servicenow

GPU Profiling for AI

MontréalPython-89

Christian Hudon
Advanced Technologies Group

Think Before You ~~Leap~~ Profile

Use the Tools, Luke!

Sampling profilers: your new best friends!

3

USE THE TOOLS, LUKE!

GPU Profiler: NVIDIA Nsight Systems

developer.nvidia.com/nsight-systems

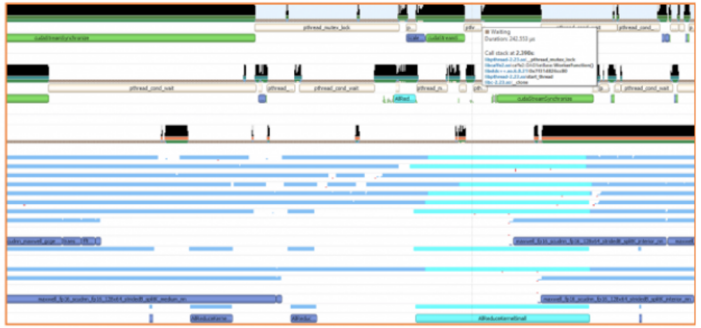
NVIDIA. DEVELOPER HOME BLOG NEWS FORUMS DOCS DOWNLOADS TRAINING ACCOUNT

SOLUTIONS PLATFORMS DOCUMENTATION DOWNLOADS RESOURCES COMMUNITY

Home > Tools

NVIDIA Nsight Systems

NVIDIA® Nsight™ Systems is a system-wide performance analysis tool designed to visualize an application's algorithms, help you identify the largest opportunities to optimize, and tune to scale efficiently across any quantity or size of CPUs and GPUs; from large server to our smallest SoC.



Download Now

Overview

NVIDIA Nsight Systems is a low overhead performance analysis tool designed to provide insights developers need to optimize their software. Unbiased activity data is visualized within the tool to help users investigate bottlenecks, avoid inferring false-positives, and pursue optimizations with higher probability of performance gains. Users will be able to identify issues, such as GPU starvation, unnecessary GPU synchronization, insufficient CPU parallelizing, and even unexpectedly expensive algorithms across the CPUs and GPUs of

USE THE TOOLS, LUKE!

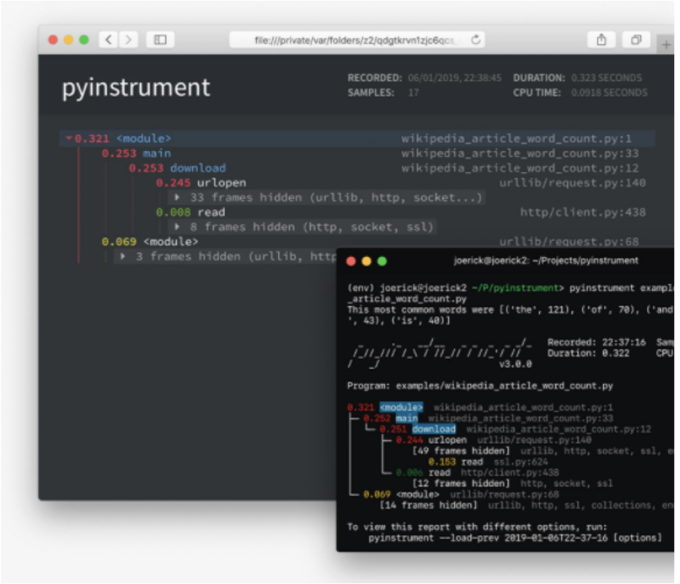
Python Profiler

PyInstrument / Py-Spy / Scalene

github.com/joerick/pyinstrument

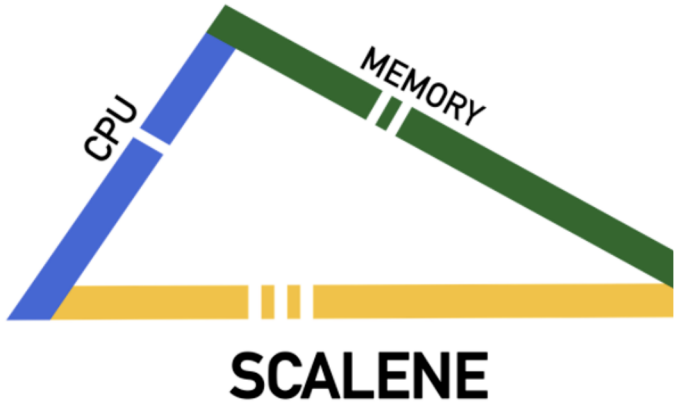
pyinstrument

pypi package 3.2.0 | build passing



Pyinstrument is a Python profiler. A profiler is a tool to help you 'optimize' your c
faster. It sounds obvious, but to get the biggest speed increase you should focus
slowest part of your program. Pyinstrument helps you find it!

