NCCL / NVIDIA Nsight Systems / NVIDIA Nsight Compute

2023 NSF-sponsored Workshop on Deep Learning Systems in Advanced GPU Cyberinfrastructure

Dr. Iraklis Anagnostopoulos

Southern Illinois University

Table of contents

1. Overview
2. NCCL
3. NVIDIA Nsight Systems
4. NVIDIA Nsight Compute

Overview

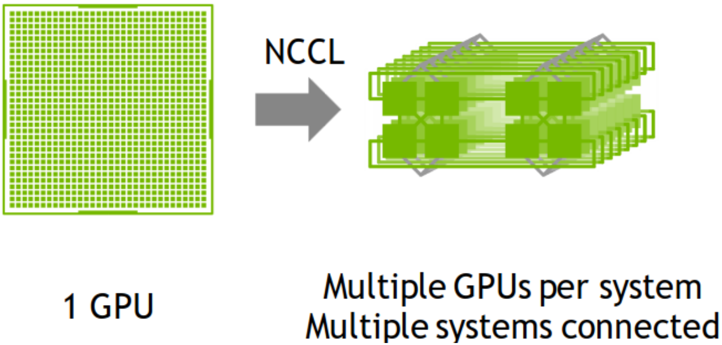
Overview

- Introduction
  - Brief introduction to NCCL, Nsight Systems, and Nsight Compute
- NCCL
  - Overview of NCCL: What it is and why it's important
  - Basic operations
  - Hands-on exercise: Implementing simple NCCL operations
- Nsight Systems
  - Overview of Nsight Systems: What it is and why it's important
  - Basic profiling
  - Hands-on exercise: Profile convolution kernels
- Nsight Compute
  - Overview of Nsight Compute: What it is and why it's important
  - Basic profiling
  - Hands-on exercise: Profile matrix multiplication kernels

NCCL

NVIDIA NCCL

- Harvesting the power of multiple GPUs
- NCCL: **N**VIDIA **C**ollective **C**ommunication **L**ibrary
  - a library developed by NVIDIA that supports high-performance multi-GPU computing.

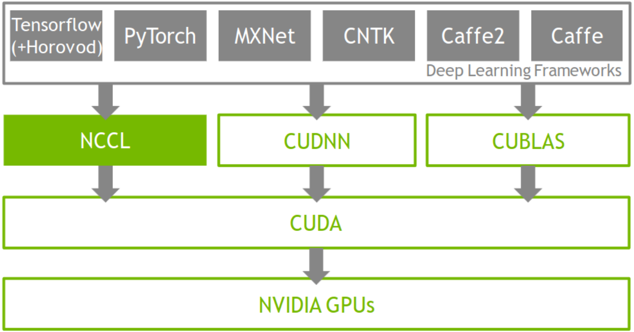


Why is NCCL important?

- In multi-GPU and multi-node computations, communication between different GPUs is a significant bottleneck.
- The data transfer between GPUs, especially when they are not on the same node, can be slow and inefficient, leading to reduced performance.
- NCCL aims to solve this problem by providing optimized and efficient collective communication routines
  - handle data transfer between GPUs
  - improving the overall performance of multi-GPU computations.

How NCCL Aids in Multi-GPU Programming

- NCCL provides high-performance implementations of send, receive, and reduction operations
- It supports a variety of reduction and broadcast operations
- NCCL is designed to work with a variety of GPU architectures and network interconnects



NCCL insights

Goal:

- Build a research library of accelerated collectives that is easily integrated and topology-aware so as to improve the scalability of multi-GPU applications

Approach:

- Pattern the library after MPI's collectives $\Rightarrow$ Not the focus of the workshop
- Handle the intra-node communication in an optimal way
- Provide the necessary functionality for MPI to build on top to handle inter-node

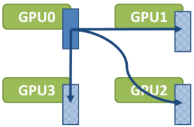
Specific Features of NCCL

- NCCL supports a variety of communication patterns
  - point-to-point
  - broadcast
  - reduce
  - all-reduce
  - reduce-scatter
  - all-gather
- Full documentation of NCCL here

