IHPCSS 2024, Kobe, Japan

Friedrich-Alexander-Universität
Erlangen-Nürnberg

GPU Performance Engineering

Sebastian Kuckuk

Erlangen National High Performance Computing Center (NHR@FAU)

Introduction

- Quick survey. Who in the audience …

- … has already experience with GPU programming
  - … with CUDA/ HIP
  - … with SYCL/OpenCL/DPC++
  - … with OpenMP target offloading
  - … with OpenACC
  - … with another technology

- … has already experience with GPU profiling

Overview

- This tutorial will cover basics of *GPU Performance Engineering*, including
  - GPU architecture
  - Potential bottlenecks (factors imposing a performance limit)
  - Writing and using micro-benchmarks
  - Harnessing profiling tools
  - Hands-on examples
- All examples and illustrations will be centered around NVIDIAs …
  - … hardware, but most concepts translate well to other vendors
  - … tools as they are the most mature (personal opinion)

Course Material

- All material is available as a git repository

  `git clone https://github.com/SebastianKuckuk/ihpcss-gpu-perf`

Folder	Contents
`/src`	Test cases – each follows name template `example/example-parallelization.cpp/cu`
`/profiles`	Output files from nsight systems and compute
`/build`	Compiled binaries
`/src/runner`	Scripts to automatically benchmark varying problem sizes
`/measurements`	Contains performance data obtained by the runner

GPU Performance Engineering

GPU Architecture

Performance Modelling

Profiling Tools

Use Cases

Outlook

GPUs in HPC

- Promises
  - Massive parallelism and performance
  - Good performance in relation to energy (FLOPs per Watt)

- Already widespread in the HPC landscape
  - c.f. Top500 list (https://www.top500.org/lists/top500/2024/06/)
    - 9 out of the top 10 supercomputers are equipped with GPUs
    - 6 NVIDIA (3 x H100, GH200, A100, V100)
    - 2 AMD (2 x MI250X)
    - 1 Intel (GPU Max Series)

- Where does the performance come from?

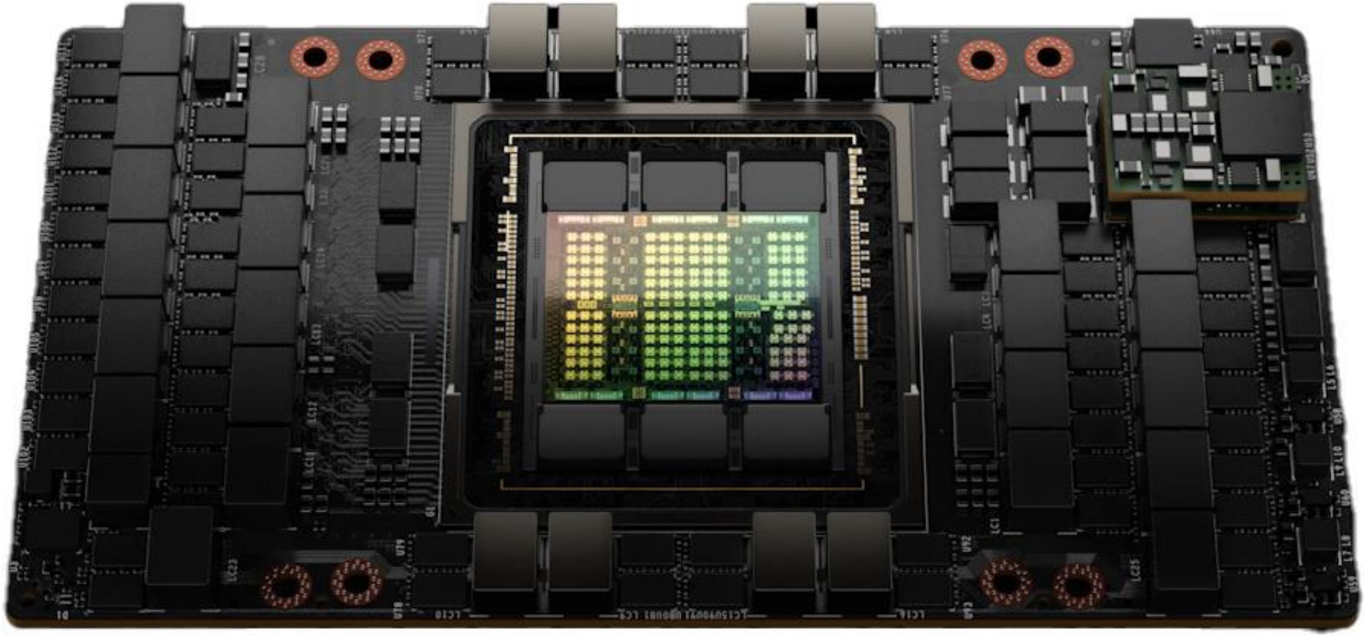
GPUs in HPC

- Where does the performance come from?

