SAMSUNG SDS

Foresee

Techtonic 2021

Disrupt

Partner

GPU Profiling을 통한

Deep Learning 학습 최적화

김성준 프로

GPU Profiling

Profiling?

Profiling (computer programming)[1]

“In software engineering, profiling ("program profiling", "software profiling") is a form of dynamic program analysis that measures, for example, the space (memory) or time complexity of a program, the usage of particular instructions, or the frequency and duration of function calls. Most commonly, profiling information serves to aid program optimization, and more specifically, performance engineering.

Profiling is achieved by instrumenting either the program source code or its binary executable form using a tool called a profiler (or code profiler). Profilers may use a number of different techniques, such as event-based, statistical, instrumented, and simulation methods.”

Ex) cProfile[2] in python

To profile a function that takes a single argument, you can do:

```
import cProfile
import re
cProfile.run('re.compile("foo|bar")')
```

(Use `profile` instead of `cProfile` if the latter is not available on your system.)

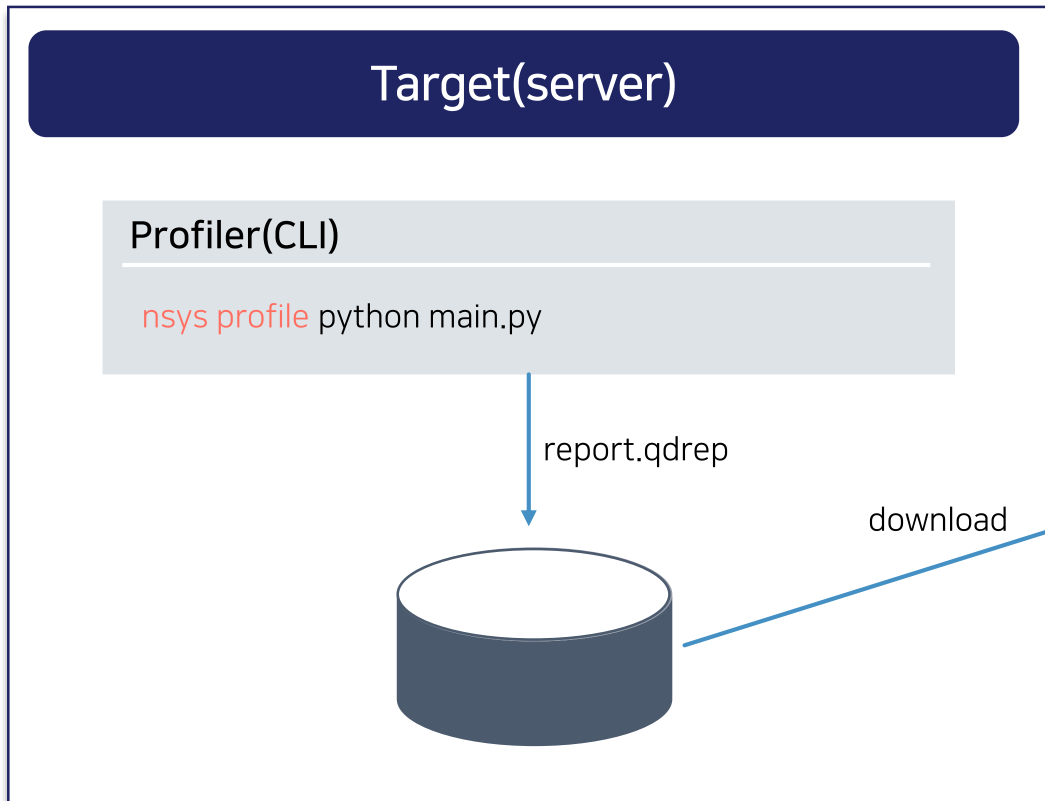
The above action would run `re.compile()` and print profile results like the following:

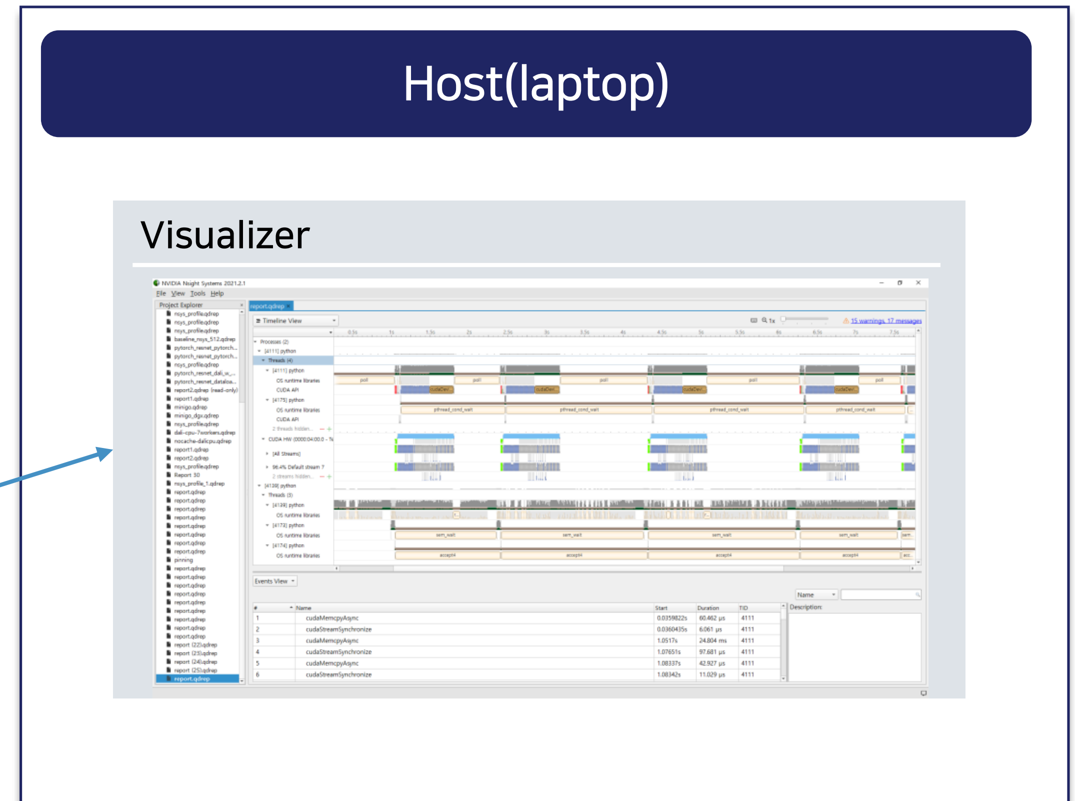
```
      197 function calls (192 primitive calls) in 0.002 seconds

Ordered by: standard name

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    0.000    0.000    0.001    0.001 <string>:1(<module>)
     1    0.000    0.000    0.001    0.001 re.py:212(compile)
     1    0.000    0.000    0.001    0.001 re.py:268(_compile)
     1    0.000    0.000    0.000    0.000 sre_compile.py:172(_compile_charset)
     1    0.000    0.000    0.000    0.000 sre_compile.py:201(_optimize_charset)
     4    0.000    0.000    0.000    0.000 sre_compile.py:25(_identityfunction)
   3/1    0.000    0.000    0.000    0.000 sre_compile.py:33(_compile)
```

[1] https://en.wikipedia.org/wiki/Profiling_(computer_programming)
[2] https://docs.python.org/3/library/profile.html

GPU Profiling - Profiler

Nsight Systems[3] (NVIDIA)