Collective communication

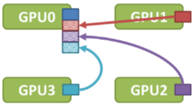
Multiple senders and/or receivers

Broadcast

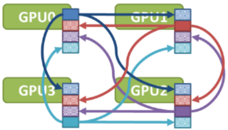


Scatter

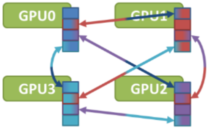
Gather



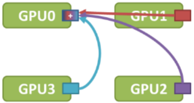
All-Gather



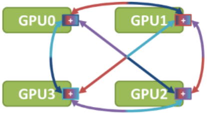
All-to-All



Reduce

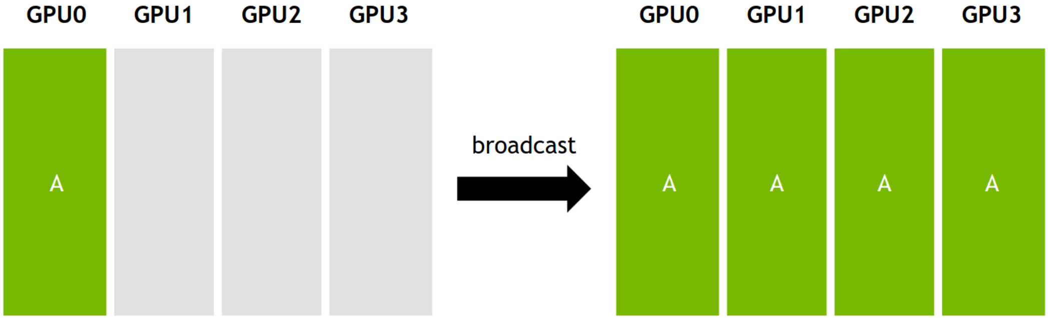


All-Reduce



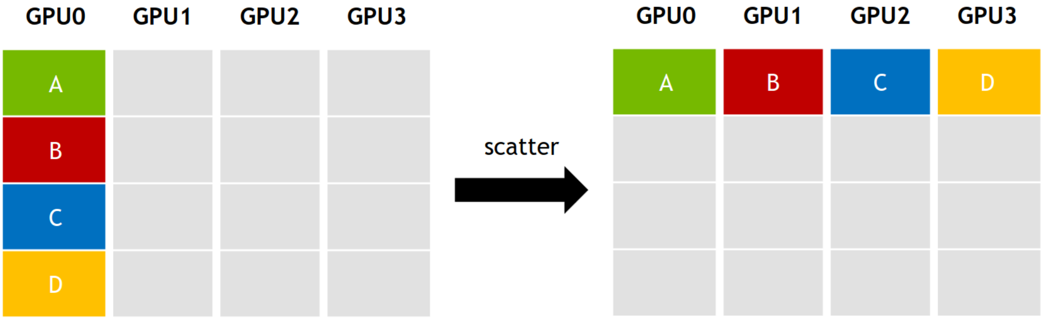
Broadcast

One sender, multiple receivers



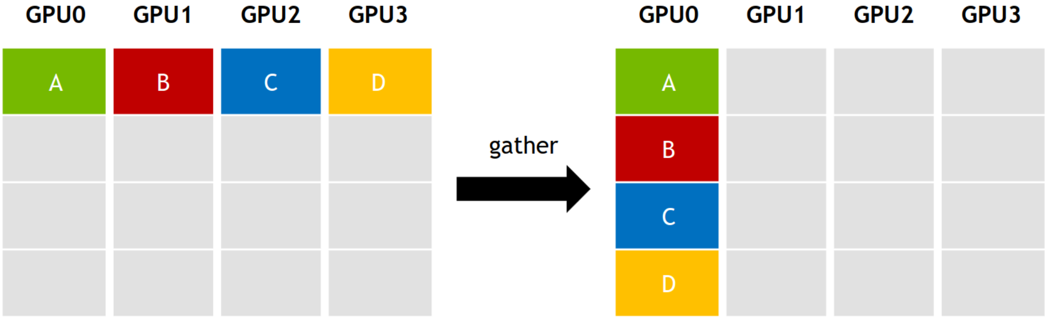
Scatter

One sender; data is distributed among multiple receivers



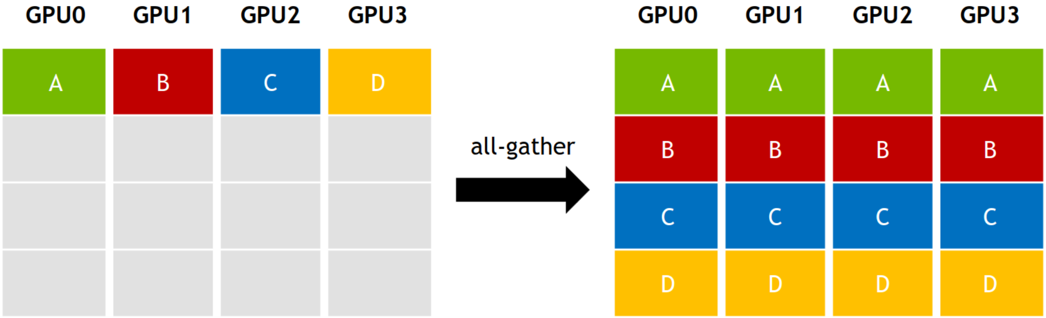
Gather

Multiple senders, one receiver



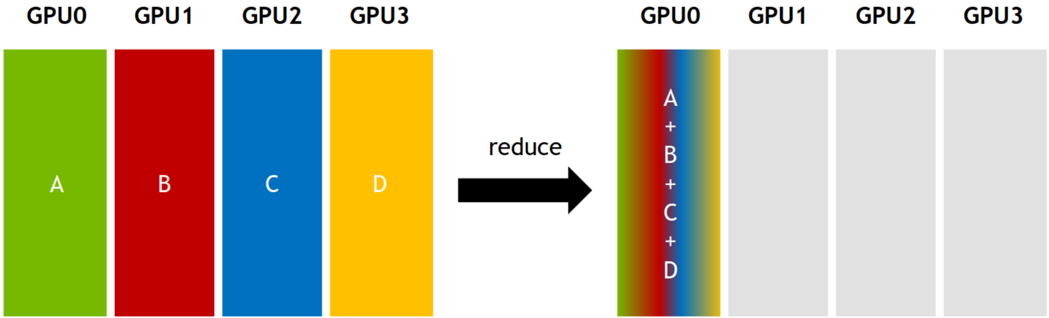
All Gather

Gather messages from all; deliver gathered data to all participants



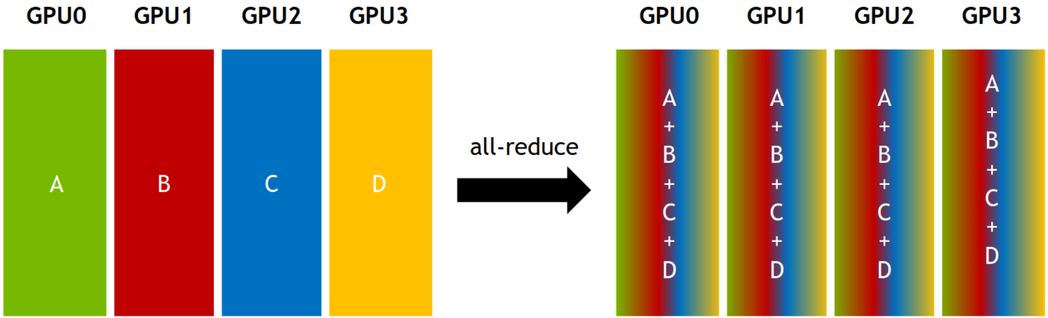
Reduce

Combine data from all senders; deliver the result to one receiver



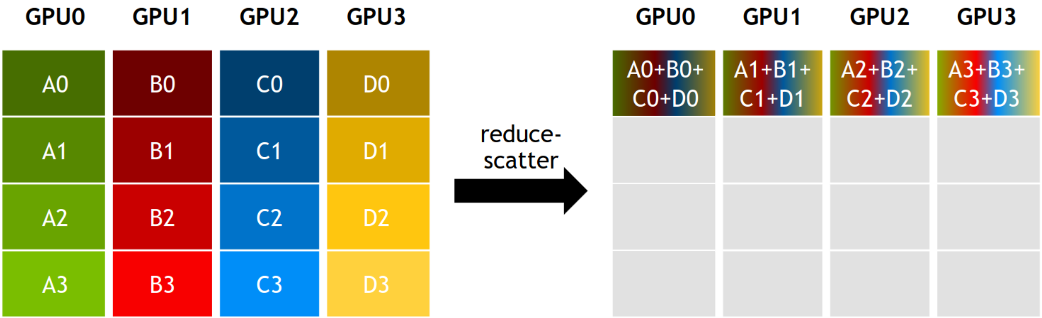
All Reduce

Combine data from all senders; deliver the result to all participants



Reduce-Scatter

Combine data from all senders; distribute result across participants



The challenge of collectives

- If collectives are so expensive, do they actually get used? YES!
- Deep Learning (All-reduce, broadcast, gather)
- Parallel FFT (Transposition is all-to-all)
- Molecular Dynamics (All-reduce)
- Graph Analytics (All-to-all)

Broadcast

ncclBcast prototype

```
int ncclBcast (
    void*          data_p      /* in/out*/,