github.com/emeryberger/scalene



scalene: a high-performance CPU and m
profiler for Python

by Emery Berger

downloads 1.6k/month | python 3.6 | 3.7 | 3.8 | 3.9 | license Apache-2.0

中文版本 (Chinese version)

About Scalene

github.com/benfred/py-spy

py-spy: Sampling profiler for Python programs

build passing | build passing | build passing

py-spy is a sampling profiler for Python programs. It lets you visualize what your Python program is spending time on without restarting the program or modifying the code in any way. py-spy is extremely low overhead: it is written in Rust for speed and doesn't run in the same process as the profiled Python program. This means py-spy is safe to use against production Python code.

py-spy works on Linux, OSX, Windows and FreeBSD, and supports profiling all recent versions of the CPython interpreter (versions 2.3-2.7 and 3.3-3.8).

Installation

Prebuilt binary wheels can be installed from PyPI with:

```
pip install py-spy
```

You can also download prebuilt binaries from the GitHub Releases Page. This includes binaries for ARM and FreeBSD, which can't be installed using pip. If you're a Rust user, py-spy can also be installed with: `cargo install py-spy`. On Arch Linux, py-spy is in AUR and can be installed with `yay -S py-spy`.

Usage

py-spy works from the command line and takes either the PID of the program you want to sample from or the command line of the python program you want to run. py-spy has three