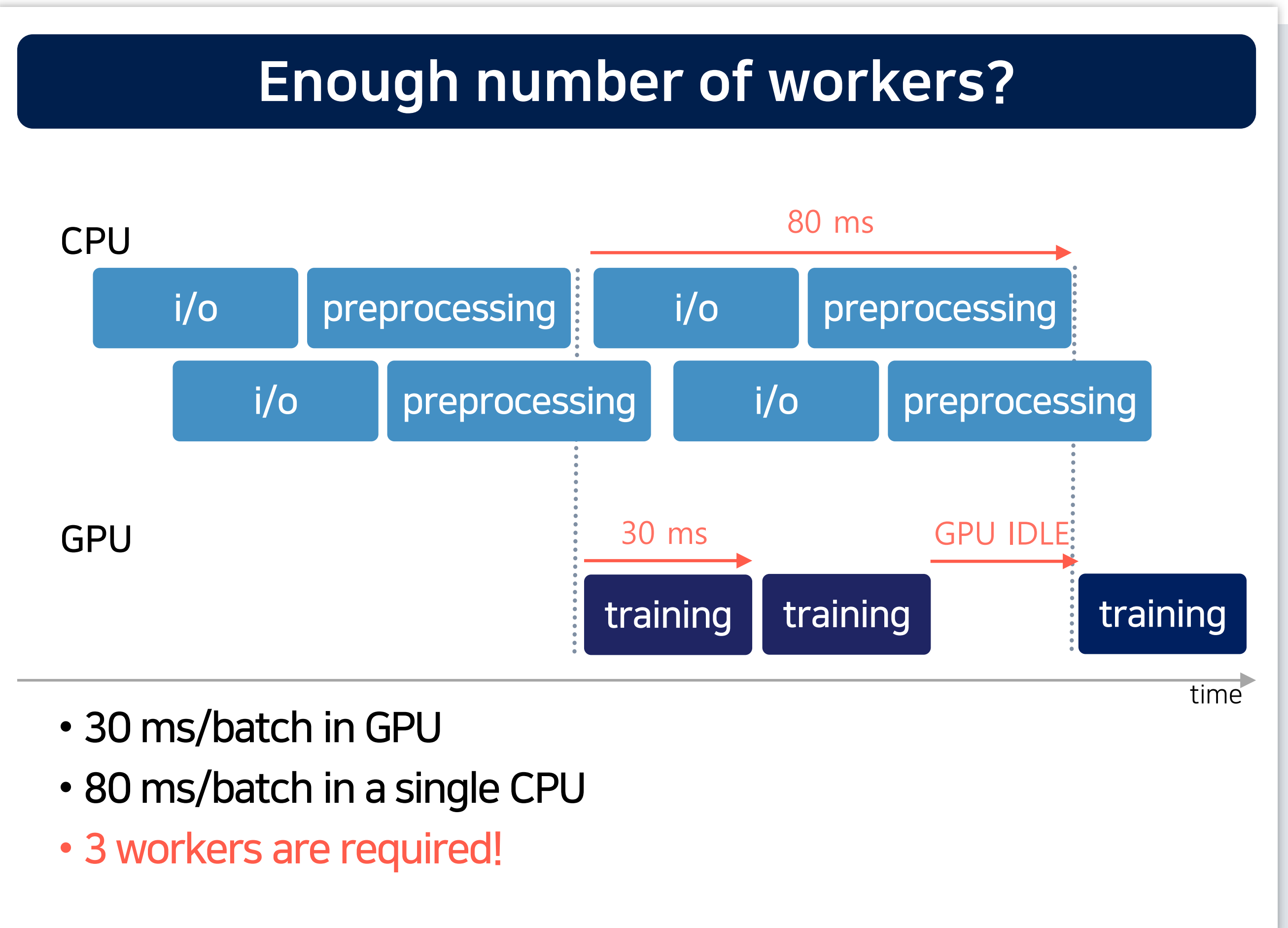
[3] https://developer.nvidia.com/nsight-systems

GPU Profiling - Profiler

What We Expect

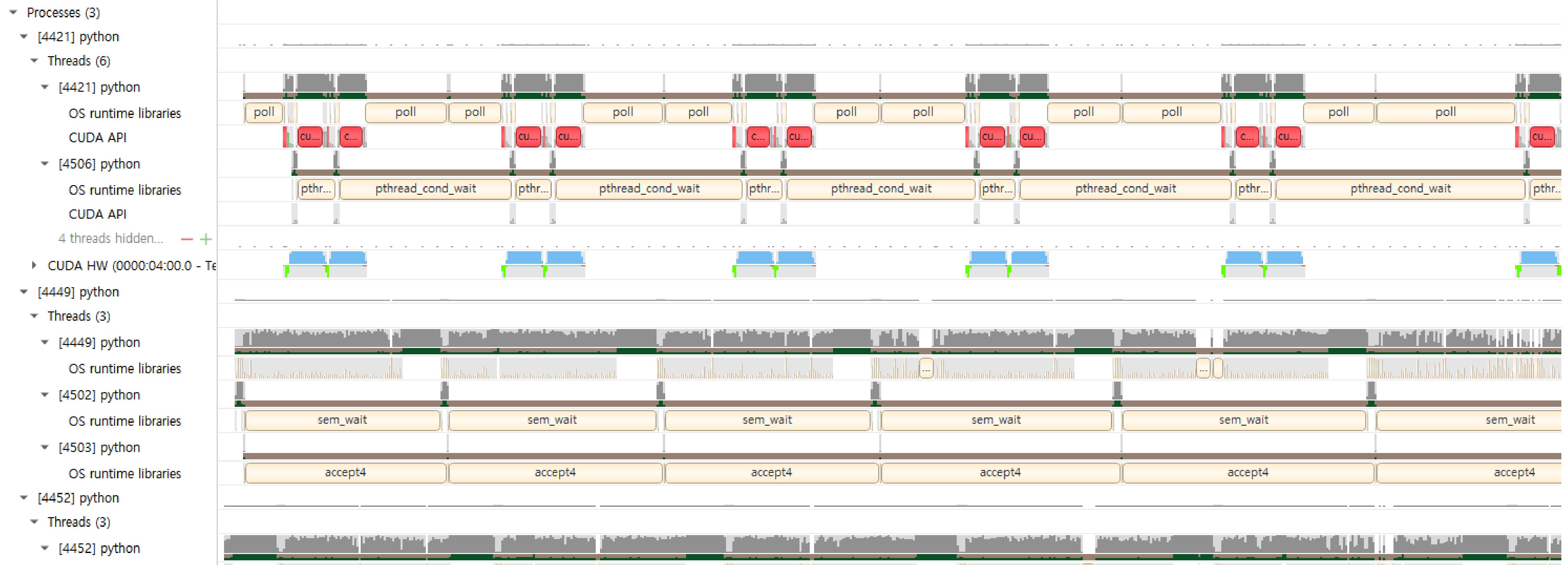
Example – CPU bottleneck

1. GPU Utilization: 50%
2. 2 cpu workers



GPU Profiling - Profiler

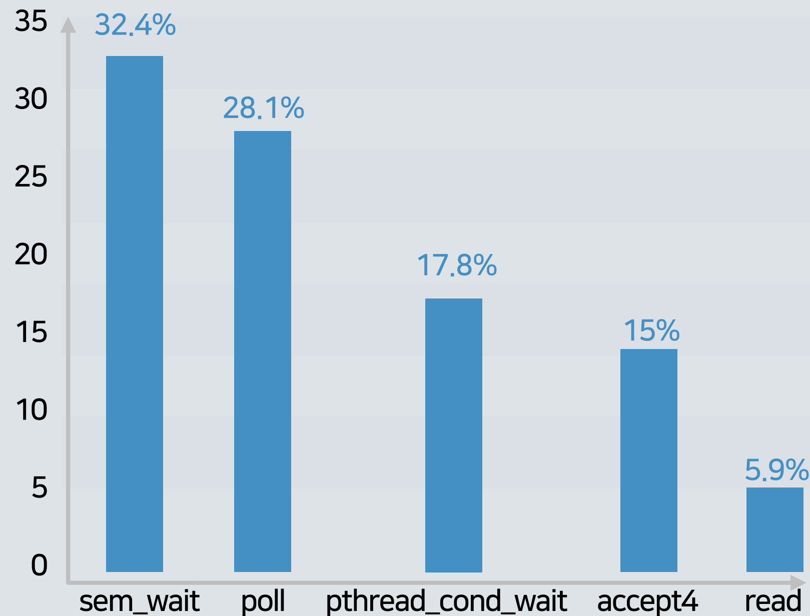
Nsight Systems (NVIDIA)



GPU Profiling – Profiler, Annotations

Nsight Systems + NVTX (NVIDIA)

OS runtime library



and pthread_cond_timedwait, munmap, open64, sched_yield, ...