    size_t         count       /*   in   */,
    ncclDataType_t datatype    /*   in   */,
    int            source_proc /*   in   */,
    ncclComm_t     comm        /*   in   */,
    cudaStream_t   stream      /*   in   */
);
```

MPI_Bcast prototype

```
int MPI_Bcast (
    void*          data_p      /* in/out*/,
    int            count       /*   in   */,
    MPI_Datatype   datatype    /*   in   */,
    int            source_proc /*   in   */,
    MPI_Comm       comm        /*   in   */
);
```

- The only difference is the the last parameter `stream`
    - It represents the sending format between the GPUs

ncclBcast

- Check the provided `ncclBcast.cu` code
- It implements a sample BROADCAST code using the package NCCL

Compilation and execution

```
$ nvcc ncclBcast.cu -o ncclBcast -lnccl
$ ./ncclBcast
```

- Identify in the code the following components:
  - Listing of GPUs (number)
  - NCCL initialization
  - Copy data from host to GPU
  - NCCL broadcast operations
  - The kernel each GPU executes
  - Explain the final outcome

Activity

- Download the code `ncclReduce.cu`
- It implements dot product (scalar product) using `ncclReduce`
- Perform the following:
    - Identify the NCCL operation
    - Modify the code so as each GPU gets the final output
    - Print the output per GPU and verify its correctness
    - Modify the code so as each input vector is initialized with random values from 0 to 10

NVIDIA Nsight Systems

NVIDIA Nsight Systems

- A system-wide performance analysis tool
- It provides a high-level overview of how an application is using the system's resources
  - CPU, and
  - GPU
- It helps to identify bottlenecks in our applications
  - areas where the CPU or GPU is being underutilized
  - excessive synchronization between the CPU and GPU
  - etc.
- It's particularly useful for understanding the "big picture"

NVIDIA Nsight Systems

- NVIDIA Nsight Systems command line tool is called nsys

```
$ nsys --version
NVIDIA Nsight Systems version 2022.1.3.3-1c7b5f7
$ nsys --help
The most commonly used nsys commands are:
        profile       Run an application and capture its profile into a QDSTRM file.
        launch        Launch an application ready to be profiled.
        start         Start a profiling session.
        stop          Stop a profiling session and capture its profile into a QDSTRM file.
        cancel        Cancel a profiling session and discard any collected data.
        stats         Generate statistics from an existing nsys-rep or SQLite file.
        status        Provide current status of CLI or the collection environment.
        shutdown      Disconnect launched processes from the profiler and shutdown the profiler.
        sessions list List active sessions.
        export        Export nsys-rep file into another format.
        analyze       Run rules on an existing nsys-rep or SQLITE file.
        nvprof        Translate nvprof switches to nsys switches and execute collection.
```

