



# CUDA TOOLS

## TOOLS FOR PROFILING AND DEBUGGING

April 2021 | Andreas Herten / Markus Hrywniak | Forschungszentrum Jülich / NVIDIA

# Outline

## Goals of this session

- Use **compute-sanitizer** to detect invalid memory accesses
- Use **cuda-gdb** to debug a CUDA program
- Gain performance insight with **NVIDIA Nsight Systems/Compute**, **nvprof**

## Contents

### Debugging

compute-sanitizer

Task 1

cuda-gdb

IDE integration

Task 2

Profiling

Nsight Suite

Others

Task 3

# Debugging

# compute-sanitizer

## Command-line functional correctness checking suite

- Compute Sanitizer is the successor of cuda-memcheck in CUDA 11
- Default: Memory error detector; similar to Valgrind's memcheck

# compute-sanitizer

## Command-line functional correctness checking suite

- Compute Sanitizer is the successor of cuda-memcheck in CUDA 11
- Default: Memory error detector; similar to [Valgrind's memcheck](#)
- Has sub-tools, via `compute-sanitizer --tool NAME:`
  - memcheck: Memory access checking (*default*)
  - racecheck: Shared memory hazard checking
  - Also: synccheck, initcheck



# compute-sanitizer

## Command-line functional correctness checking suite

- Compute Sanitizer is the successor of cuda-memcheck in CUDA 11
- Default: Memory error detector; similar to Valgrind's memcheck
- Has sub-tools, via `compute-sanitizer --tool NAME:`
  - memcheck: Memory access checking (*default*)
  - racecheck: Shared memory hazard checking
  - Also: synccheck, initcheck
- Remember to compile your program with line (or debug) information: add `-g` (host) or `-lineinfo` (device).



# compute-sanitizer

## Command-line functional correctness checking suite

- Compute Sanitizer is the successor of cuda-memcheck in CUDA 11
- Default: Memory error detector; similar to [Valgrind's memcheck](#)
- Has sub-tools, via `compute-sanitizer --tool NAME:`
  - `memcheck`: Memory access checking (*default*)
  - `racecheck`: Shared memory hazard checking
  - Also: `synccheck`, `initcheck`
- Remember to compile your program with line (or debug) information: add `-g` (host) or `-lineinfo` (device).

### Compile options for nvcc

<code>-g</code>	Debug info for <b>host</b> code
<code>-G</code>	Debug info for <b>device</b> code
<code>-lineinfo</code>	Line number for device code

# compute-sanitizer

## Command-line functional correctness checking suite

- Compute Sanitizer is the successor of cuda-memcheck in CUDA 11
- Default: Memory error detector; similar to Valgrind's memcheck
- Has sub-tools, via `compute-sanitizer --tool NAME:`
  - memcheck: Memory access checking (*default*)
  - racecheck: Shared memory hazard checking
  - Also: synccheck, initcheck
- Remember to compile your program with line (or debug) information: add `-g` (host) or `-lineinfo` (device).

### Compile options for nvcc

- `-g` Debug info for **host** code *ok*
- `-G` Debug info for **device** code *slow*
- `-lineinfo` Line number for device code *ok*





# compute-sanitizer

## Command-line functional correctness checking suite

- Compute Sanitizer is the successor of cuda-memcheck in CUDA 11
- Default: Memory error detector; similar to Valgrind's memcheck
- Has sub-tools, via compute-sanitizer --tool NAME:
  - memcheck: Memory access checking (default)
  - racecheck: Shared memory hazard checking
  - Also: synccheck, initcheck
- Remember to compile your program with line (or debug) information: add -g (host) or -lineinfo (device).

→ <https://docs.nvidia.com/cuda/sanitizer-docs/ComputeSanitizer>

### Compile options for nvcc

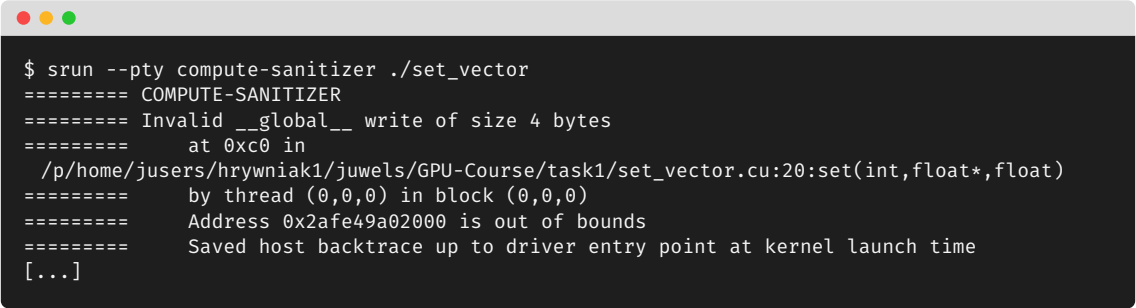
- g Debug info for **host** code *ok*
- G Debug info for **device** code *slow*
- lineinfo Line number for device code *ok*