NVIDIA Tools Extension (NVTX)[4]

release-v3 · 3 branches · 1 tag · Go to file · Add file · Code

jcohen-nvidia Move C API and docs under c/ directory in prep for adding other ... c09c47a on 24 Jun 2020 · 8 commits

c/include/nvtx3	Move C API and docs under c/ directory in prep for adding other	17 months ago
docs	Add docs/index.html to redirect into nested doxygen docs for now	2 years ago
LICENSE.txt	Add LICENSE.txt - verbatim copy of https://llvm.org/LICENSE.txt	2 years ago
README.md	fixing docs link	2 years ago

README.md

NVTX

NVIDIA Tool Extension Library (NVTX)

See https://nvidia.github.io/NVTX/doxygen/index.html for Doxygen documentation.

About

The NVIDIA® Tools Extension SDK (NVTX) is a C-based Application Programming Interface (API) for annotating events, code ranges, and resources in your applications.

Readme

Apache-2.0 License

Releases

1 tags

Packages

No packages published

[4] https://github.com/NVIDIA/NVTX

GPU Profiling - Profiler, Annotations

Nsight Systems + NVTX (NVIDIA)

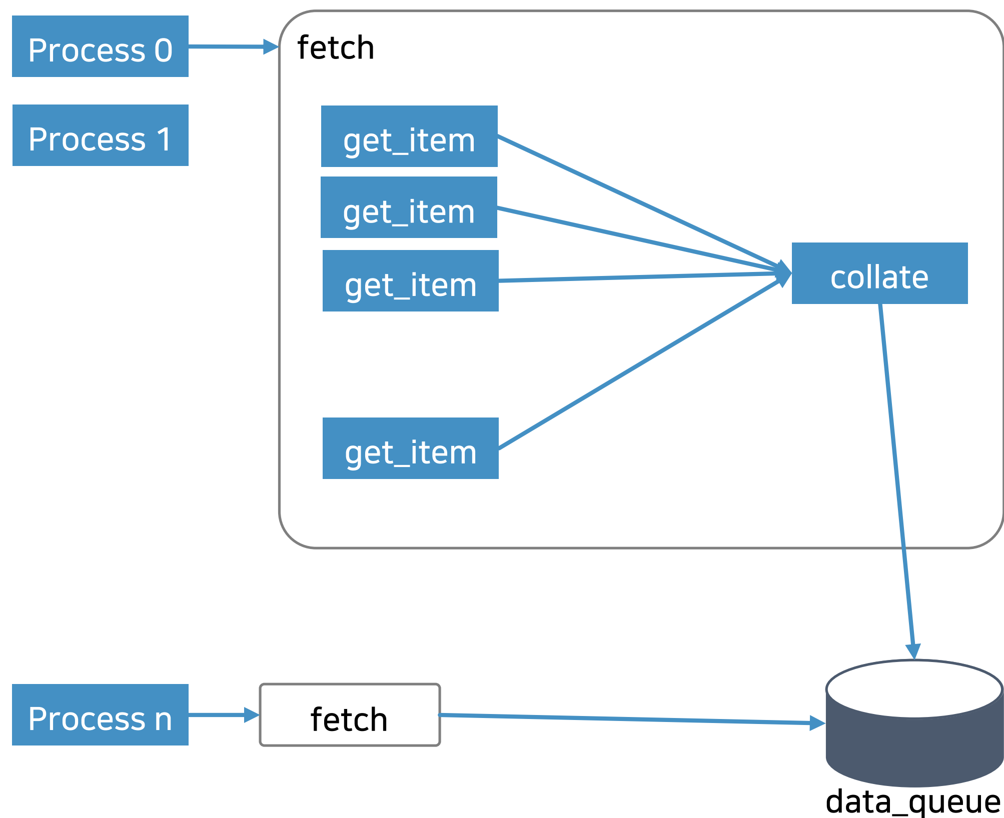
torch.cuda.nvtx

Simple python interface

```
for i, (input, target) in data_iter:
    bs = input.size(0)
    lr_scheduler(optimizer, i, epoch)
    data_time = time.time() - end

    optimizer_step = ((i + 1) % batch_size_multiplier) == 0

    from torch.cuda import nvtx

    nvtx.range_push(f'step {i}')
    loss = step(input, target, optimizer_step=optimizer_step)
    nvtx.range_pop()
```

GPU Profiling - Profiler, Annotations

NVTX Range in Pytorch Dataloader



Worker.py in pytorch

```
        try:
            data = fetcher.fetch(index)
        except Exception as e:
            if isinstance(e, StopIteration) and dataset_kind == _DatasetKind.Iterable:
                data = _IterableDatasetStopIteration(worker_id)
                # Set `iteration_end`
                #   (1) to save future `next(...)` calls, and
                #   (2) to avoid sending multiple `_IterableDatasetStopIteration`s.
                iteration_end = True
            else:
                # It is important that we don't store exc_info in a variable.
                # `ExceptionWrapper` does the correct thing.
                # See NOTE [ Python Traceback Reference Cycle Problem ]
                data = ExceptionWrapper(
                    where="in DataLoader worker process {}".format(worker_id))
    data_queue.put((idx, data))
    del data, idx, index, r  # save memory
```

GPU Profiling - Profiler, Annotations

Monkey Patch

Monkey Patch[5]

"The definition of the term varies depending upon the community using it. In Ruby, Python, and many other dynamic programming languages, the term monkey patch only refers to dynamic modifications of a class or module at runtime, motivated by the intent to patch existing third-party code as a workaround to a bug or feature which does not act as desired."

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.pi = 3.2   # monkey-patch the value of Pi in the math module
>>> math.pi
3.2
```

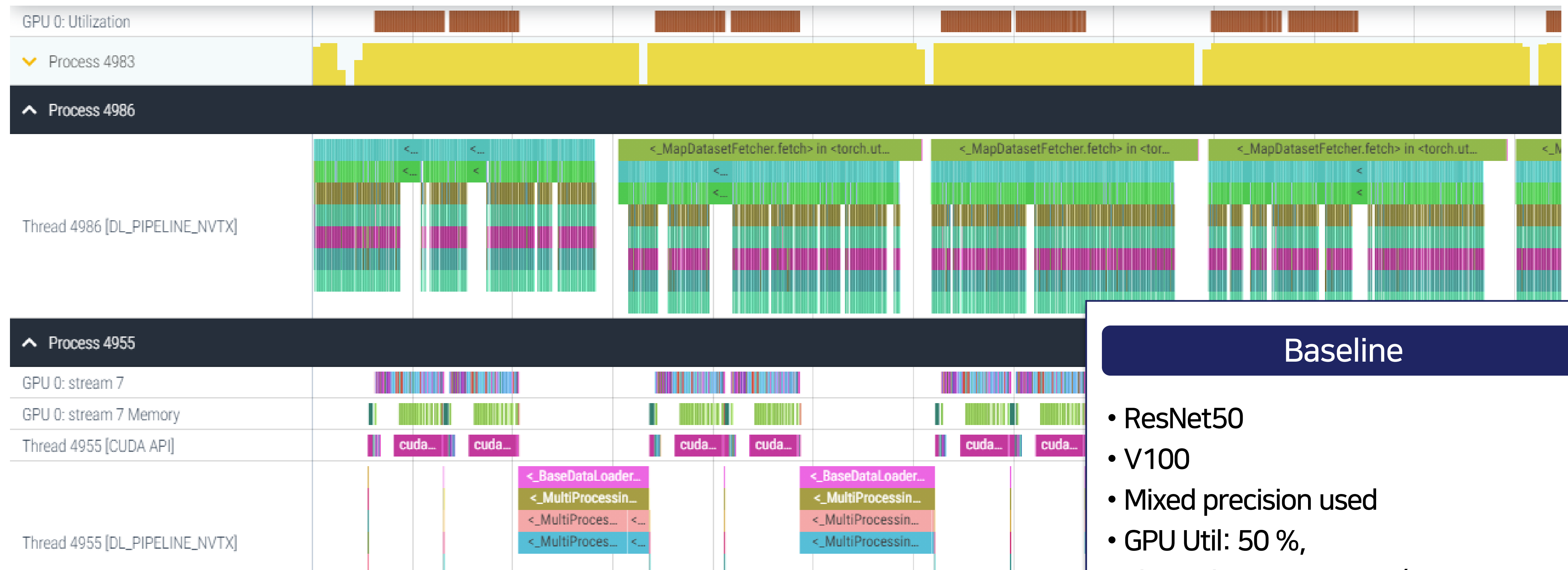
[5] https://en.wikipedia.org/wiki/Monkey_patch

Monkey Patch