NVIDIA Nsight Systems

- Official link to NVIDIA Nsight Systems here
- The complete list of features can be found here
- The Nsight Systems CLI provides a simple interface to collect on a target without using the GUI
- The collected data can then be copied to any system and analyzed later
- General command format:

```
$ nsys [command_switch][optional command_switch_options][application] [optional application_options]
```

Example: Adding two arrays

- Check the provided code `addarrays.cu`
- It adds the elements of two arrays on the GPU

Compilation and execution

```
$ nvcc -o addarrays addarrays.cu
$ nsys profile --stats=true --force-overwrite true -o report ./addarrays
```

Example: Adding two arrays

- Check the provided code `addarrays.cu`
- It adds the elements of two arrays on the GPU

Compilation and execution

```
$ nvcc -o addarrays addarrays.cu
$ nsys profile --stats=true --force-overwrite true -o report ./addarrays
```

- `nsys profile`: Start a profiling session with Nsight Systems.
- `--stats=true`: Print a summary of the profiling results to the console
- `--force-overwrite true`: Overwrite any existing profiling data with the same name
- `-o report`: Save the profiling data to a file named `report.qdrep`

Example: Adding two arrays - Output for operating sytem

Operating System Runtime API Statistics:

Time (%)	Total Time (ns)	Num Calls	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
73.9	1,113,804,107	3	371,268,035.7	393,747,147.0	1,061,513	718,995,447	359,494,459.3	sem_wait
18.0	270,611,330	19	14,242,701.6	10,069,455.0	2,558	100,129,004	24,284,703.5	poll
4.7	70,166,271	504	139,218.8	11,618.5	1,065	25,241,326	1,357,498.6	ioctl
2.8	42,662,842	15	2,844,189.5	82,218.0	26,742	20,534,983	5,898,506.4	sem_timed
0.3	5,215,754	42	124,184.6	20,918.5	9,801	2,409,020	439,191.3	mmap64
0.1	1,340,510	8	167,563.8	7,612.0	2,082	701,533	300,156.7	fread
0.1	906,253	22	41,193.3	14,498.0	2,473	371,659	83,668.8	mmap
0.0	621,904	59	10,540.7	9,399.0	3,922	20,438	3,690.5	open64
0.0	406,006	5	81,201.2	76,247.0	66,261	107,800	16,075.5	pthread_c
0.0	342,166	54	6,336.4	5,529.0	2,249	21,263	3,412.2	fopen
0.0	135,187	7	19,312.4	10,124.0	7,211	67,683	21,717.4	fgets
0.0	103,903	44	2,361.4	2,235.0	1,271	4,655	667.5	fclose
0.0	65,263	6	10,877.2	8,978.0	6,505	20,603	5,074.5	munmap
0.0	35,225	23	1,531.5	1,248.0	1,010	4,013	805.5	fcntl
0.0	34,385	5	6,877.0	6,067.0	5,440	10,062	1,928.2	open
0.0	22,066	11	2,006.0	1,814.0	1,181	3,415	798.2	read

...

Example: Adding two arrays - Output for CUDA

CUDA API Statistics:

Time (%)	Total Time (ns)	Num Calls	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
97.5	147,204,541	2	73,602,270.5	73,602,270.5	59,116	147,145,425	104,005,726.5	cudaMallocManaged
2.0	3,023,309	1	3,023,309.0	3,023,309.0	3,023,309	3,023,309	0.0	cudaDeviceSynchronize
0.5	708,519	2	354,259.5	354,259.5	260,222	448,297	132,989.1	cudaFree
0.0	41,804	1	41,804.0	41,804.0	41,804	41,804	0.0	cudaLaunchKernel
0.0	1,732	1	1,732.0	1,732.0	1,732	1,732	0.0	cuModuleGetLoadingMode

...