- Two main contributors
  - Massively parallel computational throughput
    - Doing the actual meaningful computations
  - High memory bandwidth
    - Getting the required input data and storing the output

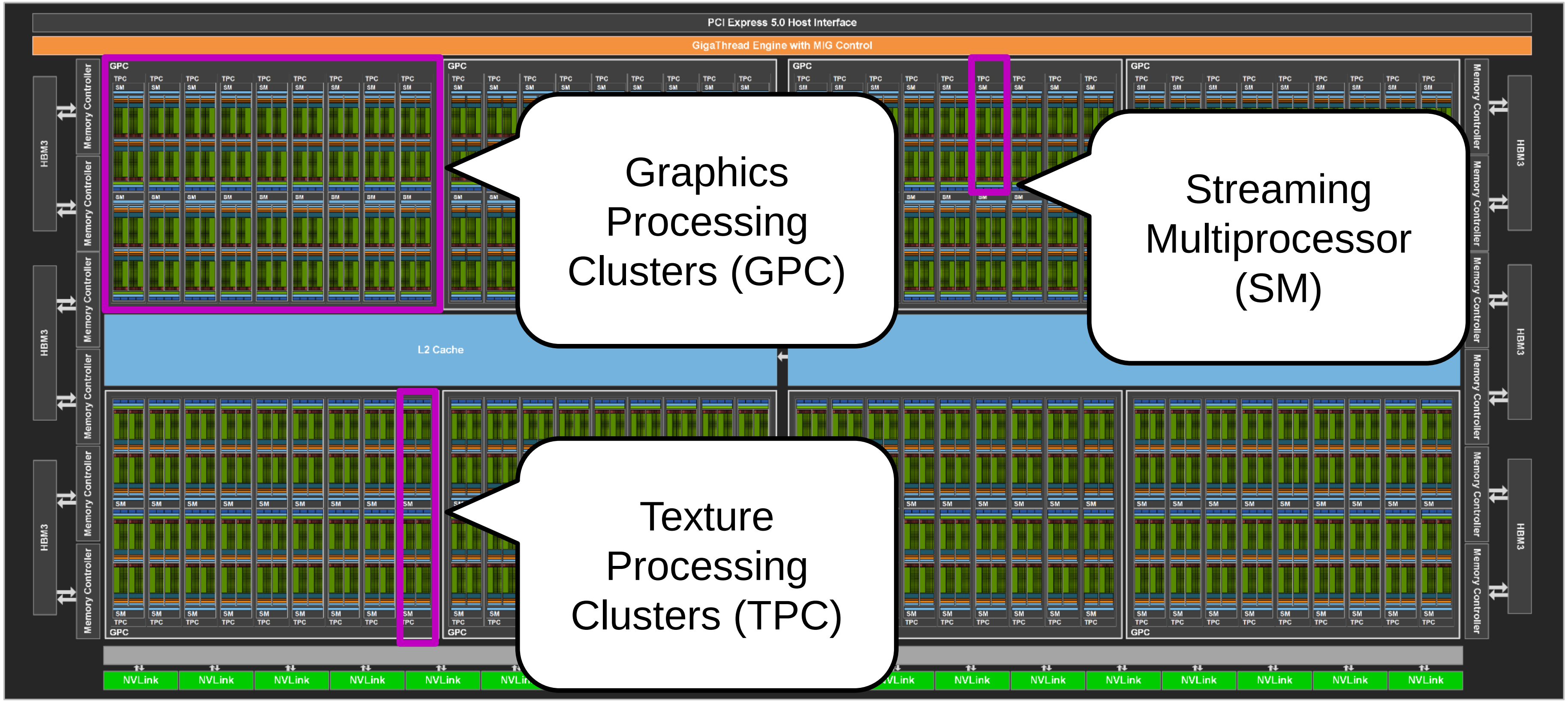
GPU Architecture – Example: H100

- Detailed documentation as whitepaper
  - https://resources.nvidia.com/en-us-tensor-core



GPU Architecture – Example: H100

GPU Architecture – Example: H100



GPU Architecture – Example: H100

- Previous slides show ‘full configuration’

- One H100 GPU features
  - 8 Graphics Processing Clusters (GPCs), each with
  - 8 or 9 Texture Processing Clusters (TPCs), each with
  - Two Streaming Multiprocessors (SMs)

➢ Total of 132 SMs

GPU Architecture – Example: H100

- Total of 132 SMs, each with
- Four sub partitions, each with
- 16 INT32 units
  - Total: 132 * 4 * 16 = 8448
- 32 FP32 units
  - Total: 132 * 4 * 32 = 16896
- 16 FP64 units
  - Total: 132 * 4 * 16 = 8448
- One tensor core
  - Total: 132 * 4 = 528
  - Each: 512 FP16/FP32-mixed-prec. FMAs

GPU Architecture – Example: H100

- Performance: $P_{chip} = n_{core} \cdot n_{super}^{FP} \cdot n_{FMA} \cdot n_{SIMD} \cdot f$