```
from torch.cuda import nvtx


def monkey_patch(mod, func_name):
    func = getattr(mod, func_name)
    msg = f'<{func_name}> in <{mod.__name__}>'

    def wrapper_func(*args, **kwargs):
        nvtx.range_push(msg)
        result = func(*args, **kwargs)
        nvtx.range_pop()
        return result

    setattr(mod, func_name, wrapper_func)
```

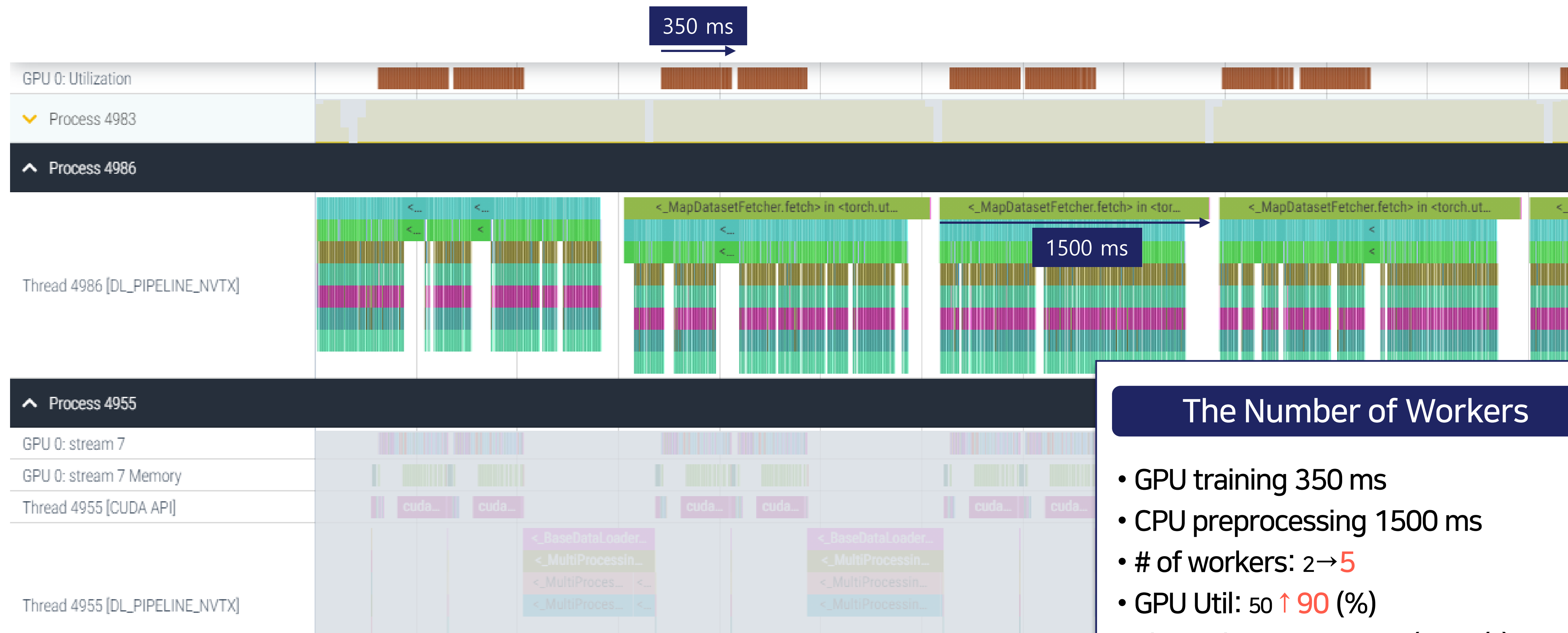
ImageNet Training

Baseline



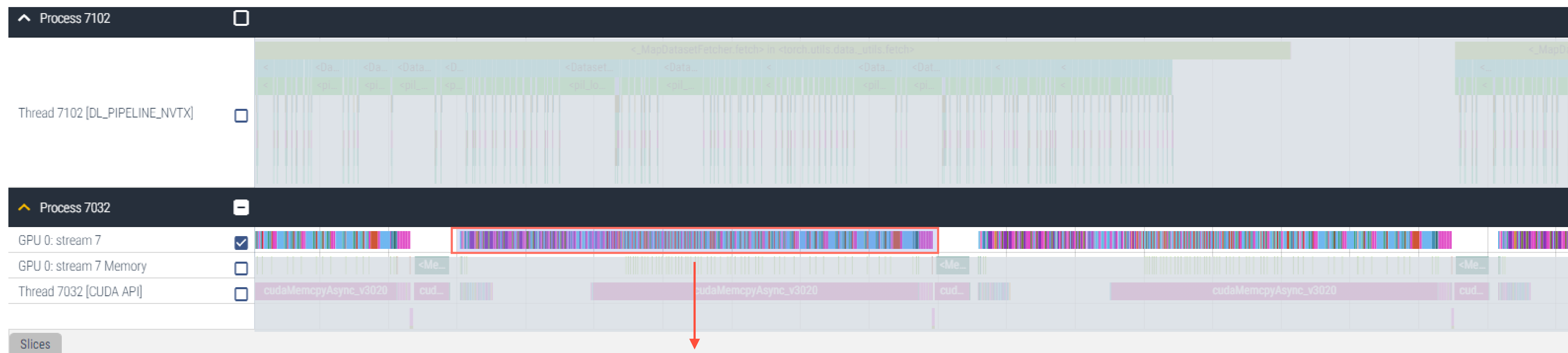
ImageNet Training

Insufficient Parallelism



ImageNet Training

Memory Format



Name	Wall duration (ms)	Avg Wall duration (ms)	Occurrences
	335.319796		1921
bn_bw_1C11_kernel_new	70.033334	1.321383	53
vectorized_elementwise_kernel	61.843018	0.057421	1077
nchwToNhwcKernel	46.474131	0.217168	214
bn_fw_tr_1C11_kernel_NCHW	36.321981	0.68532	53
kernel	20.756717	0.532223	39
volta_fp16_s884cudnn_fp16_256x128_ldg8_relu_filter1x1_stg8_interior_nchw_nn_v1	13.703038	0.856439	16
volta_fp16_s884cudnn_fp16_128x128_ldg8_relu_f2f_exp_small_nhwc2nchw_tn_v1	12.098441	0.672135	18

ImageNet Training

Memory Format

NCHW vs NHWC

According to the NVIDIA documentation[6],

