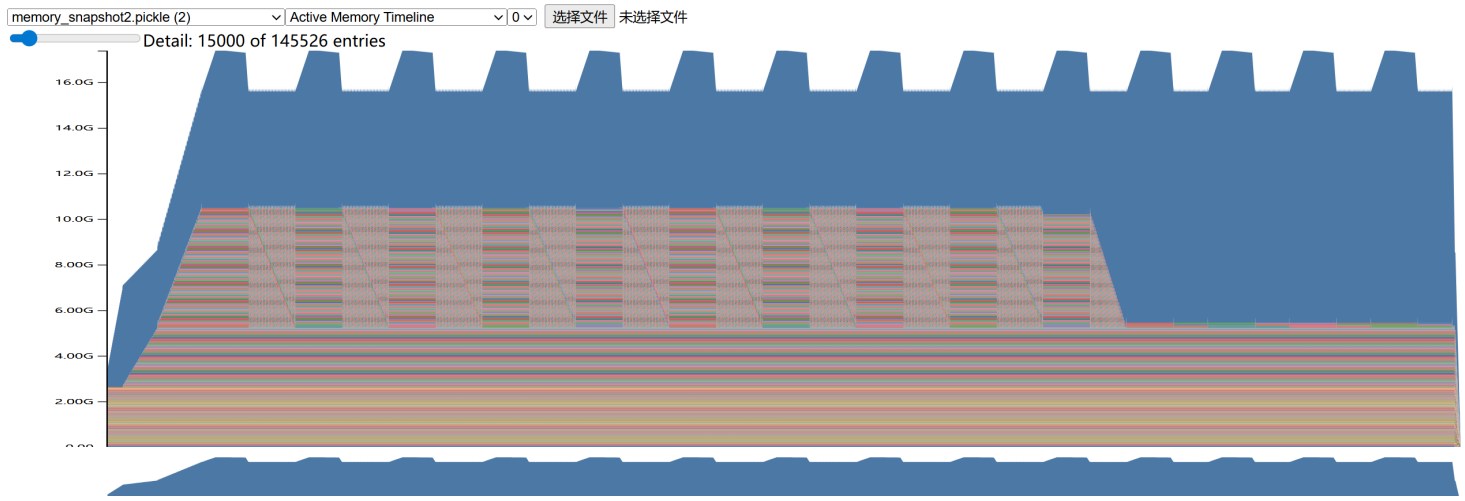
use large model size (mixed precision ver)

context length	peak
128	around 12.3 GiB
256	around 17.3 GiB

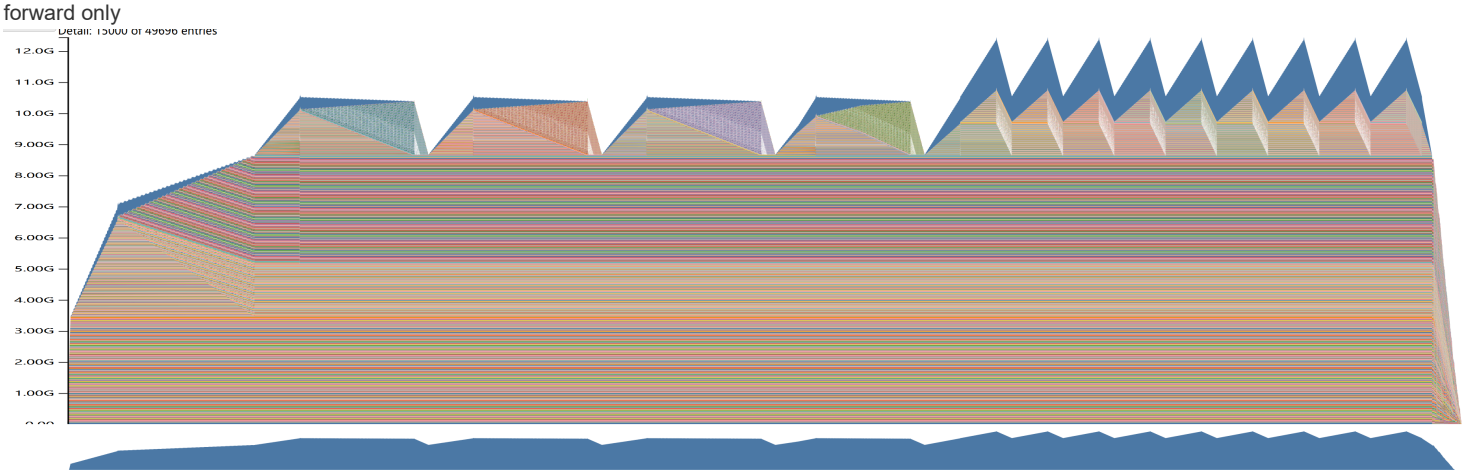
(c).