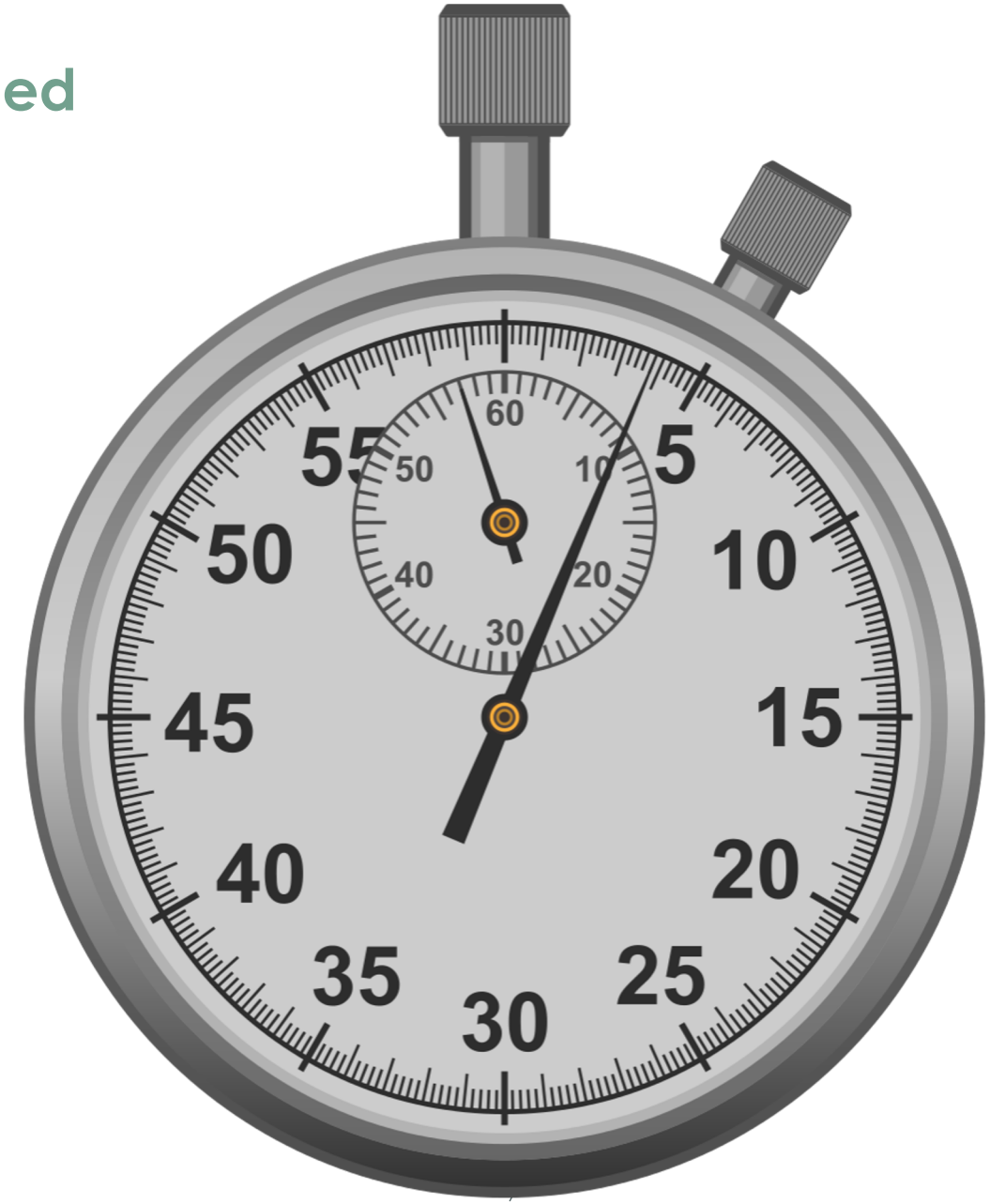
How to Prepare

Quick, repeatable iterations

6

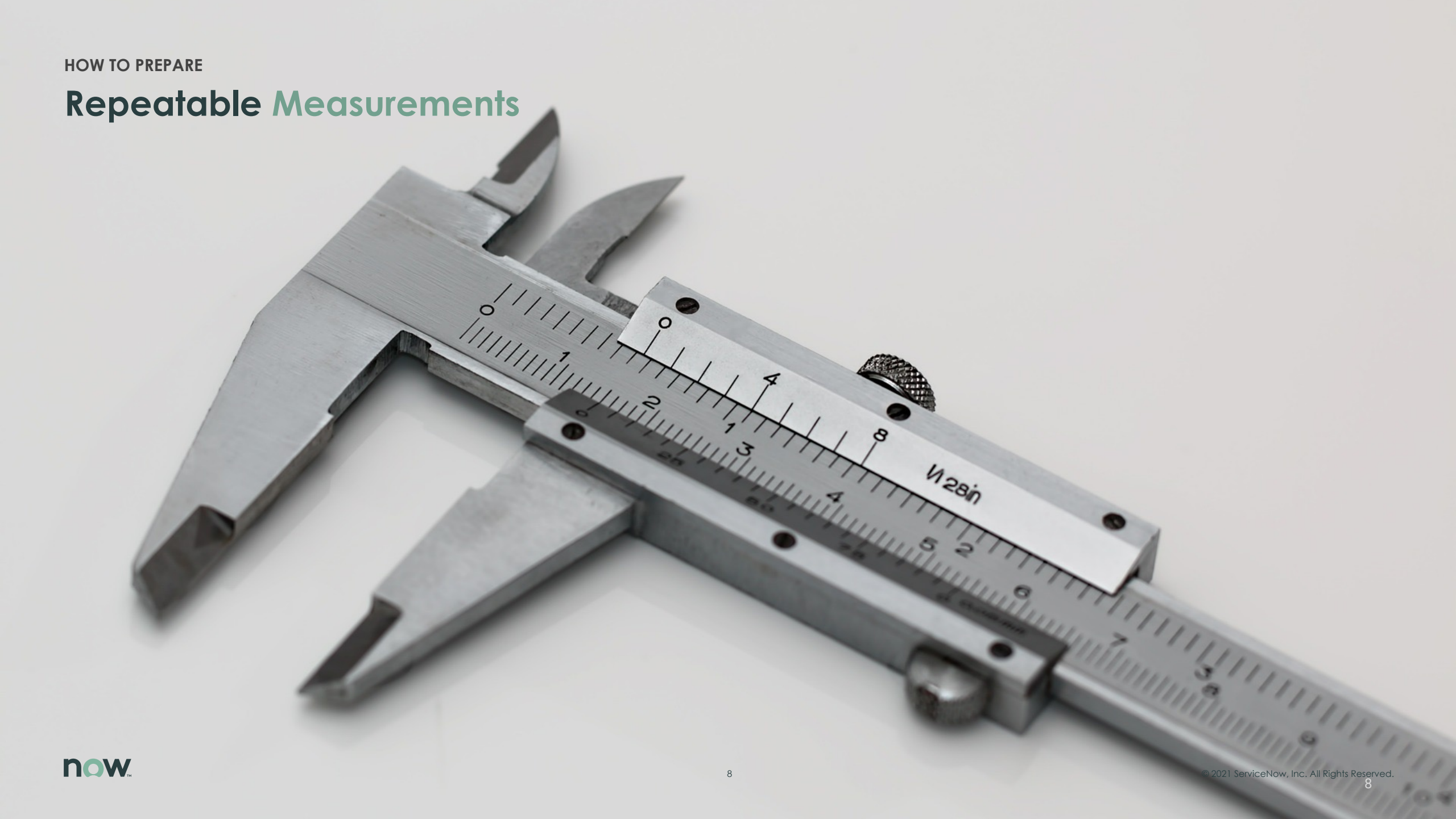
HOW TO PREPARE

Quicker Iteration Speed



HOW TO PREPARE

Repeatable Measurements



8

Add NVTX Annotations to Your Code

1 How?

```
from torch.cuda import nvtx