1. Convolution algorithms implemented for Tensor Cores require channels last memory format(NHWC).

2. Contiguous memory format(NCHW) can be used.

   But, due to automatic transpose operations, there will be some overhead.

Where, N is the batch size,

C is the number of feature maps,

H, W are the height and width of the image.

[6] https://docs.nvidia.com/deeplearning/performance/dl-performance-convolutional/index.html

Example Tensor N=1, C=3, H=5, W=4

C0

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19

C1

20	21	22	23
24	25	26	27
28	29	30	31
32	33	34	35
36	37	38	39

C2

40	41	42	43
44	45	46	47
48	49	50	51
52	53	54	55
56	57	58	59

NCHW

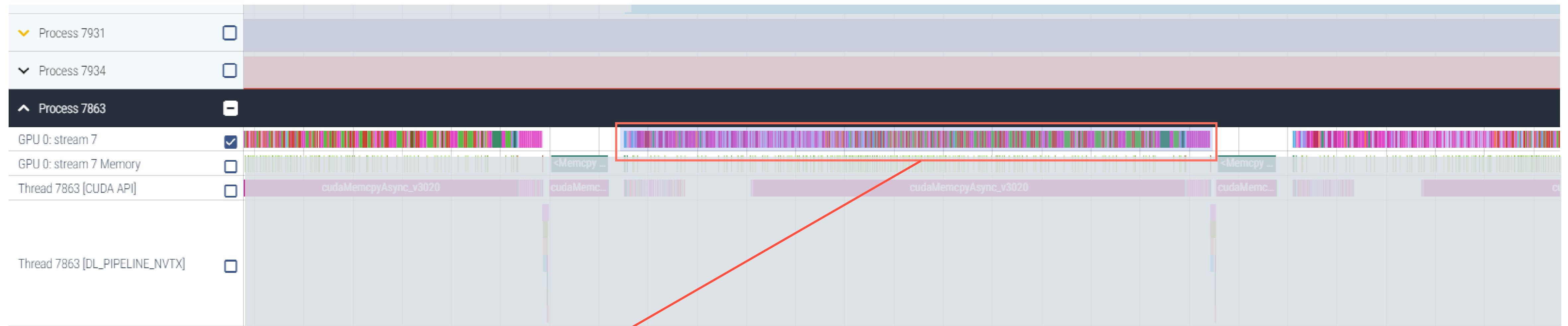
C0					C1					C2				
0	1	2	···	19	20	21	22	···	39	40	41	42	···	59

NHWC

C0	C1	C2	C0	C1	C2	C0	C1	C2	··········	C0	C1	C2
0	20	40	1	21	41	2	22	42		19	39	59

ImageNet Training

Memory format



Slices

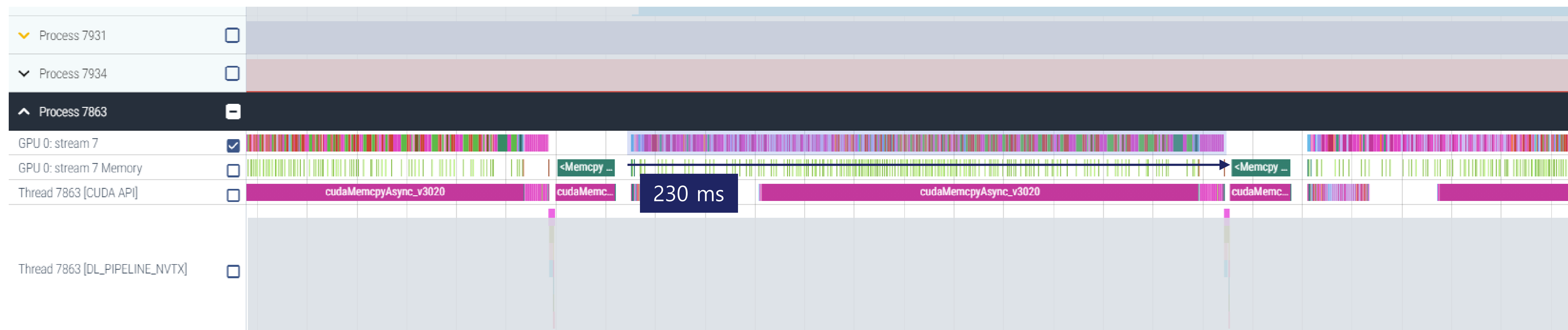
Name	Wall duration (ms)	Avg Wall duration (ms)
	229.449458	
vectorized_elementwise_kernel	61.730578	0.057317
batchnorm_bwtr_nhwc_semiPersist	35.667724	0.672975
kernel	34.150623	0.550816
batchnorm_fwtr_nhwc_semiPersist	24.126499	0.455216