with mixed precision

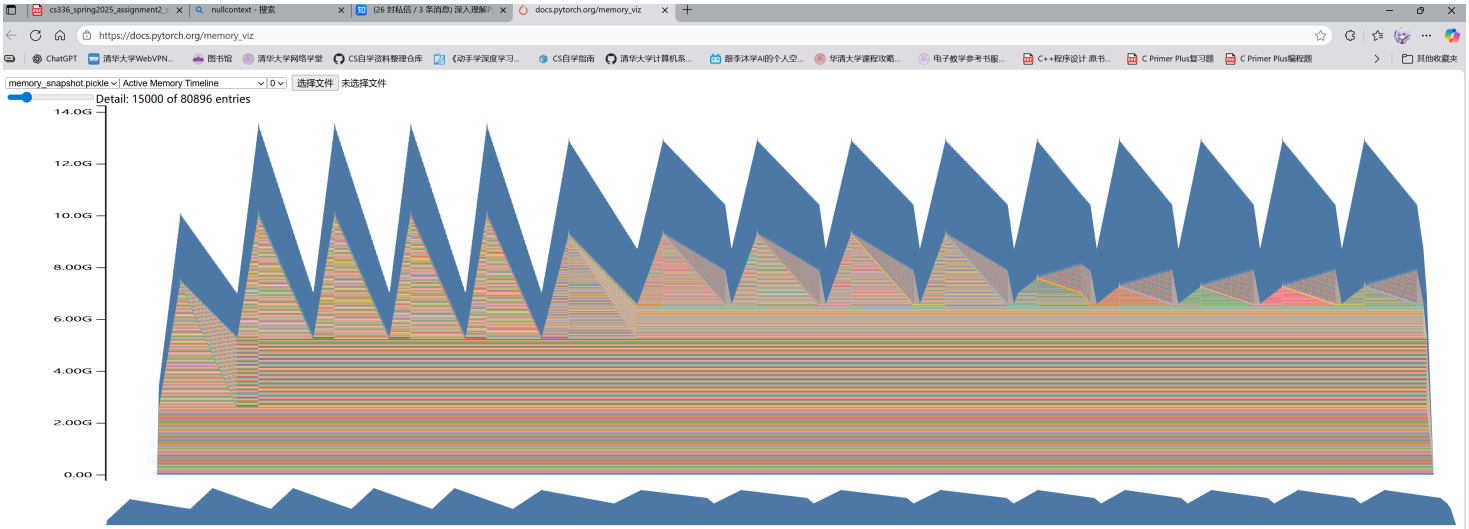
forward & backward & optimizer



forward only



forward & backward



没有很显著影响内存, 可能原因是 large size 不够大, layer 数量不够多

(d).

最大的内存 allocated

```
Addr: 0x7ff41a000000, Size: 48.8MiB (51200000 bytes), allocation,
Total memory used after allocation: 3.2GiB (345280768 bytes)
at::detail::_empty_strided_generic
at::detail::empty_strided_cuda
at::native::empty_strided_cuda
...
at::native::_to_copy
at::native::to
torch::autograd::dispatch_to
torch::autograd::THPVariable_to
...
torch/nn/modules/module.py:1343:to
benchmark_bf16_mp.py:61:main
```

在 61 行, 进行了 `transformerlm = model.BasicsTransformerLM(**hyper_params).to(device, dtype=torch.float32)` , 也就是将 model 转移到 device 上

Prob6 pytorch_attention

(a).

当 seq_len == 16384 时, 所有d_model都产生了 OOM 问题

时间统计 (s)

Forward passes:

	d_model=16	d_model=32	d_model=64	d_model=128
seq_len=256	0.028123	0.031235	0.031290	0.031774
seq_len=1024	0.037928	0.039987	0.035924	0.041369
seq_len=4096	0.732164	0.735985	0.744942	0.788171
seq_len=8192	2.868862	2.890274	2.925637	3.113246

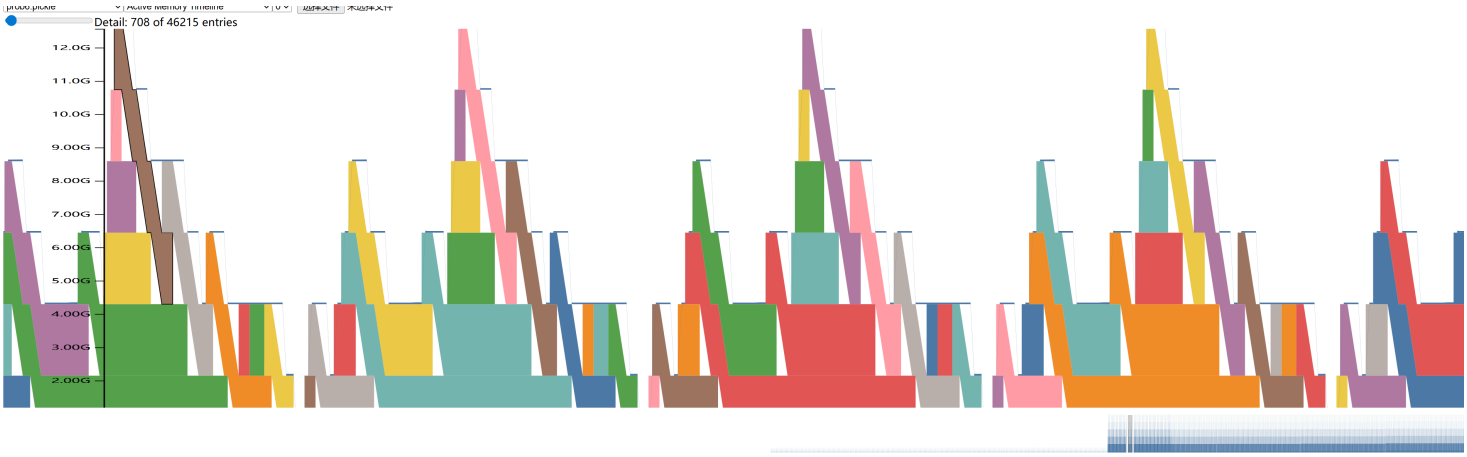
Backward passes:

	d_model=16	d_model=32	d_model=64	d_model=128
seq_len=256	0.098370	0.102585	0.108148	0.113168
seq_len=1024	0.147986	0.152316	0.146446	0.156658
seq_len=4096	1.875315	1.893239	1.906710	1.960638
seq_len=8192	7.236590	7.267536	7.348218	7.572469

Memory used(MiB):

	d_model=16	d_model=32	d_model=64	d_model=128
seq_len=256	4.019687	4.015937	4.008437	3.993437
seq_len=1024	64.078750	64.063750	64.033750	63.973750
seq_len=4096	1024.315000	1024.255000	1024.135000	1023.895000
seq_len=8192	4096.630000	4096.510000	4096.270000	4095.790000

Memory usage



Memory usage for oom:
params: 3b * s * d
grad: 3 * b * s * d
activations: b * s * s + (softmax) b * s * s
output: b * s * d
Total: sb(7d + 2s) F32 ≈ 16384(7 * 16 + 2 * 16384) * 8 ≈ 16 * 16384 * 16384, mem: 16.0 GiB (**假设均为 FP32**)

Prob7 torch compile

(a)

时间统计 (s)

Forward passes:

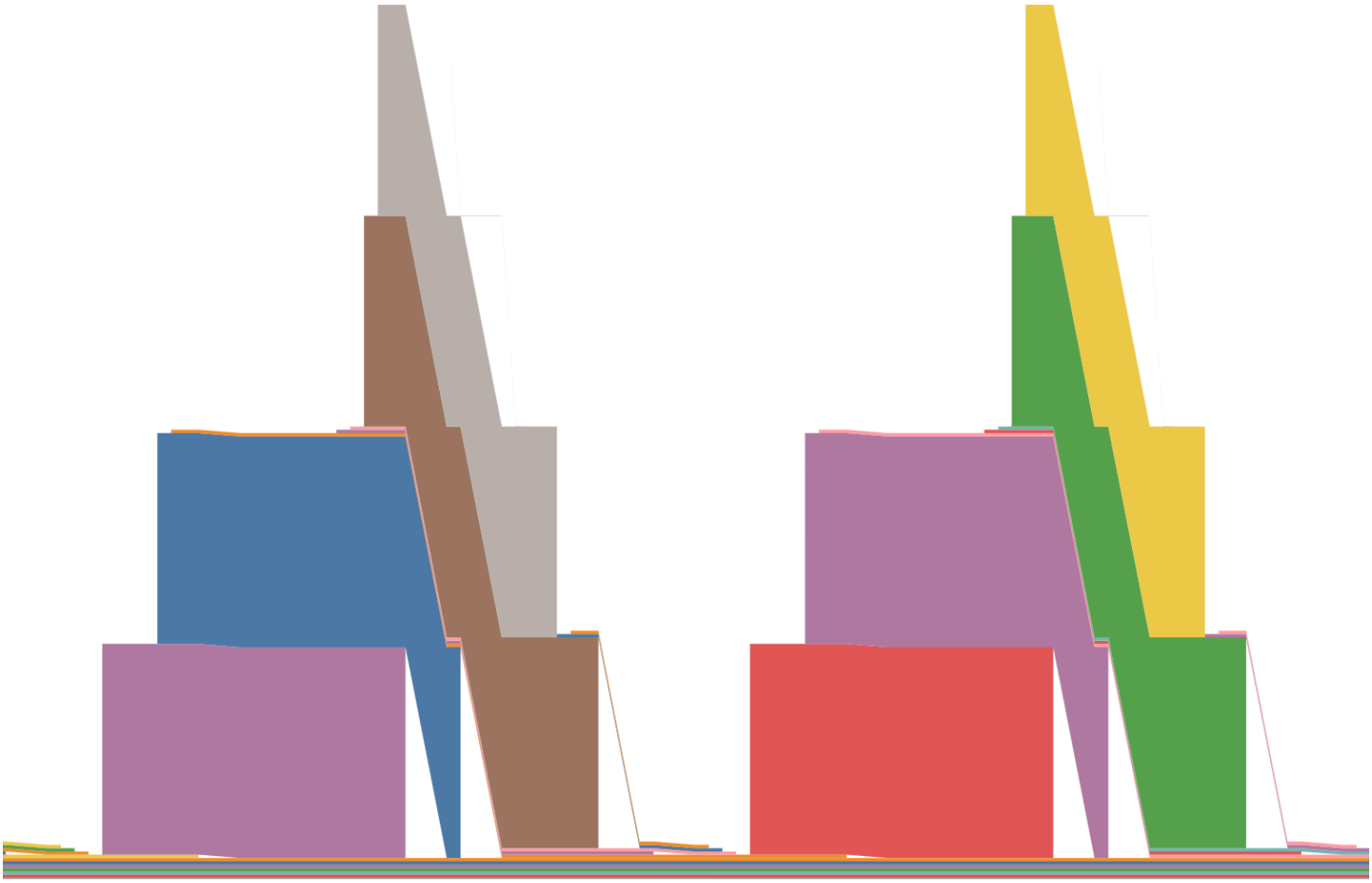
	d_model=16	d_model=32	d_model=64	d_model=128
seq_len=256	3.290159	0.033929	0.034114	0.034728
seq_len=1024	0.039195	0.040244	0.039116	0.040641
seq_len=4096	0.339077	0.345178	0.355570	0.399220
seq_len=8192	1.456346	1.477081	1.515120	1.701189