# Example

**Launch:** compute-sanitizer PROGRAM

# Example

**Launch:** `compute-sanitizer PROGRAM`



```
$ srun --pty compute-sanitizer ./set_vector
===== COMPUTE-SANITIZER
===== Invalid __global__ write of size 4 bytes
=====      at 0xc0 in
   /p/home/jusers/hrywniak1/juwels/GPU-Course/task1/set_vector.cu:20:set(int,float*,float)
=====      by thread (0,0,0) in block (0,0,0)
=====      Address 0x2afe49a02000 is out of bounds
=====      Saved host backtrace up to driver entry point at kernel launch time
[...]
```

# Task 1

## TASK 1

### Use compute-sanitizer to identify error

- Location of code: 2-Tools/exercises/tasks/task1
- Steps (see also Instructions.ipynb)
  - Fix `set-vector.cu`  
Use compute-sanitizer to fix error in `set-vector.cu`  
compute-sanitizer should run without errors!
  - Build: `make`
  - Run: `make run/make memcheck`

# cuda-gdb

## Symbolic debugger

- Powerful symbolic debugger for CUDA code
- Built on top of `gdb`
- Full usage: own course needed

# cuda-gdb

## Symbolic debugger

- Powerful symbolic debugger for CUDA code
- Built on top of `gdb`
- Full usage: own course needed

### cuda-gdb 101

```
run Starts application, give arguments with set
    args 1 2 ...

break L Create breakpoint
    L: function name, line LN, or FILE:LN

print i Print content of i

set variable i = 10 Set i to 10

info locals Print all currently set variables

info cuda threads Print current thread configuration

cuda thread N Switch context to thread number N

set cuda api_failures stop Break execution on CUDA
    errors

continue Continue running
```

→ [cheat sheet](#)



# cuda-gdb

## Symbolic debugger

- Powerful symbolic debugger for CUDA code
- Built on top of `gdb`
- Full usage: own course needed

→ <https://docs.nvidia.com/cuda/cuda-gdb/>

### cuda-gdb 101

`run` Starts application, give arguments with set args 1 2 ...

`break L` Create breakpoint  
*L: function name, line LN, or FILE:LN*

`print i` Print content of i

`set variable i = 10` Set i to 10

`info locals` Print all currently set variables

`info cuda threads` Print current thread configuration

`cuda thread N` Switch context to thread number N

`set cuda api_failures stop` Break execution on CUDA errors

`continue` Continue running

→ [cheat sheet](#)

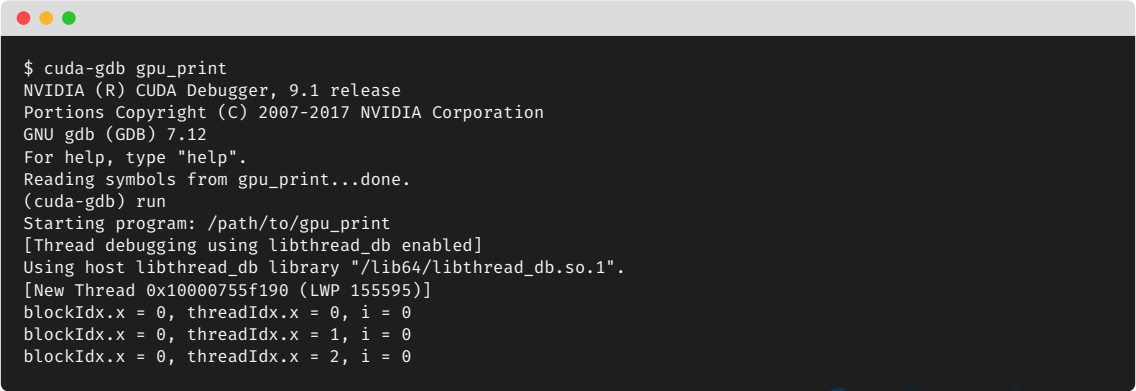


# cuda-gdb

## Example

**Launch:** `cuda-gdb app → run`

Set breakpoint with `break func` or `break L` or `break file.c:L`

A terminal window with a dark background and a light gray title bar containing three colored window control buttons (red, yellow, green). The terminal displays the output of the 'cuda-gdb' command, showing version information for NVIDIA CUDA Debugger, GNU gdb, and the execution of a program named 'gpu\_print'. It also shows thread debugging information and the start of a new thread.