sm70_xmma_fprop_implicit_gemm_f16f16_f16f32_f32_nhwckrsc_nhwc_tilesize128x128x32_stage1_warpsize2x2x1_g1_tensor8x8x4_t1r1s1_kernel	16.032499	0.552844
sm70_xmma_fprop_implicit_gemm_f16f16_f16f32_f32_nhwckrsc_nhwc_tilesize256x128x32_stage1_warpsize4x2x1_g1_tensor8x8x4_t1r3s3_kernel	6.248283	0.694253

Memory format

- Memory format: NCHW→NHWC
- No nchwTonhwcKernel
- GPU Util: 90 ↓ 68 (%)
- Throughput: 690 ↑ 850 (imgs/s)

ImageNet Training

Memory format



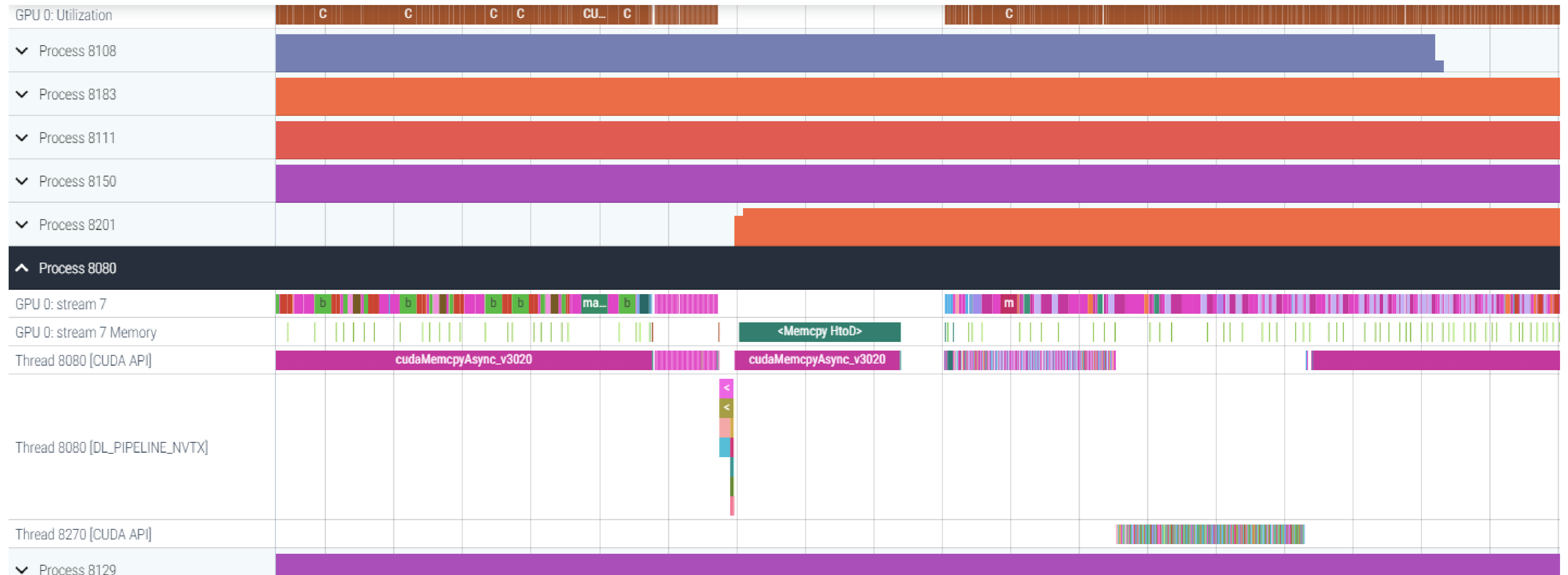
Name	Wall duration (ms)	Avg Wall duration (ms)
	229.449458	
vectorized_elementwise_kernel	61.730578	0.057317
batchnorm_bwtr_nhwc_semiPersist	35.667724	0.672975
kernel	34.150623	0.550816
batchnorm_fwtr_nhwc_semiPersist	24.126499	0.455216
sm70_xmma_fprop_implicit_gemm_f16f16_f16f32_f32_nhwckrsc_nhwc_tilesize128x128x32_stage1_warpsize2x2x1_g1_tensor8x8x4_t1r1s1_kernel	16.032499	0.552844
sm70_xmma_fprop_implicit_gemm_f16f16_f16f32_f32_nhwckrsc_nhwc_tilesize256x128x32_stage1_warpsize4x2x1_g1_tensor8x8x4_t1r3s3_kernel	6.248283	0.694253

The Number of Workers

- GPU training 350 ms → 230 ms
- CPU preprocessing 1500 ms