Backward passes:

	d_model=16	d_model=32	d_model=64	d_model=128
seq_len=256	0.260375	0.076426	0.078571	0.079015
seq_len=1024	0.100870	0.101201	0.105800	0.108794
seq_len=4096	0.827081	0.834893	0.852410	0.902821
seq_len=8192	3.290579	3.327250	3.400530	3.621665

Memory used (MiB):

	d_model=16	d_model=32	d_model=64	d_model=128
seq_len=256	4.715	4.71125	4.70375	4.68875
seq_len=1024	64.110	64.09500	64.06500	64.00500
seq_len=4096	1024.440	1024.38000	1024.26000	1024.02000
seq_len=8192	4096.880	4096.76000	4096.52000	4096.04000



(b).

	Size	d_model	d_ff	num_layers	num_heads	forward_avg_time	backward_avg_time	total_avg_time
with torch compile	large	1280	5120	32	20	0.0642682	0.0872608	0.24
without torch compile	large	1280	5120	32	20	0.0830164	0.0930348	0.23

Prob8 flash_forward_and_backward (ms)

使用 RTX4090 测试, model 为 assignment2 中提供的 BasicTransformerLM

Torch Naive Attention

Attention Torch Forward:

d_model \ seq_len	128	256	512	1024	2048	4096	8192	16384	32768
16	0.0449237	0.274046	0.283092	0.320324	0.325731	0.596342	3.28397	12.8191	51.0505
32	0.0683035	0.265134	0.311433	0.333644	0.302267	0.54288	3.28807	12.8358	51.0648
64	0.041639	0.29487	0.27355	0.36957	0.144906	0.559407	3.28521	12.8468	51.0945
128	0.0630773	0.342754	0.271543	0.375219	0.315123	0.570958	3.32354	12.8833	51.3761

Attention Torch Backward:

d_model \ seq_len	128	256	512	1024	2048	4096	8192	16384	32768
16	2.40036	2.32625	2.54427	2.78588	2.20498	2.00178	8.05614	31.7512	126.605
32	2.03277	2.27827	2.21	2.58057	2.58112	2.87687	8.03396	31.7891	126.75
64	2.12625	2.95625	2.74604	2.54082	2.37155	2.9891	8.04471	31.829	126.852
128	2.13714	2.42881	2.91332	2.81541	2.48555	2.36356	8.14776	31.9498	127.719

Flash Attention TILE_SIZE = 16

FlashAttention Triton Forward:

d_model \ seq_len	128	256	512	1024	2048	4096	8192	16384	32768
16	0.00667586	0.0816758	0.0440773	0.0433202	0.0594017	0.110447	0.26825	0.960052	3.64314
32	0.00766382	0.0279245	0.0580939	0.0717642	0.0793042	0.122902	0.360328	1.36018	5.28237
64	0.00791666	0.0468552	0.0389979	0.0529518	0.0699165	0.17531	0.416052	1.65404	6.2246
128	0.0087801	0.047328	0.0322787	0.0555022	0.0965117	0.197098	0.903344	2.90823	11.4836

FlashAttention Triton Backward:

d_model \ seq_len	128	256	512	1024	2048	4096	8192	16384	32768
16	1.34397	1.70778	1.66684	1.48521	1.35072	1.79065	3.95692	15.4761	61.3939
32	1.06962	1.65409	1.10282	1.48602	1.98737	1.95682	4.03829	15.8747	63.0077
64	1.46118	1.77582	1.93472	1.17111	1.48734	1.36301	4.08557	16.1693	63.8505
128	1.148	1.13243	1.24289	1.72285	1.32885	1.31659	4.63503	17.495	69.6125