# With PyTorch 1.8+, you can simply do:

with nvtx.range("Some event"):
   # Code here...

# Or even use it as a decorator:

class MyModel(nn.Module):

   # Other methods here...

   @nvtx.range("MyModel.forward()")
   def forward(self, *input):
      # Forward pass code here...
```

2 What?

- Dataset load
- Model initialization
- Mini-batch creation
- Train (forward, backward passes)
- Test

Where to Focus

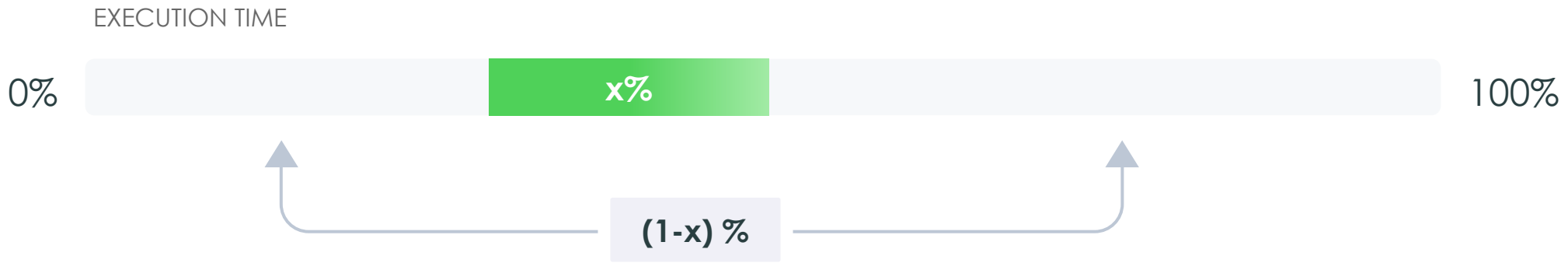
Spend time where it will pay off!

10

WHERE TO FOCUS

Amdahl's Law*

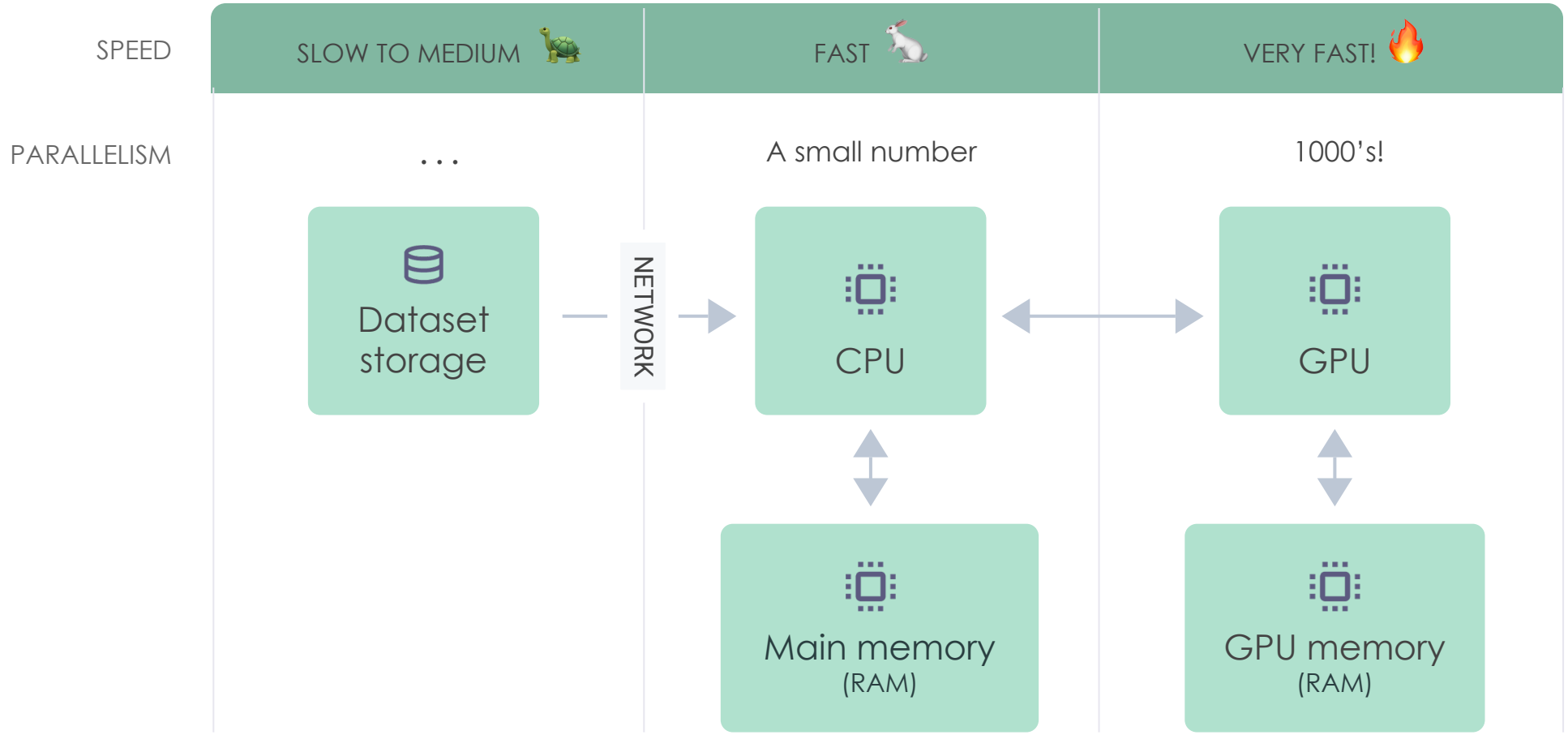
Vintage CompSci!