#Cores

Super-scalarity

FMA factor

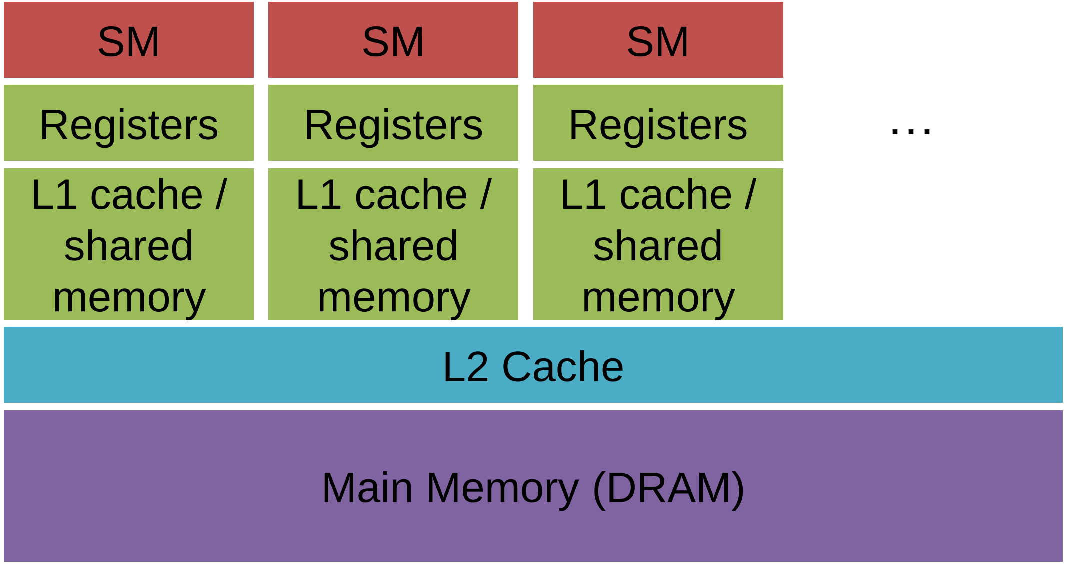
SIMD factor

Clock Speed

- For FP32
  - $P_{chip} = 16896 \cdot 1 \cdot 2 \cdot 1 \cdot 1.98\ GF/s = 66.9\ TF/s$
    $= 528 \cdot 1 \cdot 2 \cdot 32 \cdot 1.98\ GF/s$

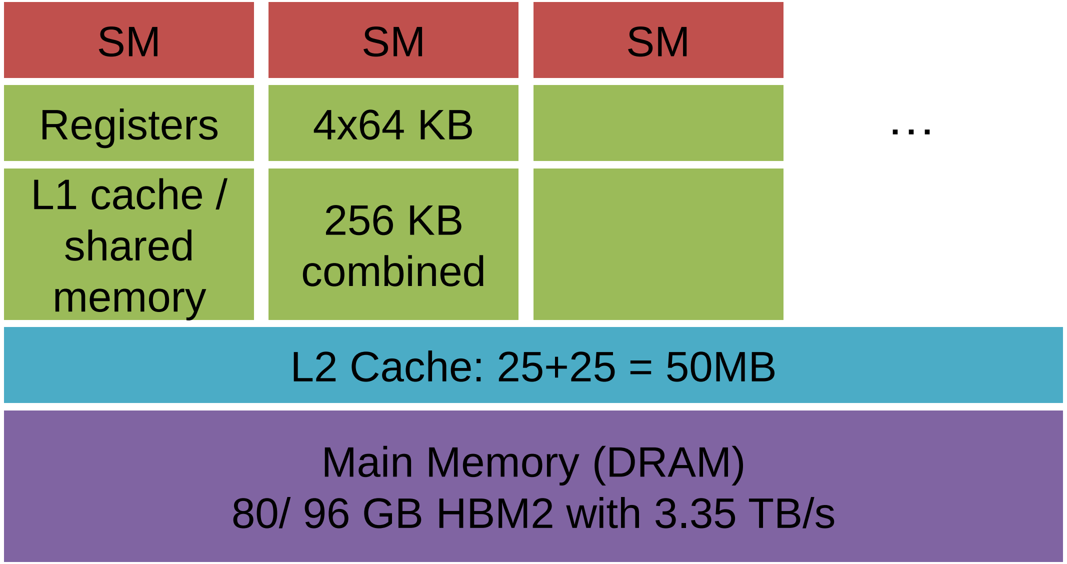
GPU Architecture – Example: H100

- Memory also follows a specific hierarchy
- Not shown: constant memory, texture memory, …



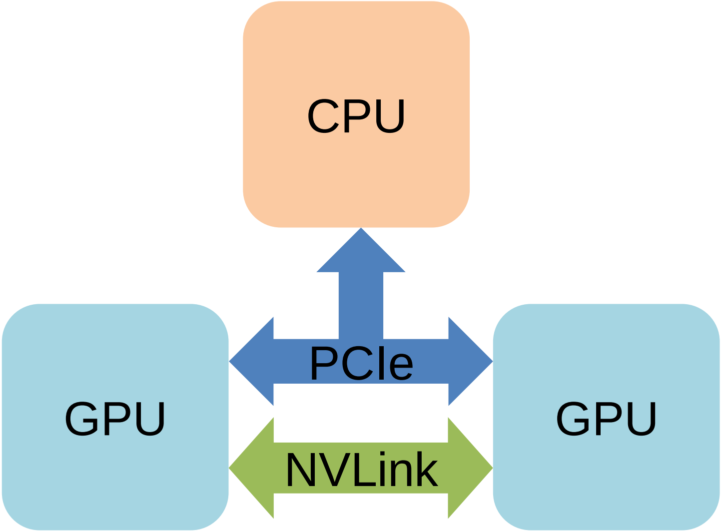
GPU Architecture – Example: H100

- Memory also follows a specific hierarchy
- Not shown: constant memory, texture memory, …



GPU Architecture – Example: H100

- GPUs are only a part of the system
- Reference: main memory with 3.35 TB/s
- Connection to CPU with PCIe 5.0 x16 with 64 GB/s per direction
- Connection to other GPUs in the same node via NVLink with 450 GB/s per direction



GPU Architecture – Generation Comparison

	V100 SXM 32 GB	A100 SXM 80 GB	H100 SXM 96 GB
Compute Capability	7.0	8.0	9.0
#Cores FP32	5120 (80 * 64)	6912 (108 * 64)	16896 (132 * 128)
FP32 Perf. [TFLOPS]	16	19	67
FP64 Perf. [TFLOPS]	7	10	34
FP64:FP32 Ratio	1:2	1:2	1:2