CUDA Kernel Statistics:

Time (%)	Total Time (ns)	Instances	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
100.0	2,979,577	1	2,979,577.0	2,979,577.0	2,979,577	2,979,577	0.0	add(int, float *, float *)

...

CUDA Memory Operation Statistics (by time):

Time (%)	Total Time (ns)	Count	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Operation
64.6	755,192	48	15,733.2	3,967.5	2,558	85,630	24,110.7	[CUDA Unified Memory memcpy HtoD]
35.4	413,676	24	17,236.5	3,855.5	1,183	98,590	27,704.2	[CUDA Unified Memory memcpy DtoH]

Example: Adding two arrays

- Did you notice the message at the top?

```
[3/8] Executing 'nvtxsum' stats report
SKIPPED: home/.../report.sqlite does not contain NV Tools Extension (NVTX) data.
```

Example: Adding two arrays

- Did you notice the message at the top?

```
[3/8] Executing 'nvtxsum' stats report
SKIPPED: home/.../report.sqlite does not contain NV Tools Extension (NVTX) data.
```

- NVTX is a C-based API for annotating events, code ranges, and resources
- It allows developers to manually instrument their application to provide additional insights when profiling.

Example: Adding two arrays

- Did you notice the message at the top?

```
[3/8] Executing 'nvtxsum' stats report
SKIPPED: home/.../report.sqlite does not contain NV Tools Extension (NVTX) data.
```

- NVTX is a C-based API for annotating events, code ranges, and resources
- It allows developers to manually instrument their application to provide additional insights when profiling.
- For example, you can use NVTX to mark the start and end of a particular section of code
- You'll be able to see exactly when that section of code was executing.

Example: Adding two arrays

- Check the provided code `addarrayNVTX.cu`
- I've added a call to `nvtxRangePushA` before the CUDA kernel is launched
- And a call to `nvtxRangePop` after the kernel has finished executing

Compilation and execution

```
$ nvcc -o addarraysNVTX addarraysNVTX.cu -lnvToolsExt
$ nsys profile --stats=true --force-overwrite true -o report ./addarraysNVTX
```

Example: Adding two arrays

- Check the provided code `addarrayNVTX.cu`
- I've added a call to `nvtxRangePushA` before the CUDA kernel is launched
- And a call to `nvtxRangePop` after the kernel has finished executing

Compilation and execution

```
$ nvcc -o addarraysNVTX addarraysNVTX.cu -lnvToolsExt
$ nsys profile --stats=true --force-overwrite true -o report ./addarraysNVTX
```

- Check the new addition to the output

```
[3/8] Executing 'nvtxsum' stats report
NVTX Range Statistics:
Time (%)  Total Time (ns)  Instances   Avg (ns)     Med (ns)     Min (ns)   Max (ns)   StdDev (ns)   Style           Range
--------  ---------------  ---------  -----------  -----------  ---------  ---------  -----------  -------  ---------------------
100.0         3,978,097          1  3,978,097.0  3,978,097.0  3,978,097  3,978,097          0.0  PushPop  CUDA Kernel Execution
```

Activity

- When we discussed talking about cuDNN, we used the `conv1d.cu`
- Use the NVTX API to monitor the activity of the convolution using cuDNN
- Use the NVTX API to monitor the activity of the convolution using CUDA
- Profile the code using NVIDIA Nsight Systems and report the findings

NVIDIA Nsight Compute

NVIDIA Nsight Compute

- A detailed, low-level profiling tool specifically for CUDA applications
- Provides a deep dive into the GPU's performance
  - memory usage,
  - instruction throughput,
  - and more
- It helps us understand exactly how our CUDA kernels are executing on the GPU