- # of workers: 5→7
- GPU Util: 68 ↑ 83 (%)
- Throughput: 850 ↑ 950 (imgs/s)

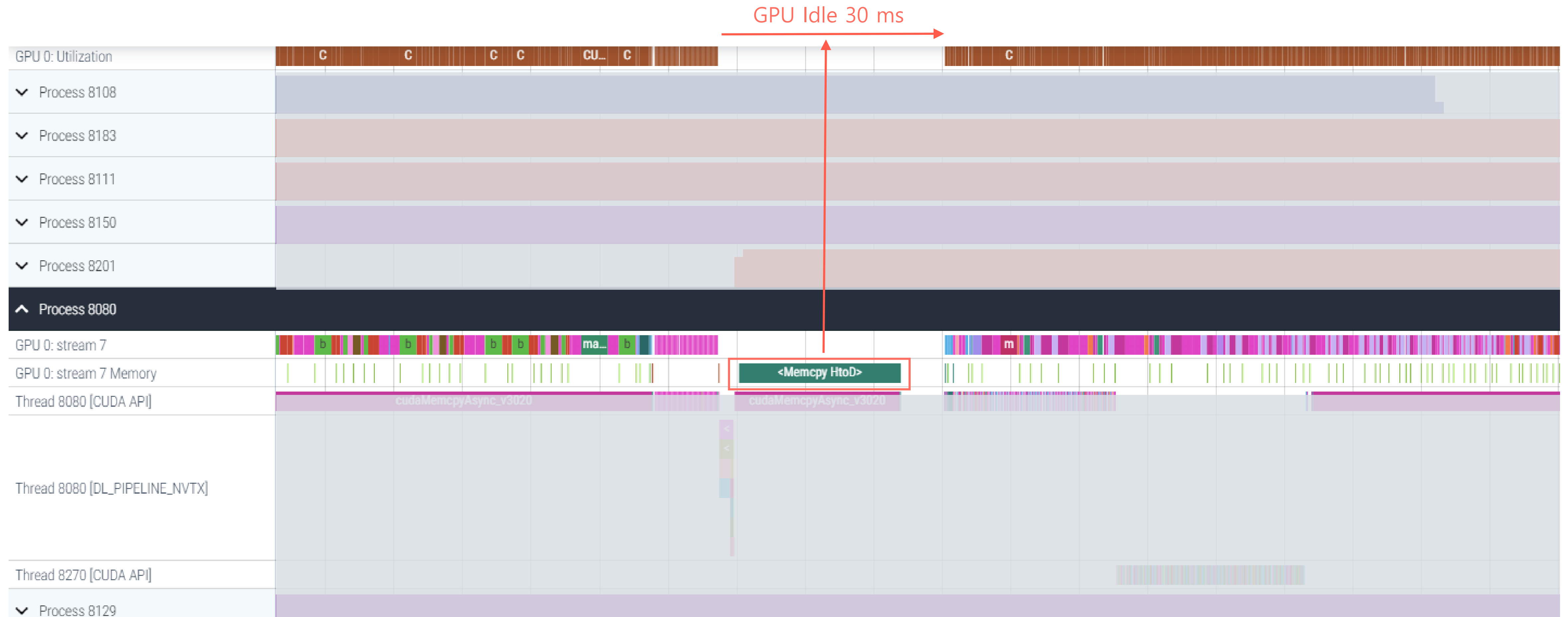
ImageNet Training

Memory Pinning



ImageNet Training

Memory Pinning



ImageNet Training

Memory Pinning

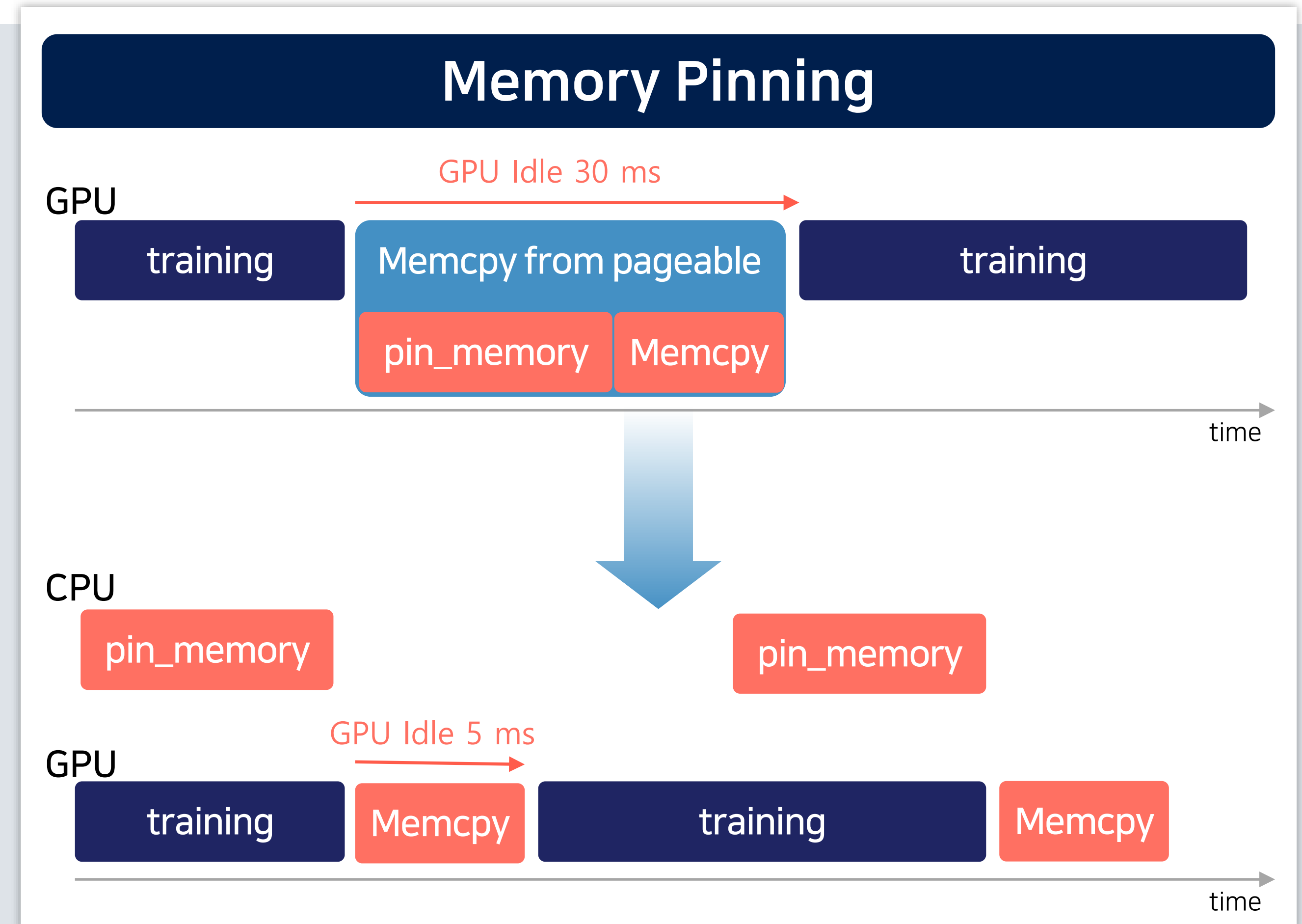
Use pinned memory buffers[7]

"Host to GPU copies are much faster when they originate from pinned (page-locked) memory. CPU tensors and storages expose a pin_memory() method, that returns a copy of the object, with data put in a pinned region.

Also, once you pin a tensor or storage, you can use asynchronous GPU copies. Just pass an additional non_blocking=True argument to a to() or a cuda() call. This can be used to overlap data transfers with computation.

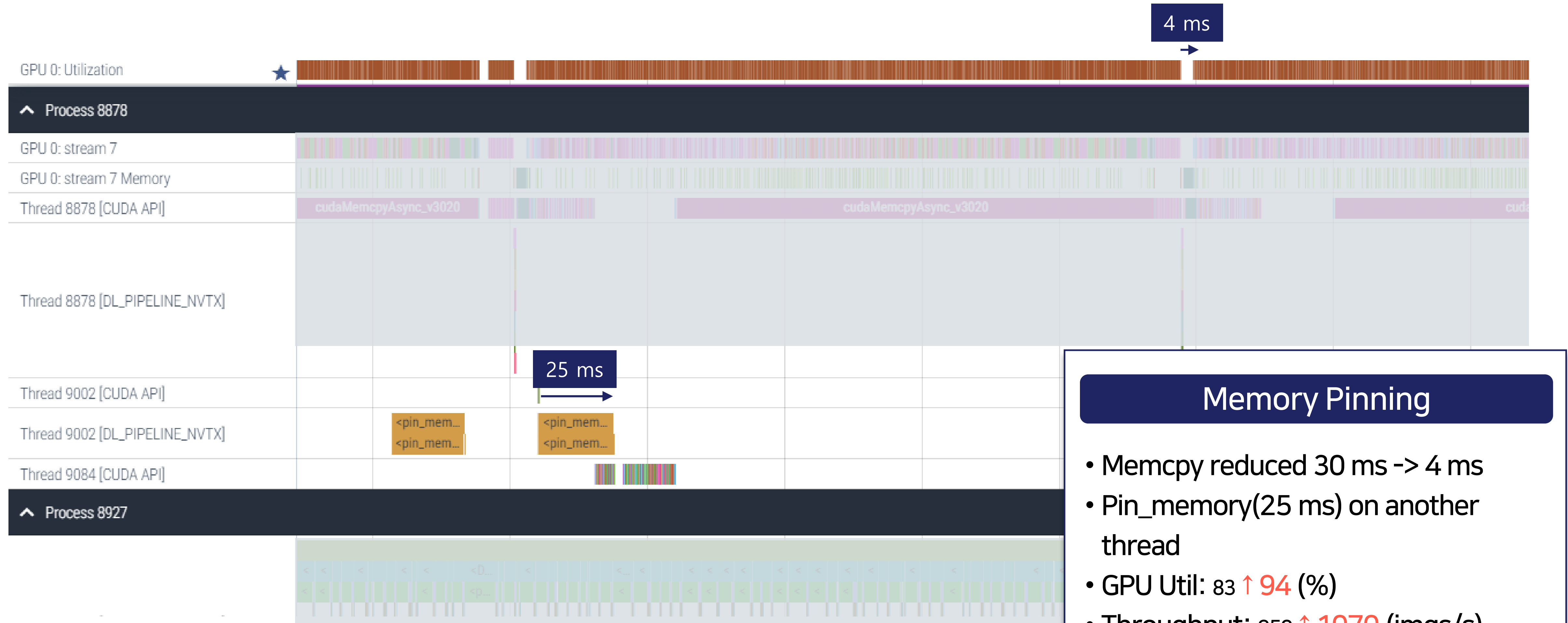
…"

[7] https://pytorch.org/docs/stable/notes/cuda.html#cuda-memory-pinning



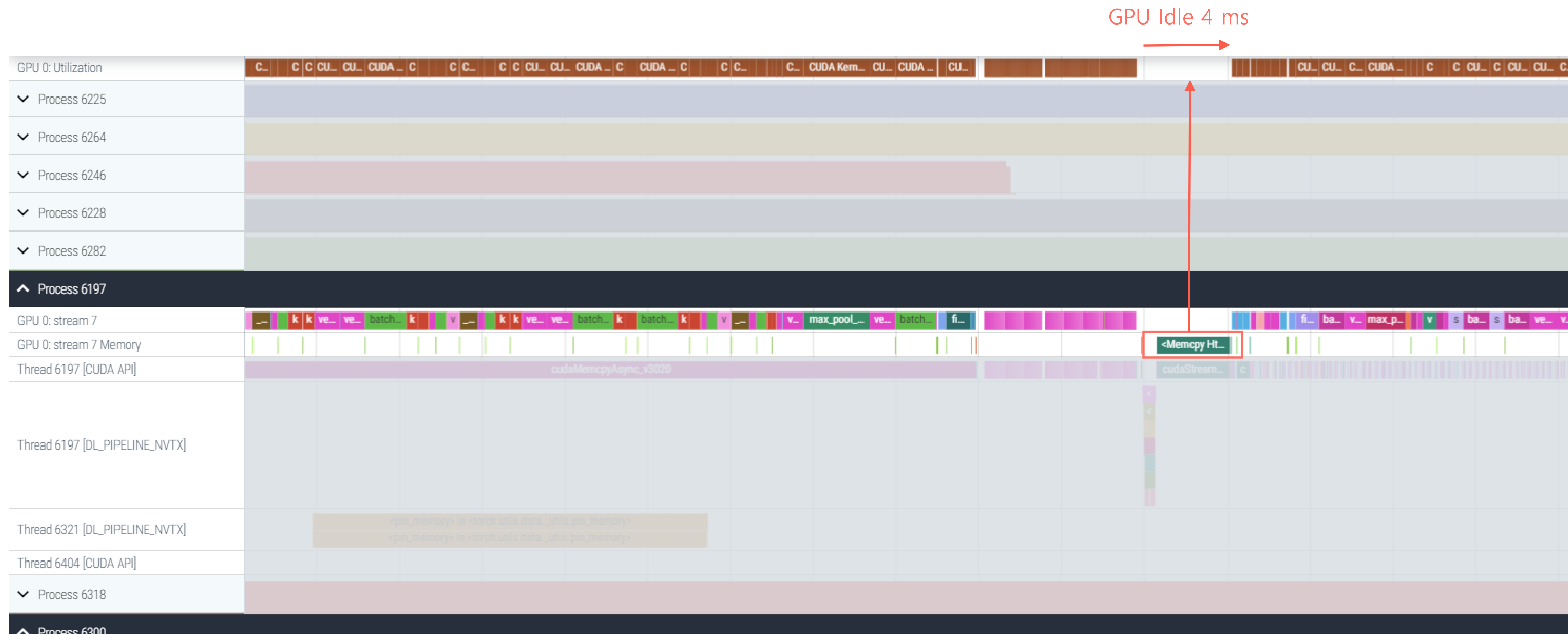
ImageNet Training

Memory Pinning



ImageNet Training

Asynchronous Memory Copy



ImageNet Training

Asynchronous Memory Copy

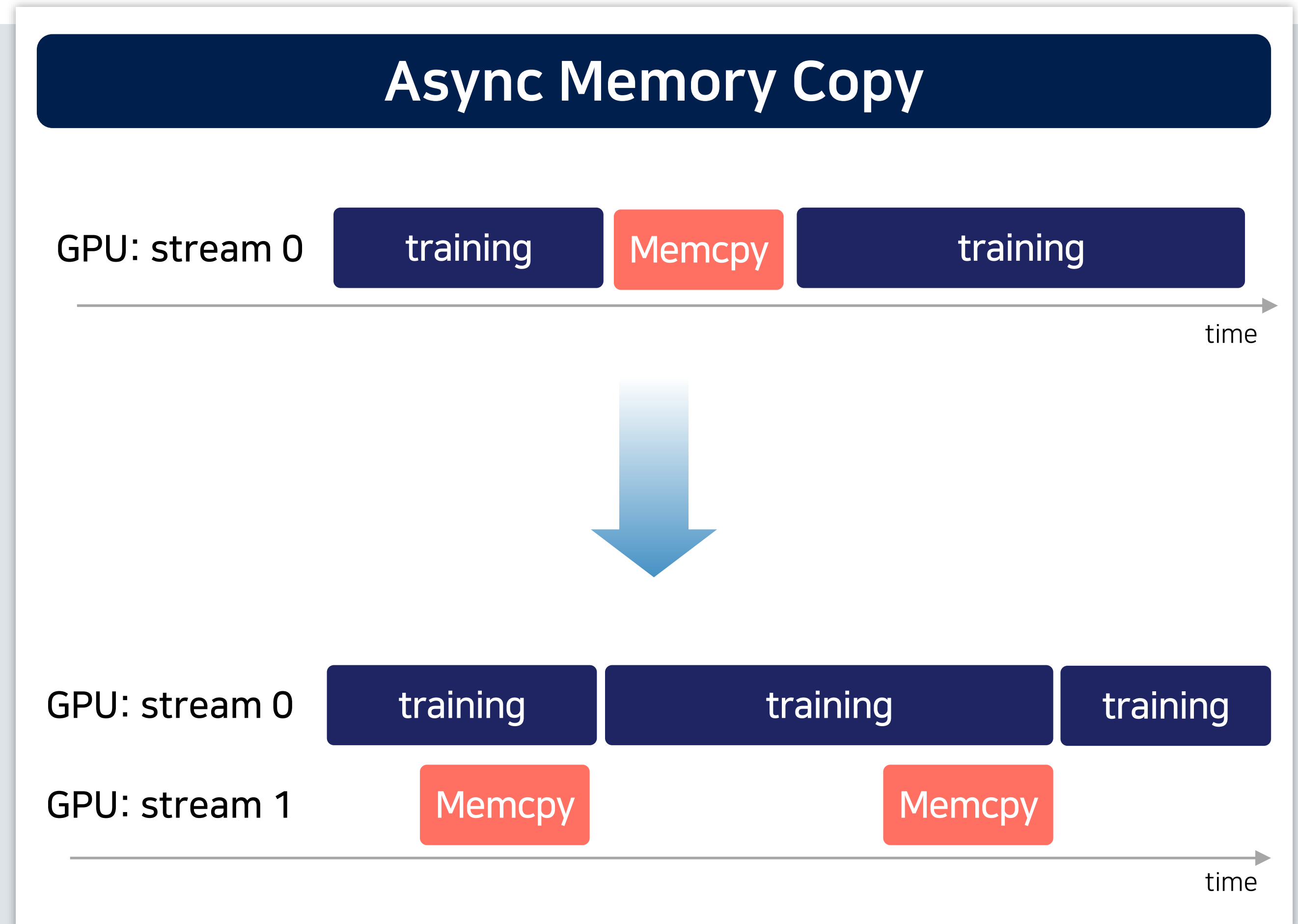
CUDA streams[8]