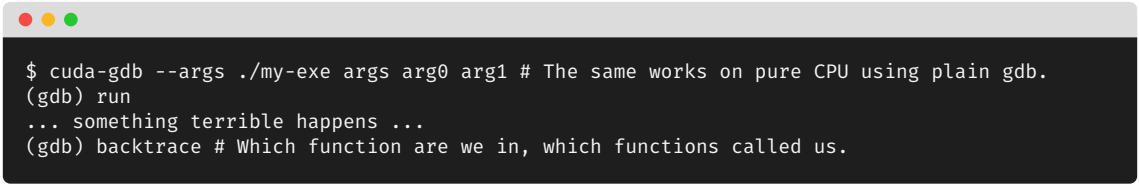
```
$ cuda-gdb gpu_print
NVIDIA (R) CUDA Debugger, 9.1 release
Portions Copyright (C) 2007-2017 NVIDIA Corporation
GNU gdb (GDB) 7.12
For help, type "help".
Reading symbols from gpu_print...done.
(cuda-gdb) run
Starting program: /path/to/gpu_print
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
[New Thread 0x10000755f190 (LWP 155595)]
blockIdx.x = 0, threadIdx.x = 0, i = 0
blockIdx.x = 0, threadIdx.x = 1, i = 0
blockIdx.x = 0, threadIdx.x = 2, i = 0
```



# cuda-gdb

## A typical debug session

This is *the one sequence* to remember, if nothing else.




```
$ cuda-gdb --args ./my-exe args arg0 arg1 # The same works on pure CPU using plain gdb.  
(gdb) run  
... something terrible happens ...  
(gdb) backtrace # Which function are we in, which functions called us.
```

# cuda-gdb

## A typical debug session

This is *the one sequence* to remember, if nothing else.



```
$ cuda-gdb --args ./my-exe args arg0 arg1 # The same works on pure CPU using plain gdb.  
(gdb) run  
... something terrible happens ...  
(gdb) backtrace # Which function are we in, which functions called us.
```

Think of gdb as a *shell* for debugging with some extensions:

```
(gdb) bt # same as above - can abbreviate every command as long as unambiguous  
(gdb) c # continue - see next slides  
(gdb) help b # get help on any topic, abbreviation or command - try it out!  
(gdb) apropos cuda.*stop # regex search through help topics
```

# Breakpoints

Interrupt execution when a certain (source) location is reached.

```
(gdb) break foo          # break at function/kernel/template  
(gdb) break file:line    # break at line  
(gdb) list file.cpp:5    # forgot where - show source code!
```

- Conditional breakpoints

```
(gdb) break foo if i == 42 && threadIdx.x == 23
```

# At The Breakpoint

When execution is halted the program state can be inspected.

- Getting information

```
(gdb) backtrace      # show stack of functions to here  
(gdb) list           # show source code context
```

# At The Breakpoint

When execution is halted the program state can be inspected.

- Getting information

```
(gdb) backtrace      # show stack of functions to here  
(gdb) list           # show source code context
```

- Showing variables and memory

```
(gdb) print bar      # print value of variable  
(gdb) print arr[0]@4 # print first 4 values in array
```

# At The Breakpoint

When execution is halted the program state can be inspected.

- Getting information

```
(gdb) backtrace      # show stack of functions to here
(gdb) list           # show source code context
```

- Showing variables and memory

```
(gdb) print bar      # print value of variable
(gdb) print arr[0]@4 # print first 4 values in array
```

- Also, the state can be *written*

```
(gdb) set variable bar = 42 # set a variable
(gdb) call foo(23)          # call function inside debuggee
```

# Leaving the Breakpoint

- Resuming execution

```
(gdb) continue # resume until next breakpoint (same if in loop)
(gdb) step      # resume until next line
(gdb) next      # same, but do not follow calls
(gdb)           # here: 'next' - repeats last command
```

# Leaving the Breakpoint

- Resuming execution

```
(gdb) continue # resume until next breakpoint (same if in loop)
(gdb) step      # resume until next line
(gdb) next      # same, but do not follow calls
(gdb)           # here: 'next' - repeats last command
```

- Deleting a breakpoint

```
(gdb) delete <id> # id is returned when setting a breakpoint
```



# GPU-specifics

## Information

So far, most techniques have been general. Now, we start delving into the specifics of `cuda-gdb`. *Note:* Most need `-g` to be useful.