* See Wikipedia article

11

The System View & the Bottlenecks



Lessons Learned

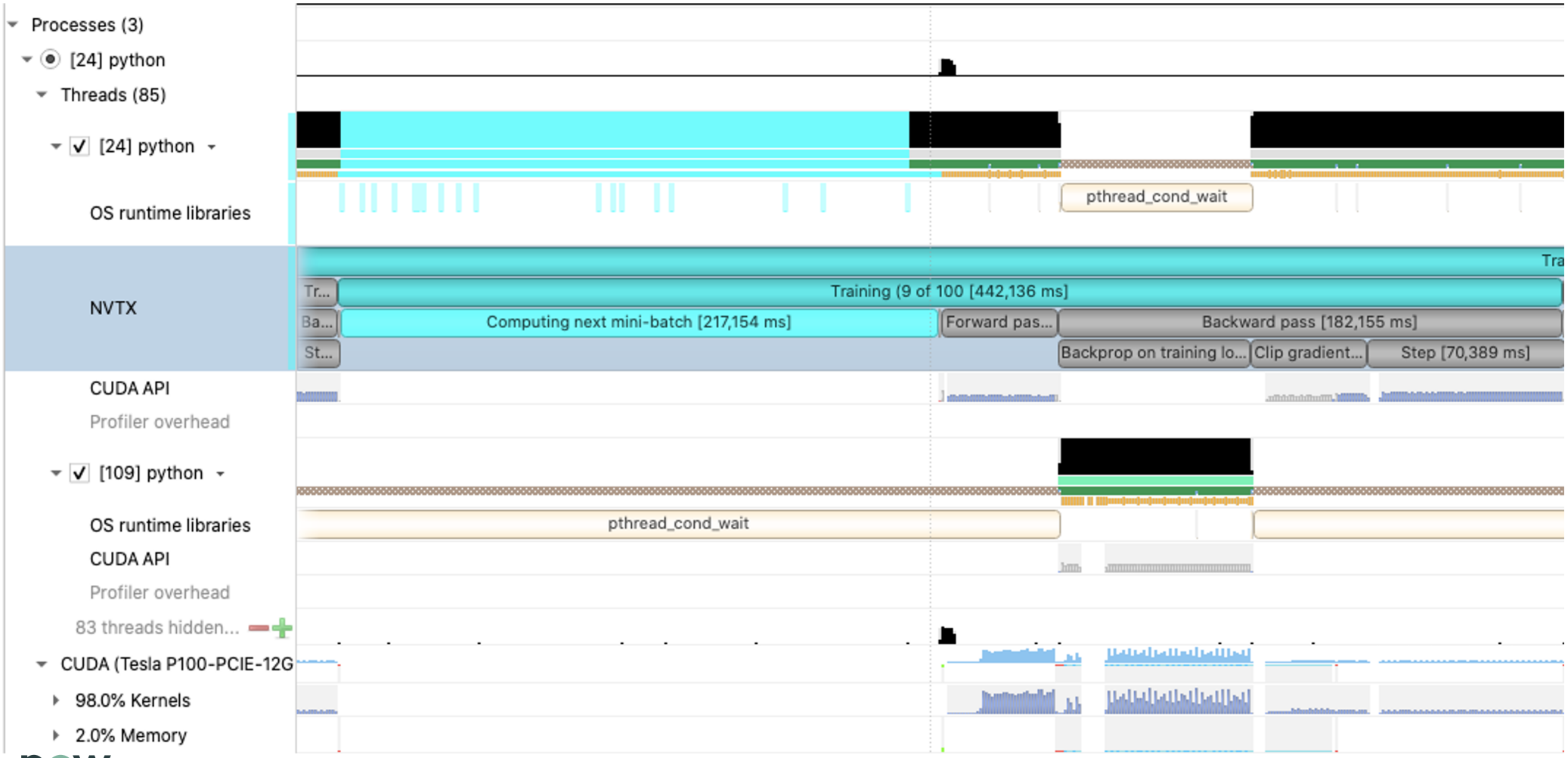
Keep up with the GPU

Lots of cores, fast memory

14

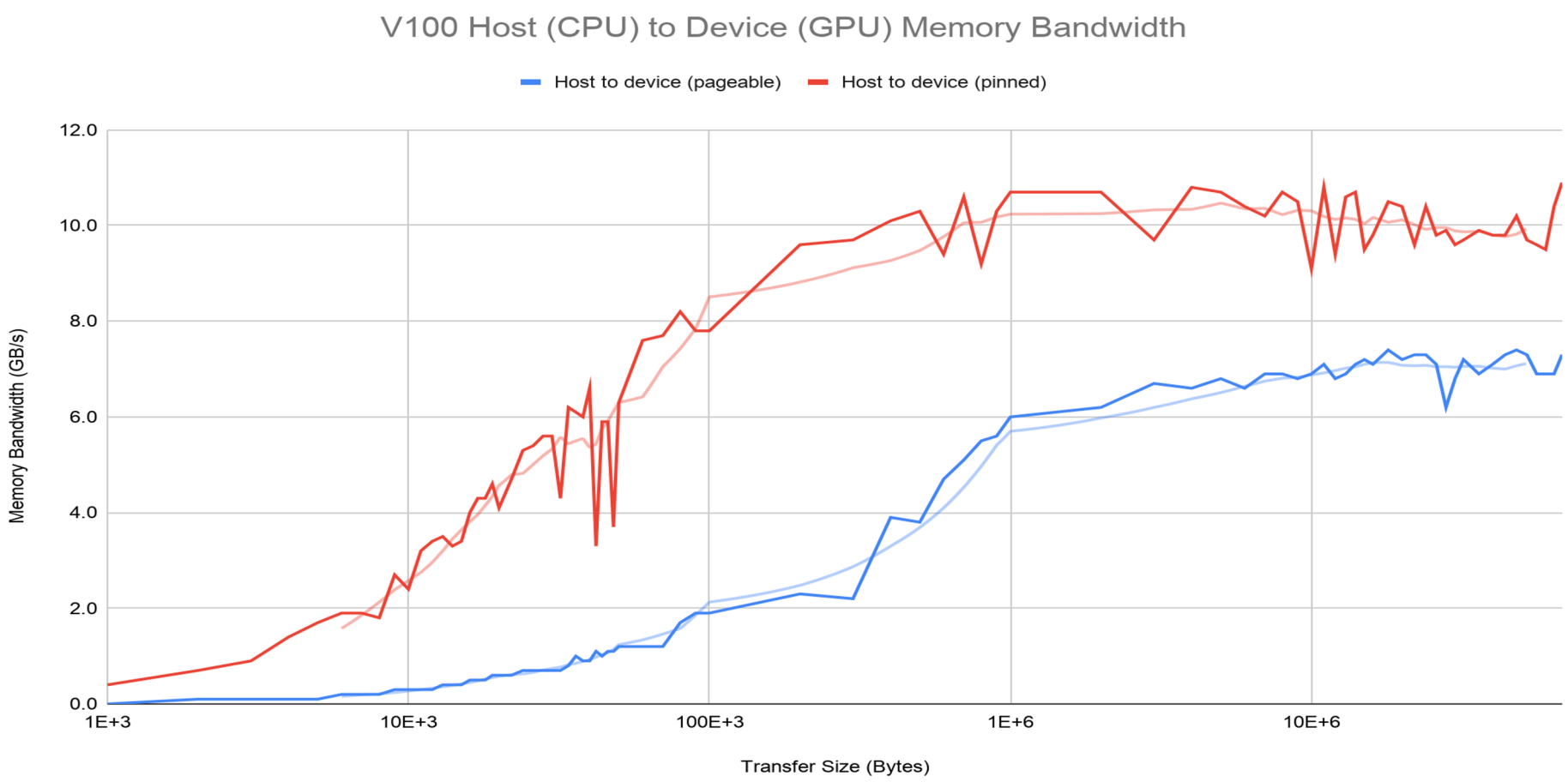
KEEP UP WITH THE GPU

Check for upstream bottlenecks
and fix them first



15

Transfer Data in Big Chunks



16

WHERE TO FOCUS (FLASHBACK!)

Measure instead of guessing
But no need to be perfect

NVIDIA / cuda-samples

Watch 62 | Star 912 | Fork

Code | Issues 7 | Pull requests 6 | Actions | Projects | Wiki | Security

master | cuda-samples / Samples / bandwidthTest / | Go to file | Add fil

mdoijade Add and update samples for cuda 11.1 support — cd76533 on 15 Sep — Hist

..		
Makefile	Add and update samples for cuda 11.1 support	2 months a
NsightEclipse.xml	Add and update samples for cuda 11.1 support	2 months a
README.md	Add and update samples for cuda 11.1 support	2 months a
bandwidthTest.cu	Add and update samples for cuda 11.0 support	6 months a
bandwidthTest_vs2015.sln	Add and update samples with CUDA 10.1 support	2 years a
bandwidthTest_vs2015.vcxproj	Add and update samples for cuda 11.1 support	2 months a
bandwidthTest_vs2017.sln	Add and update samples with CUDA 10.1 support	2 years a
bandwidthTest_vs2017.vcxproj	Add and update samples for cuda 11.1 support	2 months a
bandwidthTest_vs2019.sln	Add and update samples with CUDA 10.1 Update 1 support	2 years a