"A CUDA stream is a linear sequence of execution that belongs to a specific device. You normally do not need to create one explicitly: by default, each device uses its own "default" stream.

Operations inside each stream are serialized in the order they are created, but operations from different streams can execute concurrently in any relative order, unless explicit synchronization functions (such as synchronize() or wait_stream()) are used.

…"

[8] https://pytorch.org/docs/stable/notes/cuda.html



ImageNet Training

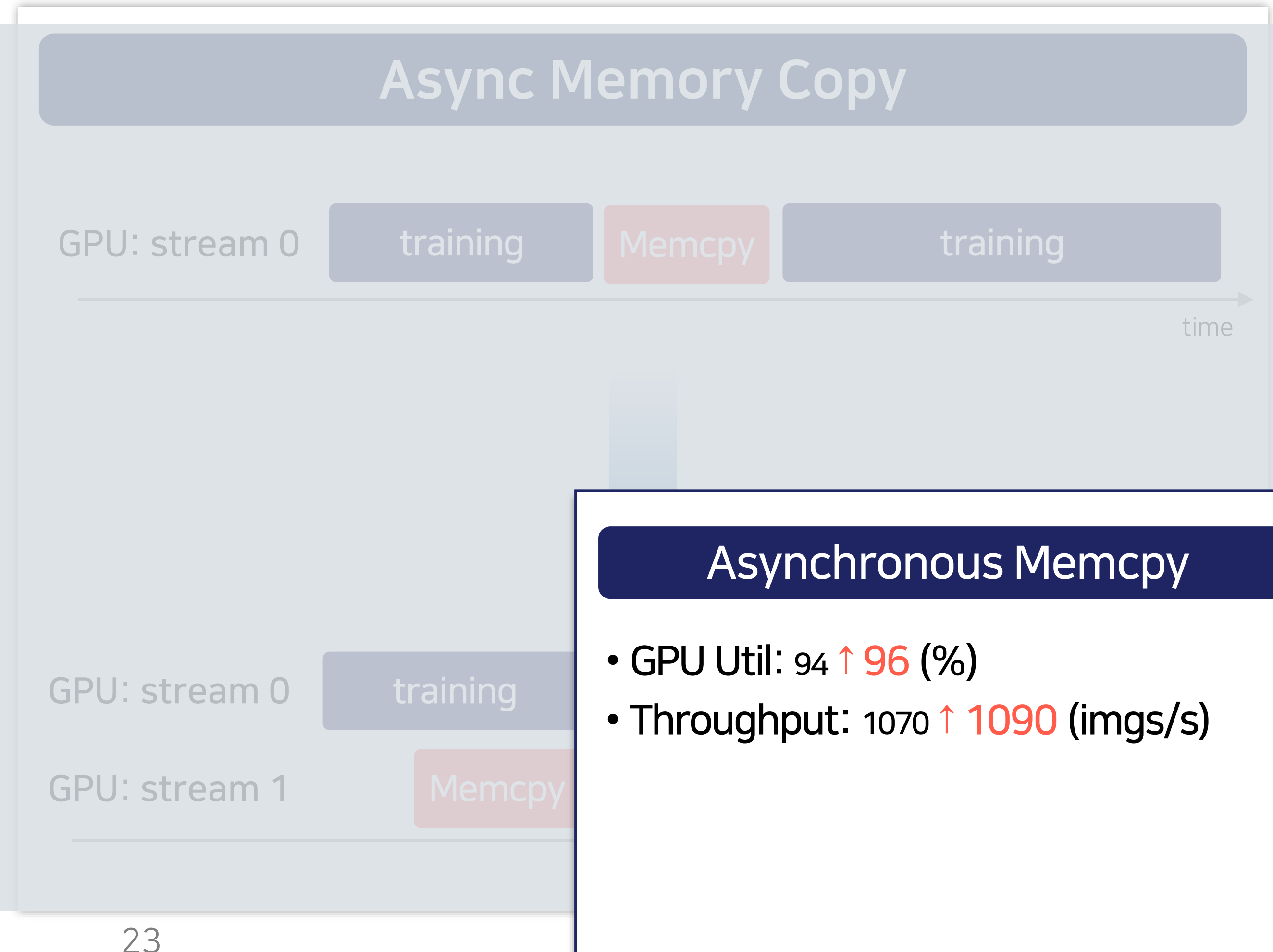
Asynchronous Memory Copy

CUDA streams[8]

"A CUDA stream is a linear sequence of execution that belongs to a specific device. You normally do not need to create one explicitly: by default, each device uses its own "default" stream.

Operations inside each stream are serialized in the order they are created, but operations from different streams can execute concurrently in any relative order, unless explicit synchronization functions (such as synchronize() or wait_stream()) are used.

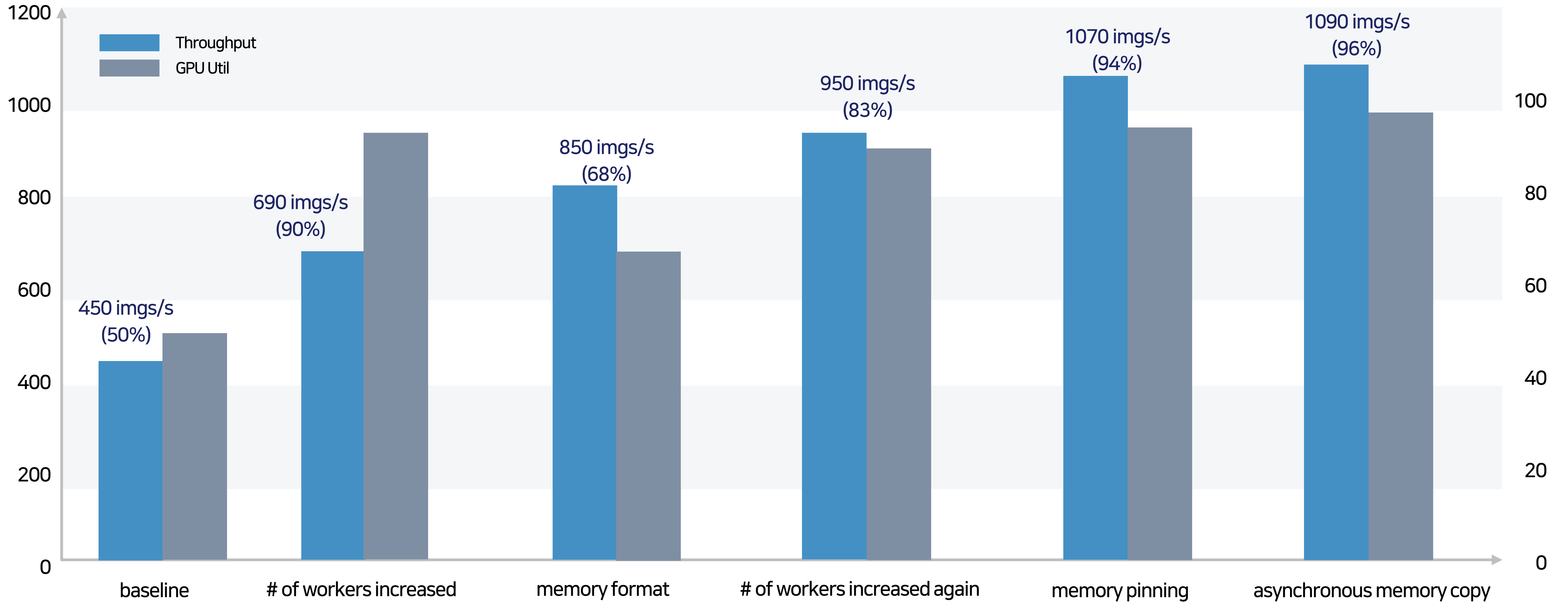
..."



[8] https://pytorch.org/docs/stable/notes/cuda.html

ImageNet Training

Summary



Thank you

SAMSUNG SDS