Remember the GPU execution model of

- kernels, grids, blocks, threads (logically)
- devices, SMs, warps, lanes (physically).

You can retrieve detailed information about any of these when stopped on a kernel:

```
(cuda-gdb) info cuda kernels # what is currently running
Kernel Parent Dev Grid Status   SMs Mask  GridDim  BlockDim Invocation
*      0      -   0    3 Active 0xffffffff (57,1,1) (128,1,1) initialize_boundaries()
(cuda-gdb) i cuda threads
BlockIdx ThreadIdx To BlockIdx ThreadIdx Count      Virtual PC  Filename  Line
Kernel 0
*  (0,0,0)  (0,0,0)    (56,0,0) (127,0,0)  7296 0x000055555555da6eb0  jacobi.cu  93
```

# GPU-specifics

## Focus

cuda-gdb is working with a subset of threads called the ‘focus’.

```
(cuda-gdb) cuda device sm warp lane block thread # Show current focus  
block (0,0,0), thread (0,0,0), device 0, sm 0, warp 3, lane 0  
(cuda-gdb) # Change focus  
(cuda-gdb) cuda device <d> sm <s> warp <w> lane <l> block <b> thread <t>  
(cuda-gdb) # in both cases, you can leave out items
```

Working with the focus is extremely helpful when only some threads produce an error, e.g. out-of-bounds access.

# IDE integration

- Why use an integrated development environment (IDE)?
  - Source code editor with CUDA C / C++ highlighting
  - Project / file management with integration of version control
  - Build system
  - Graphical interface for debugging heterogeneous applications
- Plugins for the **Eclipse** platform. Recommended:  
<https://github.com/NVIDIA/nsight-training>  
→ <https://developer.nvidia.com/nsight-eclipse-edition/>
- On Windows: Nsight Visual Studio Edition  
→ <https://developer.nvidia.com/nsight-visual-studio-edition/>
- Up-and-coming: Nsight Visual Studio Code Edition  
→ <https://developer.nvidia.com/nsight-visual-studio-code-edition/>

Debug - findmax/src/findmax.cu - Cider

File Edit Source Refactor Navigate Search Run Project Window Help

Debug Variables CUDA Information Breakpoints

findmax [C/C++ Application]

- cudaFindMax [0] [device: 0] (Suspended : Step)
  - CUDA Thread (0,0,0) Block (0,0,0)
    - cudaFindMax() at findmax.cu:114 0x91f3a8
  - CUDA Thread (1,0,0) Block (0,0,0)
  - Block (0,0,0) [sm: 0] (256 Active Threads)
  - Block (1,0,0) [sm: 2] (256 Active Threads)

Variables: sm 2 warp 7

[0] cudaFindMax	Running	Device 0	<<<(32,1,1), (256,1,1)>>>
(1,0,0)	Running	SM 2	
(224,0,0)	Running	Warp 7 Lane 0	findmax.cu:113 (0x91f318)
(225,0,0)	Running	Warp 7 Lane 1	findmax.cu:113 (0x91f318)
(226,0,0)	Running	Warp 7 Lane 2	findmax.cu:113 (0x91f318)
(227,0,0)	Running	Warp 7 Lane 3	findmax.cu:113 (0x91f318)

findmax.cu

```
uint32_t nextElement;
uint32_t i = firstElementIndex + threadsCount;

for (; i < ARRAY_SIZE; i += threadsCount) {
    nextElement = array[i];
    if (nextElement > max) {
        max = nextElement;
        maxIndex = i;
    }
}
threadMax[threadIdx.x] = max;
threadMaxIdx[threadIdx.x] = maxIndex;
```

Console

findmax [C/C++ Application] findmax  
Running single-threaded host code  
Max number is 0x800000 with index 2737098  
  
Running multi-threaded device code

Outline Disassembly Registers

Name	T(0,0,0)B(0,0,0)	T(1,0,0)B(0,0,0)
R0	0	1
R1	16776272	16776272
R2	4935629	2024586
R3	8192	8193
R4	3149939	8115414
R5	4	4
R6	1048576	1048576
R7	4	4
R8	32768	32772
R9	0	0
R10	8387951	16778240
R11	0	0
R12	1048576	1048576
R13	0	0
R14	0	0