Flash Attention TILE_SIZE = min(max(triton.next_power_of_2(N_QUERIES // 16), 16), 32)

FlashAttention Triton Forward:

d_model \ seq_len	128	256	512	1024	2048	4096	8192	16384	32768
16	0.00653908	0.0465722	0.0343692	0.0503745	0.0687898	0.103134	0.133727	0.388258	1.44299
32	0.00712324	0.0400971	0.083369	0.0696306	0.0660635	0.087271	0.160495	0.547802	2.10066
64	0.00774129	0.051381	0.0547503	0.0770818	0.0654438	0.113063	0.225937	0.893116	3.13556
128	0.00943073	0.0345451	0.0231297	0.0678934	0.0782563	0.148981	0.541289	2.1526	8.5506

FlashAttention Triton Backward:

d_model \ seq_len	128	256	512	1024	2048	4096	8192	16384	32768
16	1.62592	1.58421	1.67927	2.24697	1.33685	1.5109	3.82474	14.8944	59.1442
32	1.74258	1.33925	1.43691	1.48768	1.55436	1.58921	3.83377	15.0538	59.8057
64	1.24792	1.27093	1.773	1.43276	2.02625	1.5885	3.89281	15.4143	60.8307
128	1.34698	1.03743	0.928447	0.967129	0.953448	0.926795	4.27269	16.7163	67.0423

- 以上数据均为使用 `triton.testing.do_bench` 后, 进行 5 轮 warm_ups 后, 计算 10 轮 iters 的平均用时
- 由于无法复用梯度, 为了兼容 `triton.testing.do_bench` 接口, 因此 `bwd` 计算时包含了 `fwd` ,
- 所用 `FlashAttention` 未实现 `bwd` Triton Kernel 的编写
- 在数据规模不够多时, forward pass 的速度远快于 backward pass 的速度, 原因未知(考虑 sum 操作 / backward 本身特性)

源代码:

```
class Attention(nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, Q, K, V, is_causal=False):
        if not is_causal:
            return model.scaled_dot_product_attention(Q, K, V)
        mask = torch.tril(torch.ones(Q.shape[-2], K.shape[-2], device=Q.device)).bool()
        return model.scaled_dot_product_attention(Q, K, V, mask)

def fwd_torch(Q, K, V):
    return attn(Q, K, V, True)

def bwd_torch(Q, K, V):
    O = attn(Q, K, V, True)
    O.sum().backward()
    Q.grad = K.grad = V.grad = None
    return O
```

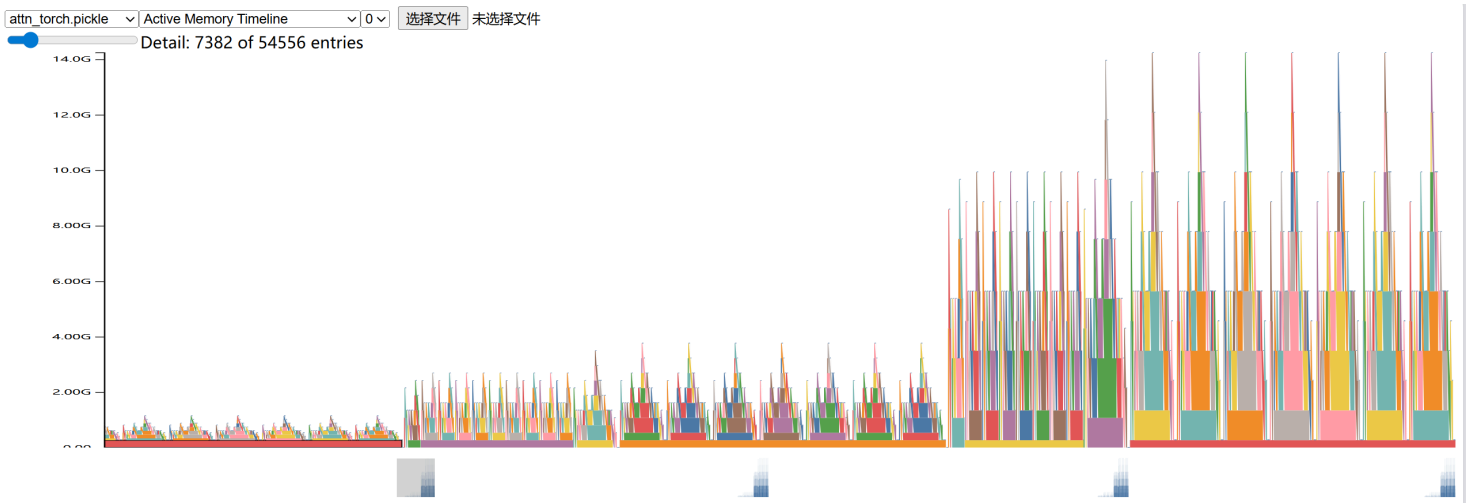
关于 Triton Kernel 编写

- 当TILE_SIZE=64时, 出现了 "triton.runtime.errors.OutOfResources: out of resource: shared memory, Required: 107008, Hardware limit: 101376. Reducing block sizes or `num_stages` may help." 的报错,
  说明当前GPU的shared memory容量超出上限
- 此外 TILE_SIZE 的大小最好是 2 的幂次, 方便用 TILE_SIZE 整除 seq_len, 从而让 GPU 尽可能不出现空档 (貌似如果不是2的幂次会抛出"ValueError: Shape element 0 must be a power of 2"的错误)
- tl.dot 好像有 "Input shapes should have M >= 16, N >= 16 and K >= 16" 的要求