GPU Architecture – Generation Comparison

	V100 SXM 32 GB	A100 SXM 80 GB	H100 SXM 96 GB
Compute Capability	7.0	8.0	9.0
#Cores FP32	5120 (80 * 64)	6912 (108 * 64)	16896 (132 * 128)
FP32 Perf. [TFLOPS]	16	19	67
FP64 Perf. [TFLOPS]	7	10	34
FP64:FP32 Ratio	1:2	1:2	1:2
Memory [GB]	32	80	96
L2 Cache [MB]	6	40	50

GPU Architecture – Generation Comparison

	V100 SXM 32 GB	A100 SXM 80 GB	H100 SXM 96 GB
Compute Capability	7.0	8.0	9.0
#Cores FP32	5120 (80 * 64)	6912 (108 * 64)	16896 (132 * 128)
FP32 Perf. [TFLOPS]	16	19	67
FP64 Perf. [TFLOPS]	7	10	34
FP64:FP32 Ratio	1:2	1:2	1:2
Memory [GB]	32	80	96
L2 Cache [MB]	6	40	50
Bandwidth [GB/s]	981	2039	3352
PCIe (BW/ direction [GB/s])	3.0 x16 (16)	4.0 x16 (32)	5.0 x16 (63)
NVLink per direction [GB/s]	150	300	450

GPU Architecture – Generation Comparison

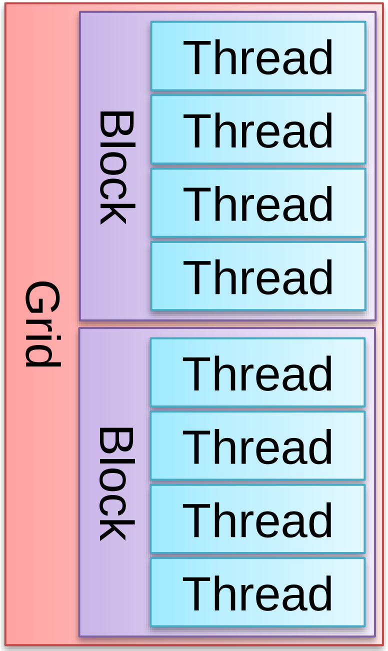
	V100 SXM 32 GB	A100 SXM 80 GB	H100 SXM 96 GB
Compute Capability	7.0	8.0	9.0
#Cores FP32	5120 (80 * 64)	6912 (108 * 64)	16896 (132 * 128)
FP32 Perf. [TFLOPS]	16	19	67
FP64 Perf. [TFLOPS]	7	10	34
FP64:FP32 Ratio	1:2	1:2	1:2
Memory [GB]	32	80	96
L2 Cache [MB]	6	40	50
Bandwidth [GB/s]	981	2039	3352
PCIe (BW/ direction [GB/s])	3.0 x16 (16)	4.0 x16 (32)	5.0 x16 (63)
NVLink per direction [GB/s]	150	300	450
TDP [W]	250	400	700

GPU Parallelism

- Threads executing GPU kernels are organized hierarchically
  - Multiple **threads** form on (thread) **block**
  - Multiple thread **blocks** form a **grid**

- This is done to match the hierarchical hardware
  - Each **Grid** is executed on one **device** (GPU)
  - **Blocks** are mapped to **SMs**
  - **Threads** are mapped to **cores** (execution units)

  - **Threads** of a single block are executed in **warps** (groups of **32 threads**)



GPU Performance Engineering

GPU Architecture

Performance Modelling

Profiling Tools

Use Cases

Outlook

Performance Modelling – General Approach

1. Obtain relevant metrics for relevant code regions

   - Execution time
   - Bytes read/ written
   - FLOPS performed
   - And many more …

   Somewhere between kernels/ loops and whole application; usually limited to ‘meaningful’ work

2. Relate observed performance to theoretical limits
   - … or to measured limits – see next slides

3. Figure out where optimization is possible/ reasonable
   - Try to hit at least one bottleneck, then shift to another bottleneck

Performance Modelling – General Approach

1. Obtain relevant metrics for relevant code regions

2. Relate observed performance to theoretical limits

3. Figure out where optimization is possible/ reasonable

➢ Tools can be a great asset in all steps

Micro Benchmarks

- Motivation: theoretical limits can be too optimistic, micro benchmarks give a more realistic expectation
- Two examples: computational performance and sustained bandwidth
- Codes in
  - `src/fma/fma-*`
  - `src/stream/stream-*`
- This also represents a manual approach to performance modelling – doing explicit timing combined with manual code analysis

Micro Benchmarks

- Live Demo on Bridges-2

```
git clone https://github.com/SebastianKuckuk/ihpcss-gpu-perf

module load nvhpc/21.7

interact -p GPU-shared --gres=gpu:v100-32:1 --time 2:00:00

cd src/fma
make –j bench

cd ../stream
make –j bench
```