# Task 2

## TASK 2

### Debug with cuda-gdb

- Location of code: 2-Tools/exercises/tasks/task2
- Steps (see also Instructions.ipynb)
  - Let thread 4 from first block print 42 (instead of 0)  
Do not change the source code! Use the variable view.
  - Build program: make
  - Debug with **cuda-gdb**
    - 1 First, start interactive compute session with  
`eval $JSC_SUBMIT_CMD bash -i`
    - 2 and then...  
`cuda-gdb ... start cuda-gdb (see also make debug-cuda-gdb)`  
(See solutions directory for a solution of issued commands!)

# Profiling

# Motivation for Measuring Performance

- **Improvement** possible only if program is **measured**

*Don't trust your gut!*

- Identify:

**Hotspots** Which functions take most of the time?

**Bottlenecks** What are the limiters of performance?

- Manual timing possible, but tedious and error-prone  
Feasible for small applications, impractical for complex ones

## → **Profiler**

- In-detail insights
- No code changes needed!
- Easy access to hardware counters (*PAPI, CUPTI*)

# The Nsight profiler suite

20/20 Nsight

- Visual Profiler and nvprof will be deprecated in a future CUDA release

- New tools

**Nsight Systems** System timeline, CPU/GPU sampling & tracing –  
<https://developer.nvidia.com/nsight-systems>

- module load Nsight-Systems
- nsys (CLI) and nsys-ui (GUI)

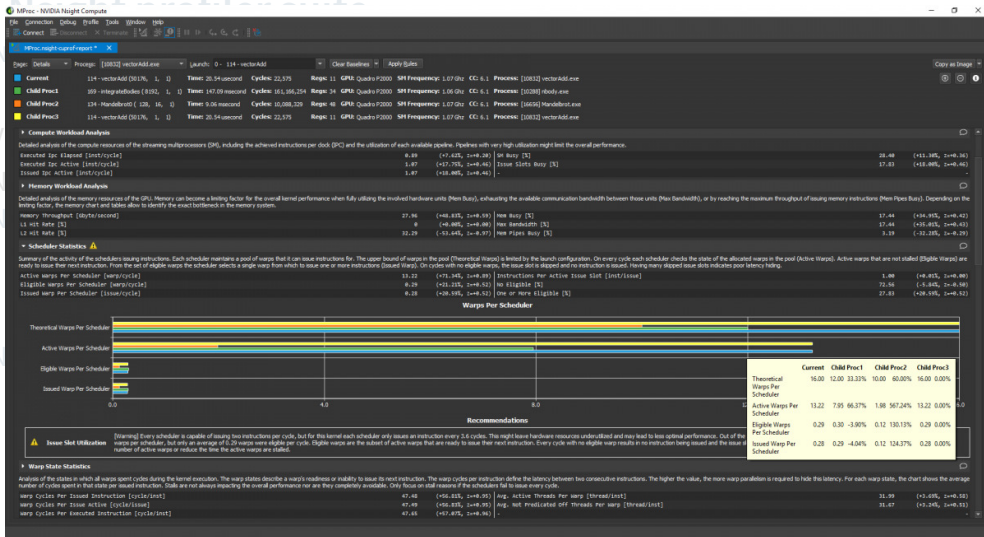
**Nsight Compute** GPU kernel profiler –  
<https://developer.nvidia.com/nsight-compute>

- module load Nsight-Compute
- ncu (CLI) and ncu-ui (GUI)

Screenshots by NVIDIA from the websites of the respective tool.








# Application Timeline

## Getting an overview through Nsight Systems

Get the necessary modules on JUWELS



```
module load GCC Nsight-Systems Nsight-Compute
```

- Record the timeline

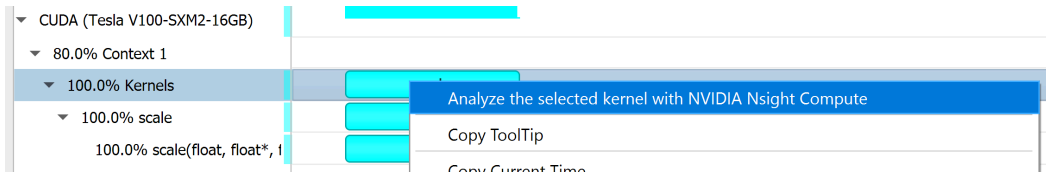
- `nsys profile -o scale_report ./scale_vector_um`