bandwidthTest_vs2019.vcxproj	Add and update samples for cuda 11.1 support	2 months a

README.md

bandwidthTest - Bandwidth Test

Description

This is a simple test program to measure the memcopy bandwidth of the GPU and memcpy bandwidth across PCI-e. This test application is capable of measuring device to device copy bandwidth, host to device copy bandwidth for pageable and page-locked memory, and device to host copy bandwidth for pageable and page-locked memory.

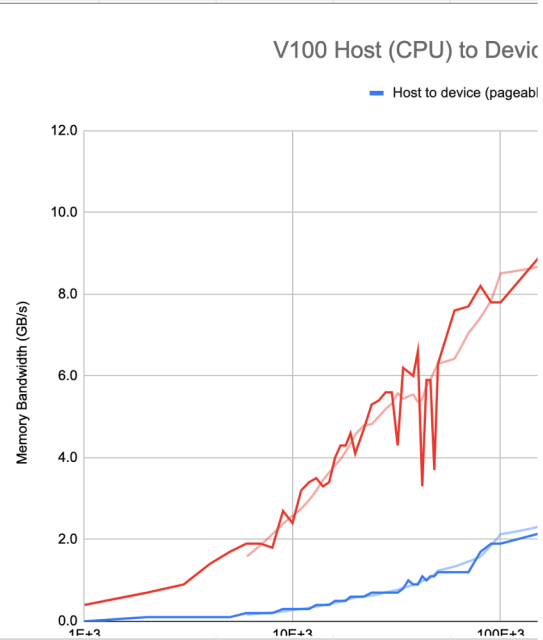
CUDA Memory Transfer Speed

File Edit View Insert Format Data Tools Add-ons Help — Last edit was on September 2

0.1

	A	B	C
1	Transfer Size (Bytes)	Memory Bandwidth (GB/s)	
2		Host to device (pageable)	Host to device (pinned)
3	1E+3	0.0	0.4
4	2E+3	0.1	0.7
5	3E+3	0.1	0.9
6	4E+3	0.1	1.4
7	5E+3	0.1	1.7
8	6E+3	0.2	1.9
9	7E+3	0.2	1.9
10	8E+3	0.2	1.8
11	9E+3	0.3	2.7
12	10E+3	0.3	2.4
13	11E+3	0.3	3.2
14	12E+3	0.3	3.4
15	13E+3	0.4	3.5
16	14E+3	0.4	3.3
17	15E+3	0.4	3.4
18	16E+3	0.5	4.0
19	17E+3	0.5	4.3
20	18E+3	0.5	4.3
21	19E+3	0.6	4.6
22	20E+3	0.6	4.1
23	22E+3	0.6	4.7
24	24E+3	0.7	5.3
25	26E+3	0.7	5.4
26	28E+3	0.7	5.6
27	30E+3	0.7	5.6
28	32E+3	0.7	4.3
29	34E+3	0.8	6.2
30	36E+3	1.0	6.1
31	38E+3	0.9	6.0

Note: tests run using "*bandwidthTest*" CUDA sample program. Noisy measureme
Also, device to host memory bandwidth follows similar curves.