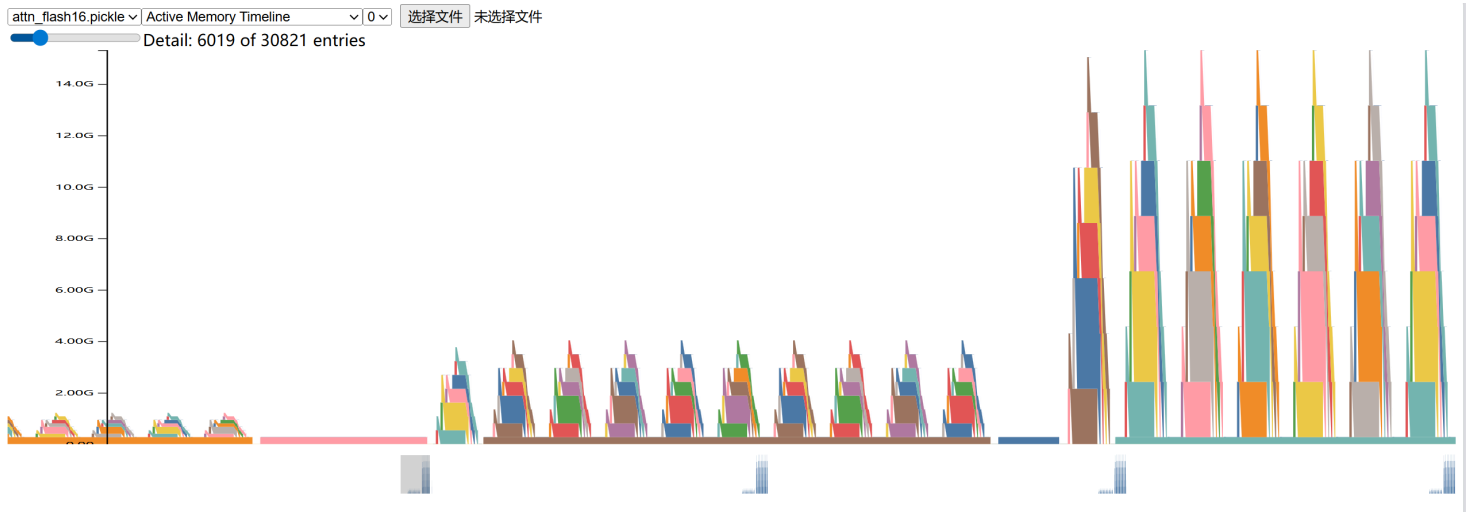
```
Traceback (most recent call last):
  File "/root/assignment2-systems/.venv/lib/python3.12/site-packages/triton/language/core.py", line 35, in wrapper
    return fn(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^
  File "/root/assignment2-systems/.venv/lib/python3.12/site-packages/triton/language/core.py", line 1548, in dot
    return semantic.dot(input, other, acc, input_precision, max_num_imprecise_acc, out_dtype, _builder)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/root/assignment2-systems/.venv/lib/python3.12/site-packages/triton/language/semantic.py", line 1488, in dot
    assert lhs.shape[-2].value >= min_dot_size[0] and lhs.shape[-1].value >= min_dot_size[2] \
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError: Input shapes should have M >= 16, N >= 16 and K >= 16
```