- Default set of traces selected (CUDA API, ...), many more options

→ <https://docs.nvidia.com/nsight-systems/UserGuide/index.html#cli-profiling>

# Kernel profiling

## Analysis with Nsight Compute



- Use GUI (also remotely), or run command line:

- `ncu --set full -k scale -s 0 -c 1 -f -o my_report ./scale_vector_um`

- Important switches for metrics collection, pre selected sets

- Fully customizable, `ncu --help`. Check `--list-metrics` and `--query-metrics`

→ <https://docs.nvidia.com/nsight-compute/>

# Other Profilers

Because there's so much more

- Special measurement registers (*performance counters*) of GPU exposed to third-party applications via **CUPTI** (*CUDA Profiling Tools Interface*)  
→ Enables professional profiling tools for GPU!

# Other Profilers

Because there's so much more

- Special measurement registers (*performance counters*) of GPU exposed to third-party applications via **CUPTI** (*CUDA Profiling Tools Interface*)

→ Enables professional profiling tools for GPU!

**PAPI** API for measuring performance counters, also GPU

For example: `cuda::device:0:threads_launched`

**Score-P** Measures CPU and GPU profile of program

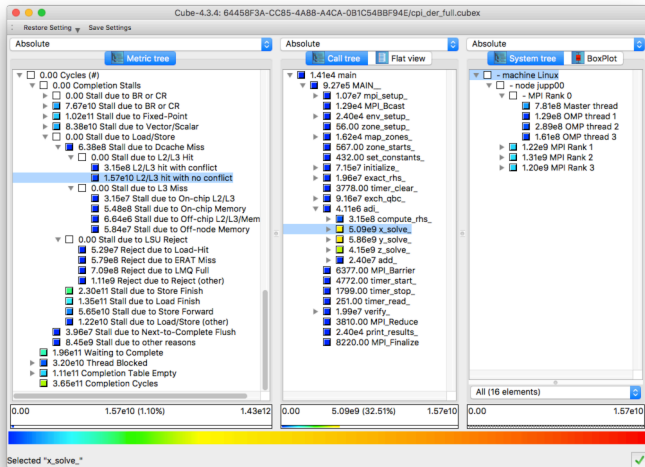
Prefix `nvcc` compilation with `scorep`, set `SCOREP_CUDA_ENABLE=yes`, run

**Cube** Displays performance report from Score-P concisely

**Vampir** Display report from Score-P in timeline view, also multiple MPI ranks

# Score-P

## Analysis with Cube



Actually, no GPU information displayed here....

# Task 3

## TASK 3

### Analyze and profile `scale_vector_um`

- Location of code: `2-Tools/exercises/tasks/task3/`
- See `Instructions.ipynb`
- Use CLI to gather profile, GUI for viewing (X-forwarding or Xpra, described in `.ipynb`)
  - Use `nsys profile` to write `scale_vector_um`'s timeline to file
  - Start Nsight Systems (`nsys-ui`) on the login node; import timeline
  - Use `ncu` to collect metric information
  - Import, analyze in Nsight Compute GUI `ncu-ui`
- Objective: Get to know the tools! What's the runtime of the kernel?



# Conclusions

## What we've learned

- Debugging
  - Detect false memory accesses with **compute-sanitizer**
  - Debug from console with **cuda-gdb**
- Profiling
  - Use **Nsight Systems/Compute** for analysis and optimization
  - **nsys, ncu** in console, also for batch jobs

# Conclusions

## What we've learned

- Debugging
  - Detect false memory accesses with **compute-sanitizer**
  - Debug from console with **cuda-gdb**
- Profiling
  - Use **Nsight Systems/Compute** for analysis and optimization
  - **nsys, ncu** in console, also for batch jobs

**Thank you  
for your attention!**  
[a.herten@fz-juelich.de](mailto:a.herten@fz-juelich.de)  
[mhrywniak@nvidia.com](mailto:mhrywniak@nvidia.com)



# APPENDIX

## Appendix Glossary

# Glossary I

**CUDA** Computing platform for **GPUs** from NVIDIA. Provides, among others, CUDA C/C++. 2, 13, 14, 15, 27

**NVIDIA** US technology company creating **GPUs**. 45

**CPU** Central Processing Unit. 37, 38

**GPU** Graphics Processing Unit. 37, 38, 39, 45

# References: Images, Graphics I