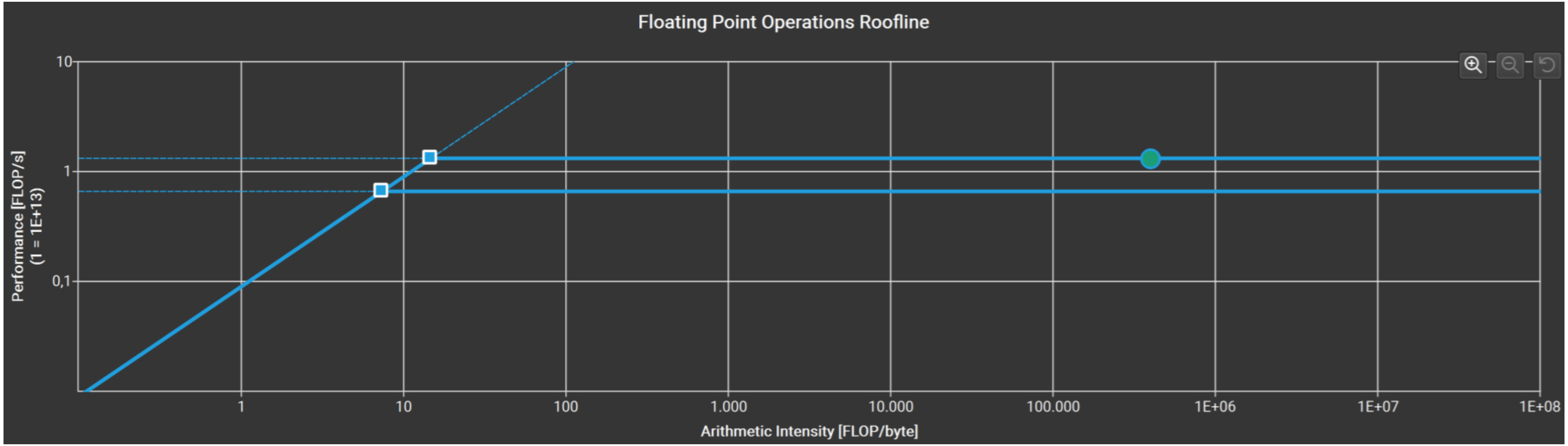
Micro Benchmarks

```
OpenMP Target:
../../build/fma/fma-omp-target
$((1024 * 1024)) 2 10
  #cells / #it:   1048576 / 10
  elapsed time:   1444.26 ms
  per iteration:  144.426 ms
  MLUP/s:         7.26031
  bandwidth:      0.0580825 GB/s
  compute:        15226 GFLOP/s
  Passed result check
```

```
OpenMP Target:
../../build/stream/stream-omp-target
$((64 * 1024 * 1024)) 2 10
  #cells / #it:   67108864 / 10
  elapsed time:   13.8456 ms
  per iteration:  1.38456 ms
  MLUP/s:         48469.4
  bandwidth:      775.51 GB/s
  compute:        48.4694 GFLOP/s
  Passed result check
```

Roofline Diagram

- A roofline model can be set up based on
  - Theoretical or measured peak performance
  - Arithmetic intensity of the kernel (FLOPS performed per byte transferred)
  - Nsight compute can set one up automatically – but might include unexpected effect (c.f. later slides and demo)



Further GPU Performance Inhibitors

Plotting data in roofline model leads to one of three possibilities

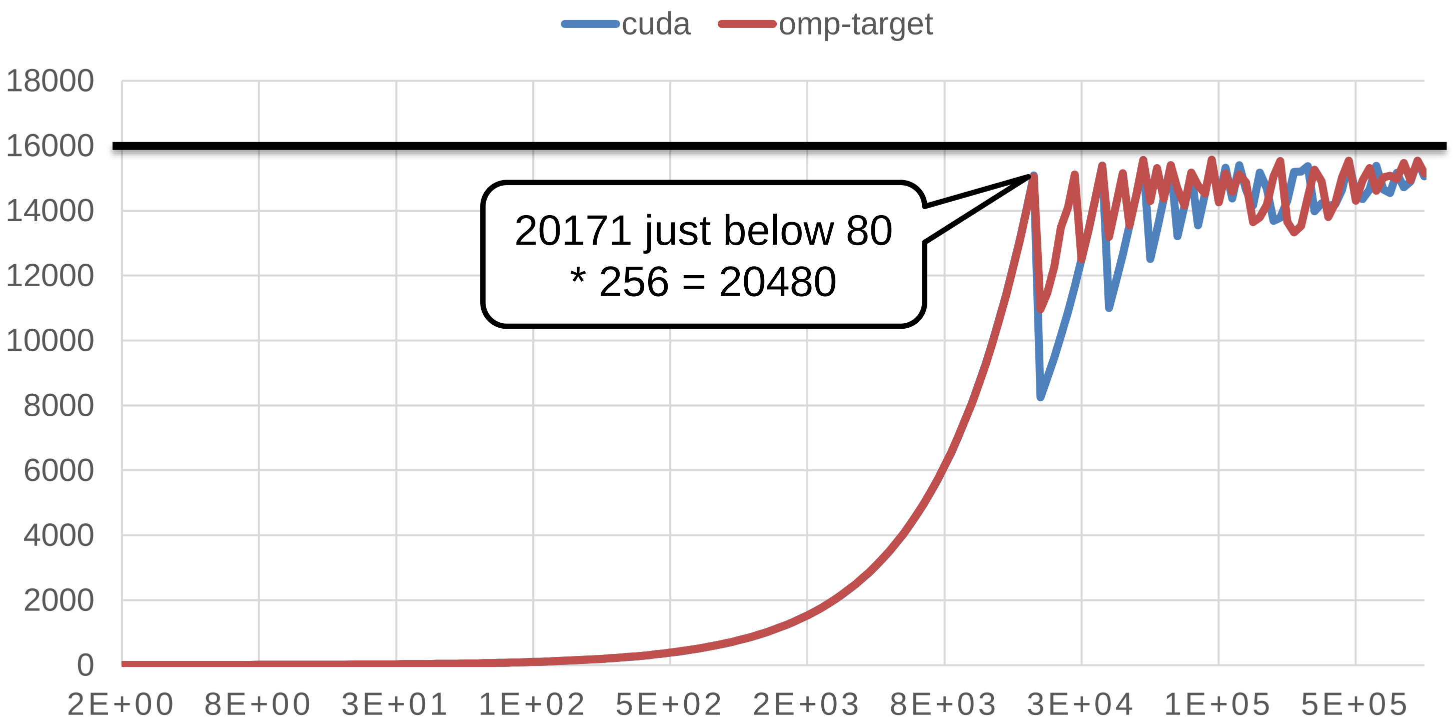
1. Measured performance is at the roofline
   - That's good, the application uses the available resources well
2. Measured performance is above the roofline
   - Usually: either optimization/ elimination of computations/ memory accesses, or
   - the peak bandwidth is higher than the limit (e.g. due to caching)
3. Measured performance is (way) below the roofline
   - Further investigation is needed to identify (and fix) potential performance issues