关于显存占用

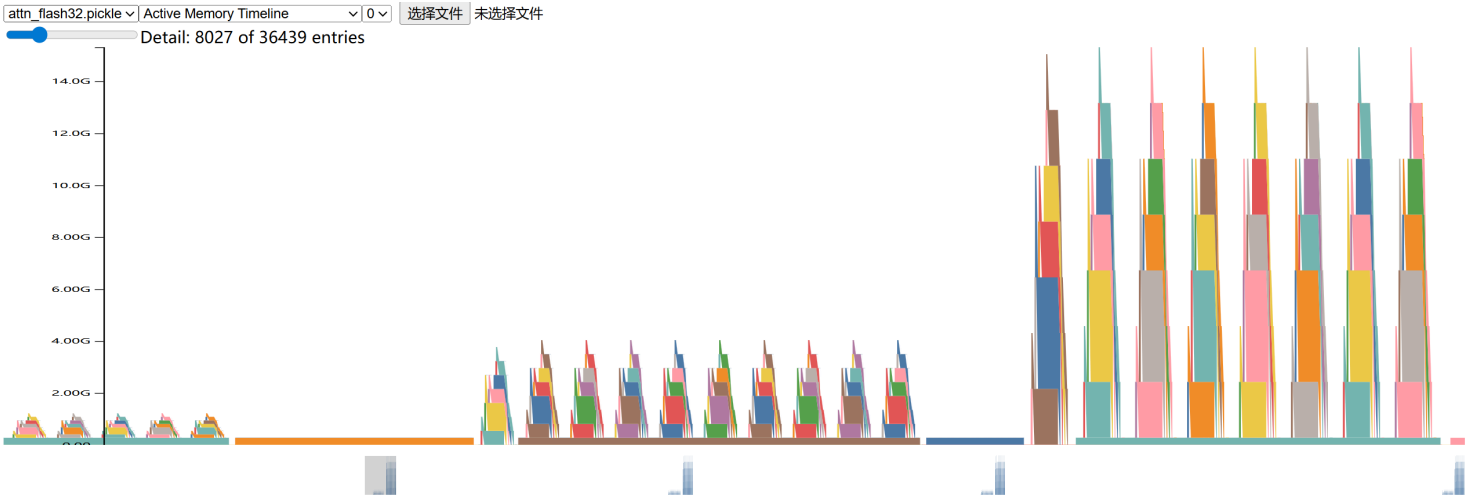
Atten_torch



Flash_torch (TILE_SIZE = 16)



Flash_torch (TILE_SIZE = min(max(triton.next_power_of_2(N_QUERIES // 16), 16), 32))



易错点: **Flash Attention 并不会影响内存在 GPU 上的分配** (上图中都是大约 15 GiB), 而是影响在 global memory(慢) 和 shared memory(快) 上的内存读写!!

Prob9 distributed all_reduce

world_size == 2, 2 * RTX4090

	data_size=1024.0MiB	data_size=256.0MiB	data_size=64.0MiB	data_size=16.0MiB	data_size=4.0MiB	data_size=1.0MiB
word size=1	2.71634	2.73385	2.71788	2.72141	2.7766	2.73034
word size=2	3.12189	3.2328	3.08259	3.09528	2.95279	3.12101

distributed 训练似乎会减慢一点速度

Prob10-13 naive_ddp_benchmarking, minimal_ddp_flat_benchmarking, ddp_overlap_individual_parameters_benchmarking, ddp_bucketed_benchmarking

model_size == medium, world_size == 2, 2 * RTX4090, 显存 24 GiB

当前 GPU 配置无法采用 model_size == "large"

without flatten

rank	train time	reduce time	reduct portion
rank1	0.503	0.0662	0.13
rank0	0.506	0.27	0.53

with flatten

rank	train time	reduce time	reduct portion
rank1	0.493	0.0314	0.064
rank0	0.5	0.0372	0.074

把 grad 打包后 reduce, 减少通信 overhead

ddp_overlap_no_bucket

rank	train time
rank1	0.407
rank0	0.408

ddp_overlap_with_different_bucket_sizes (MiB)

rank	train time	bucket_size
rank1	0.419	1
rank0	0.419	1

rank	train time	bucket_size
rank1	0.405	10
rank0	0.405	10

rank	train time	bucket_size
rank1	0.406	100
rank0	0.407	100

rank	train time	bucket_size
rank1	0.457	1000
rank0	0.458	1000

当param的参数量不够多时, 增大 bucket_size (如最下面的 1000 MiB) 只会降低效率, 需要 bucket_size 在一个合适的水平 (如 5-50 次 grad all_reduce 开销), 才能提高效率