Tesla V100-SXM2-32GB

Fast Code for
Inner Loops

Vectorize!

18

Vectorize your Python Code

pyinstrument

RECORDED: 11/23/2020, 4:26:57 PM DURATION: 58.4 SECONDS
SAMPLES: 49103 CPU TIME: 74.5 SECONDS

```
58.375 <module>                                                                                  main.py:17
  57.325 Fire                                                                                    fire/core.py:78
    ▸ 3 frames hidden (fire)
    57.317 run                                                                                   common/experiment.py:93
      57.317 gin_wrapper                                                                         gin/config.py:948
        ▸ 1 frames hidden (gin)
        57.311 instance                                                                          main.py:49
          56.380 gin_wrapper                                                                     gin/config.py:948
            ▸ 1 frames hidden (gin)
            56.048 trainer                                                                       ../trainer.py:31
              31.716 __iter__                                                                    common/sampler.py:45
                11.769 [self]                                                                    :45
                11.389 prod                                                                      <__array_function__ internals>:2
                  ▸ 8 frames hidden (<__array_function__ internals>, numpy, <built-in>...)
                6.186 RandomState.randint                                                        <built-in>:0
                  ▸ 1 frames hidden (<built-in>)
                1.063 max                                                                        <built-in>:0
                  ▸ 1 frames hidden (<built-in>)
                0.589 min                                                                        <built-in>:0
                  ▸ 1 frames hidden (<built-in>)
              6.598 decorate_context                                                             torch/autograd/grad_mode.py:23
                ▸ 8 frames hidden (torch, <built-in>, )
              4.270 clip_grad_norm_                                                              torch/nn/utils/clip_grad.py:9
                ▸ 4 frames hidden (torch, <built-in>)
              3.653 backward                                                                     torch/tensor.py:181
                ▸ 3 frames hidden (torch, <built-in>)
              3.291 to                                                                           torch/nn/modules/module.py:529
                ▸ 8 frames hidden (torch, <built-in>)
```


19

Make the GPU Busier

Parallelism & Pipelining

20

More Data
More Parallelism!

Make the GPU Busier

Overlap Data Transfers with Processing

1 How?

```
from torch.utils import data
from torch import cuda

dataloader = data.DataLoader(dataset,
   batch_size=1024, pin_memory=True, num_workers=8)

while not_done:
   for X, y in dataloader:
      X = X.cuda(non_blocking=True)
      y = y.cuda(non_blocking=True)

      # CPU work here will overlap with
      # memory transfers!

      # GPU work here will wait until
      # preceding CUDA operations finish.
```

2 What

- Good for basic parallelism between CPU and GPU
- More powerful when combined with streams!

Overlap Processing: CUDA streams

1 How?

```
from torch import cuda

# Define streams using capital-S Stream()
stream1 = cuda.Stream()
stream2 = cuda.Stream()

# Common work here...
# ... wait for that work to finish.
cuda.synchronize()

# Different streams, running in parallel
with cuda.stream(s1):
   # ...
with cuda.stream(s2):
   # ...

# Wait for all work to finish.
cuda.synchronize()
```

2 Rules

- **Default stream synchronizes with all streams!***
- **For other streams, calls are:**
  - Ordered within a stream
  - Unordered with other streams

For reference documentation, see: https://pytorch.org/docs/stable/cuda.html#streams-and-events

* Except for streams with non-blocking flag set.

For Further Studies

Other Tools in the Toolbox

- **Better PyTorch**
  - Improved PyTorch profiler in > 1.8.1
  - PyTorch-Lightning (easy multi-gpu & distributed training, same with float16!)
  - TorchScript
- **Do more on the GPU**
  - RAPIDS.AI: CuDF, CuML, etc.
  - DALI
  - CUVI
  - CuPy (to port NumPy code)
  - … and more: search for NVIDIA CUDA-X
- **Faster Python**
  - @numba.jit
  - @numba.cuda.jit

25

Thank you.

For a copy of the slides, etc.
http://christianhudon.name/talks#mp-gpu-profiling

We're hiring! servicenow.com/careers

now