GPU Performance Issues

- Issue: GPUs require massive parallelism
  - To utilize all computational resources (~ 10k threads)
  - To hide memory access latencies with running multiple threads (~ 100k threads)
- The following slides show results obtained with the same benchmark code
  - You can use the runner scripts to replicate
  - Results can be very sensitive against chosen parallelization parameters (number of threads, size of thread blocks)
- All results are obtained on a single V100-SXM2-32GB GPU (Bridges-2) and A100-SXM4-80GB GPU

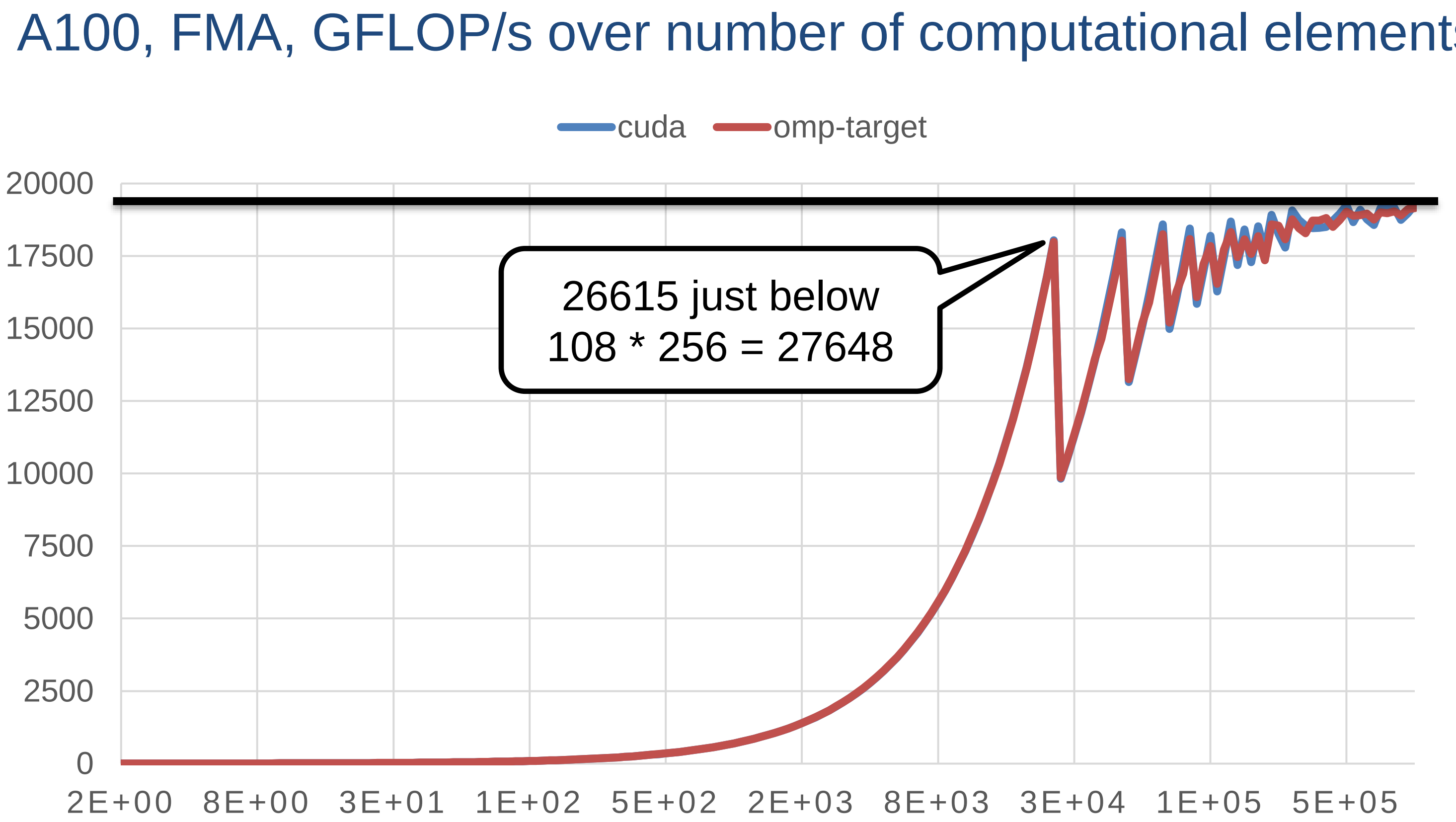
GPU Performance Issues

V100, FMA, GFLOP/s over number of computational elements



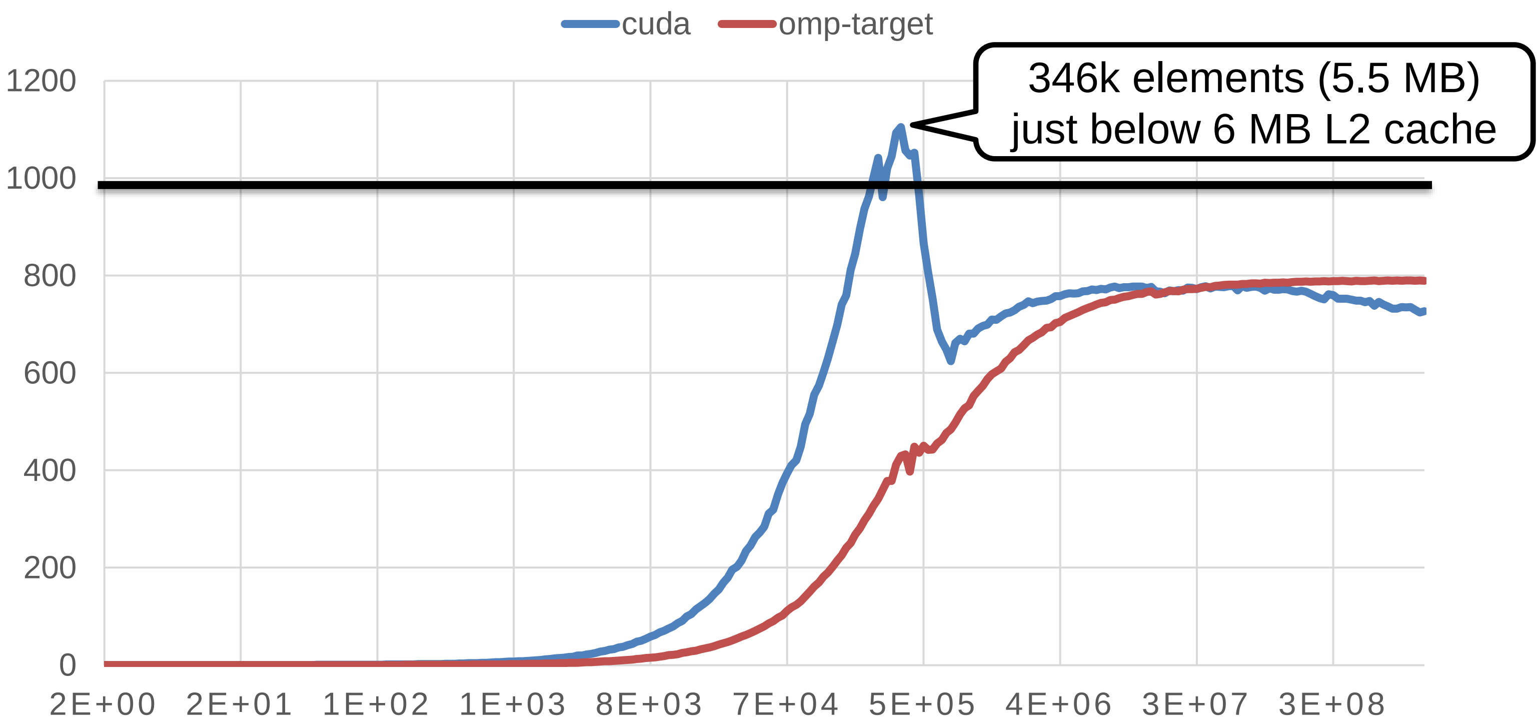
sebastian.kuckuk@fau.de

GPU Performance Issues



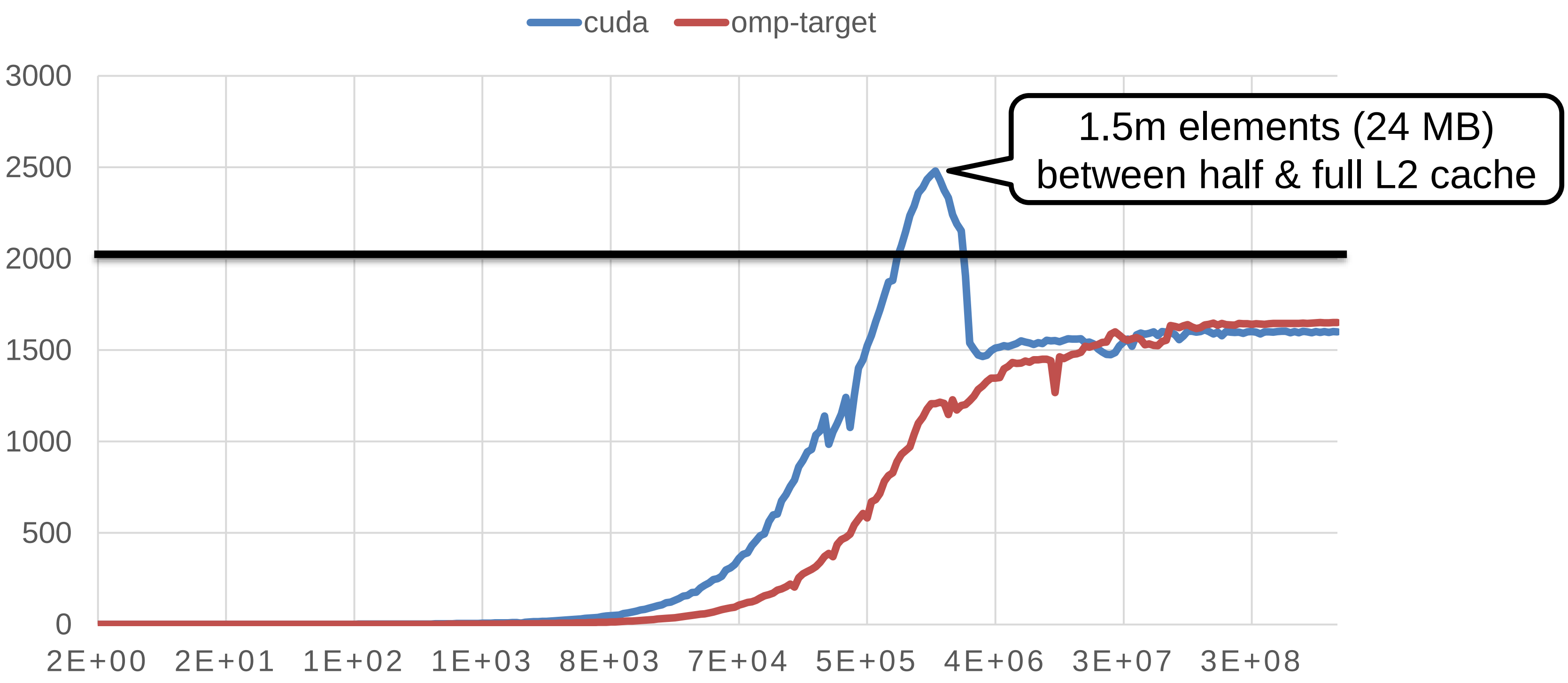
GPU Performance Issues

V100, stream, bandwidth in GB/s over number of array elements