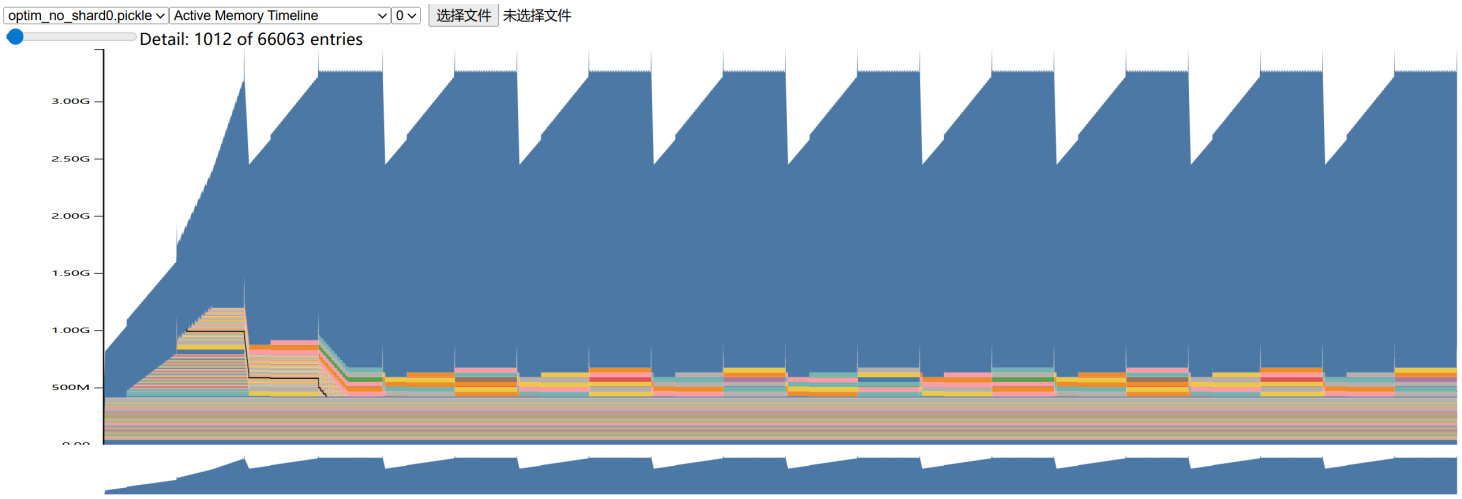
Prob15 optim benchmarking

model_size == medium, world_size == 2, 2 * RTX4090, 显存 24 GiB

速度分析

without shard

rank	train time
rank1	0.313
rank0	0.313



with shard

(with shard 下, pytorch的memory_profile无法追踪)

rank	train time
rank1	0.413
rank0	0.414

可以看到, shard optimizer 会导致一定程度的速度减慢 (原因是每次 optimize 后需要进行 broadcast 的开销)

内存分析:

以下是在 `bm_optim.py` 中通过 `torch.cuda.memory_allocated(rank)` 分析得到的内存使用情况

```
-----------------Optimizer_no_shard start-----------------
Rank0: Init Allocated: 0.00 MiB
Rank1: Init Allocated: 0.00 MiB
Rank0: After model Allocated: 772.46 MiB
Rank1: After model Allocated: 772.46 MiB
Rank0: After output 0 Allocated: 996.61 MiB; Output memory: 224.16 MiB
Rank1: After output 0 Allocated: 996.61 MiB; Output memory: 224.16 MiB
Rank1: After backward 0 Allocated: 1569.69 MiB; Gradient memory: 573.07 MiB
Rank0: After backward 0 Allocated: 1569.69 MiB; Gradient memory: 573.07 MiB
Rank0: After optimizer 0 Allocated: 3132.85 MiB; Optimizer state memory: 2136.24 MiB
Rank1: After optimizer 0 Allocated: 3132.85 MiB; Optimizer state memory: 2136.24 MiB

-----------------Optimizer_shard start-----------------
Rank0: Init Allocated: 0.00 MiB
Rank1: Init Allocated: 0.00 MiB
Rank0: After model Allocated: 772.46 MiB
Rank1: After model Allocated: 772.46 MiB
Rank0: After output 0 Allocated: 996.61 MiB; Output memory: 224.16 MiB
Rank1: After output 0 Allocated: 996.61 MiB; Output memory: 224.16 MiB
Rank1: After backward 0 Allocated: 1569.69 MiB; Gradient memory: 573.07 MiB
Rank0: After backward 0 Allocated: 1569.69 MiB; Gradient memory: 573.07 MiB
Rank0: After optimizer 0 Allocated: 2436.00 MiB; Optimizer state memory: 1439.39 MiB
Rank1: After optimizer 0 Allocated: 2275.17 MiB; Optimizer state memory: 1278.55 MiB
```

- 可以看到, 当使用两张卡时, state memory 占用会变成大约原来的 1/2 (不完全是 1/2 的原因: 因为 param 是逐块分配到不同 rank 上的, 因此大小略有差异)

- 同时可以看到 gradient 与 model size 大小大致持平, optimizer state 的大小约为 model size 的 2 倍
- 课上说, 总 memory 约为 model size 的 8 倍, 这是建立在:

```
存储参数本身: FP16
每个参数的梯度: FP16
权重副本: FP32
Adam 一阶动量: FP32
Adam 二阶动量: FP32
```

的假设上, 而现在我们的参数都用 FP32, 不存储 master weights, 因此大约是 4 倍的 model size

与 ZeRO 1 的区别

???

问题

1. **不一致的显存 profile**
   分析 optimizer shard 导致的显存变化时, 发现 Nvidia-smi 监控和 `torch.cuda.memory_allocated(rank)` 得到的显存信息不一致
   特别地, 对于 optimizer shard 的 benchmarking 中, 发现即便真实的显存占用(由 `torch.cuda.memory_allocated(rank)` 反映) shard 会明显低一截, 但是 nvidia-smi 监控的显存占用显示, shard 反而高于 no_shard 的 optimizer
2. **小参数量下forward 和 backward 的时间计算**
   理论上 forward 和 backward 的 FLOPS 约为 1: 2, 实测当 model size == "large", 且 context_length == 512 时测量结果也大约符合(见 **Prob2 python standard library (s)** 部分(0.155: 0.30), 以及 **Prob8 naive torch Attention**部分(51: (127-51)), 以及 **Prob6** 和 **Prob7**).
   但是当参数量不够大时, 对于 TransformerLM 的 timing 显示 forward pass 时间与 backward pass 时间大致相等, 但是对于单个 scale dot Attention 而言, forward pass 时间要远小于 backward pass 时间.