NVIDIA Nsight Compute

- A detailed, low-level profiling tool specifically for CUDA applications
- Provides a deep dive into the GPU's performance
  - memory usage,
  - instruction throughput,
  - and more
- It helps us understand exactly how our CUDA kernels are executing on the GPU
- Nsight Systems $\Rightarrow$ high-level overview
- Nsight Compute $\Rightarrow$ detailed analysis

NVIDIA Nsight Compute

- NVIDIA Nsight Compute command line tool is called `ncu`

```
$ ncu --version
NVIDIA (R) Compute Command Line Profiler
Copyright (c) 2012-2019 NVIDIA Corporation
Version 2019.1 (Build 25595643)
$ ncu --help
General Options:
  -h [ --help ]                       Print this help message.
  -v [ --version ]                    Print the version number.
  --mode arg (=launch-and-attach)     Select the mode of interaction
                                      with the target application
  ...
```

NVIDIA Nsight Compute

- Official link to NVIDIA Nsight Compute here
- The complete list of features can be found here
- The Nsight Compute CLI provides a simple interface to collect on a target without using the GUI
- The collected data can then be copied to any system and analyzed later
- General command format:

```
$ ncu [options] [program] [program-arguments]
```

Example: Adding two arrays

- Check the provided code `addarrays.cu`
- It adds the elements of two arrays on the GPU

Compilation and execution

```
$ nvcc -o addarrays addarrays.cu
$ ncu ./addarrays
```

Example: Adding two arrays

- Check the provided code `addarrays.cu`
- It adds the elements of two arrays on the GPU

Compilation and execution

```
$ nvcc -o addarrays addarrays.cu
$ ncu ./addarrays
```

- After running this command, Nsight Compute will profile your application and print a summary of the profiling results to the console
- You can also use the `-o` option to save the profiling data to a file

```
$ ncu -o report ./addarrays
```

Example: Adding two arrays - Other useful commands

- Tells Nsight Compute to print a summary of the profiling results for each kernel:

```
$ ncu --summary per-kernel ./addarrays
```

Example: Adding two arrays - Other useful commands

- Tells Nsight Compute to print a summary of the profiling results for each kernel:

```
$ ncu --summary per-kernel ./addarrays
```

- If you're interested in the number of instructions executed and the memory bandwidth used:

```
$ ncu --metrics inst_executed,mem_bandwidth ./addarrays
```

Example: Adding two arrays - Other useful commands

- Tells Nsight Compute to print a summary of the profiling results for each kernel:

```
$ ncu --summary per-kernel ./addarrays
```

- If you're interested in the number of instructions executed and the memory bandwidth used:

```
$ ncu --metrics inst_executed,mem_bandwidth ./addarrays
```

- Collect and display a specific set of metrics:

```
$ ncu --metrics smsp__inst_executed.avg.pct_of_peak_sustained_active,\
smsp__cycles_elapsed.avg.per_second ./addarrays
```

  - average percentage of peak sustained active instructions executed
  - average number of cycles elapsed per second

Example: Adding two arrays - More useful commands

- Collect and display all metrics:

```
$ ncu --metrics all ./addarrays
```

Example: Adding two arrays - More useful commands

- Collect and display all metrics:

```
$ ncu --metrics all ./addarrays
```

- Collect and display a specific section of the report:

```
$ ncu --sections SpeedOfLight  ./addarrays
```

  - "SpeedOfLight" section includes metrics related to the theoretical and achieved performance of the GPU.

Example: Adding two arrays - More useful commands

- Collect and display all metrics:

```
$ ncu --metrics all ./addarrays
```

- Collect and display a specific section of the report:

```
$ ncu --sections SpeedOfLight  ./addarrays
```

  - “SpeedOfLight” section includes metrics related to the theoretical and achieved performance of the GPU.

- Profile a specific kernel:

```
$ ncu --kernel-id ::add: ./addarrays
```

  - average percentage of peak sustained active instructions executed
  - average number of cycles elapsed per second

Activity

- When we discussed talking about cuBLAS, we used the `matmul.cu`
- Use the Nsights Compute to monitor the activity of the program
- Use the Nsights Compute to monitor the activity of the CUDA kernel

Questions?