GPU Performance Issues

A100, stream, bandwidth in GB/s over number of array elements



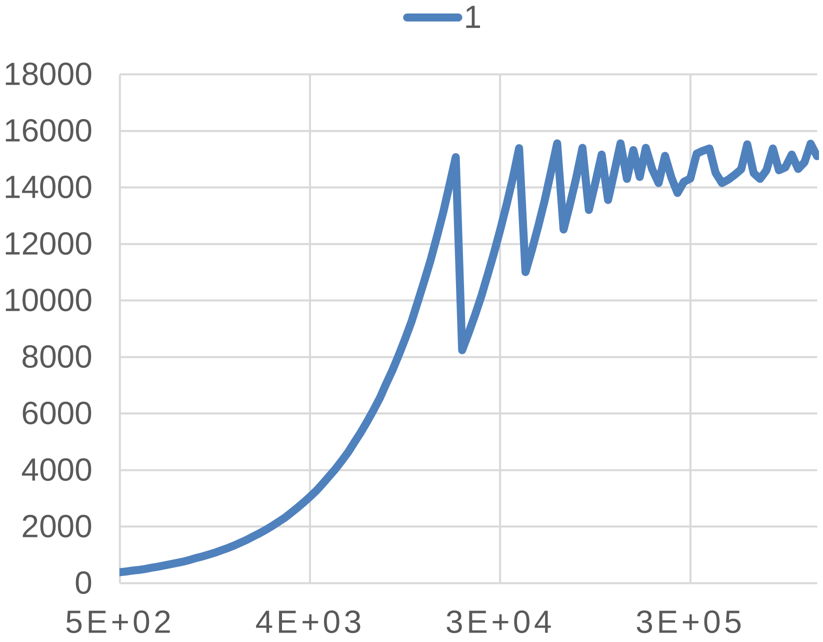
GPU Performance Issues

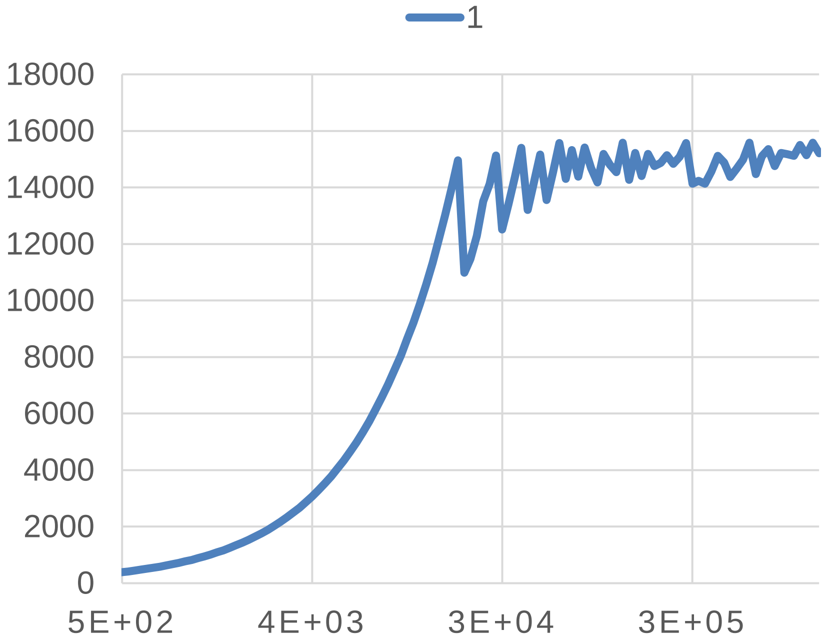
- Issue: GPUs require uniformity
  - Branch divergence
  - Granularity in memory accesses

- Adaptation of the previous benchmarks by adding strides
  - `src/strided-stream`
  - `src/strided-fma`

GPU Performance Issues

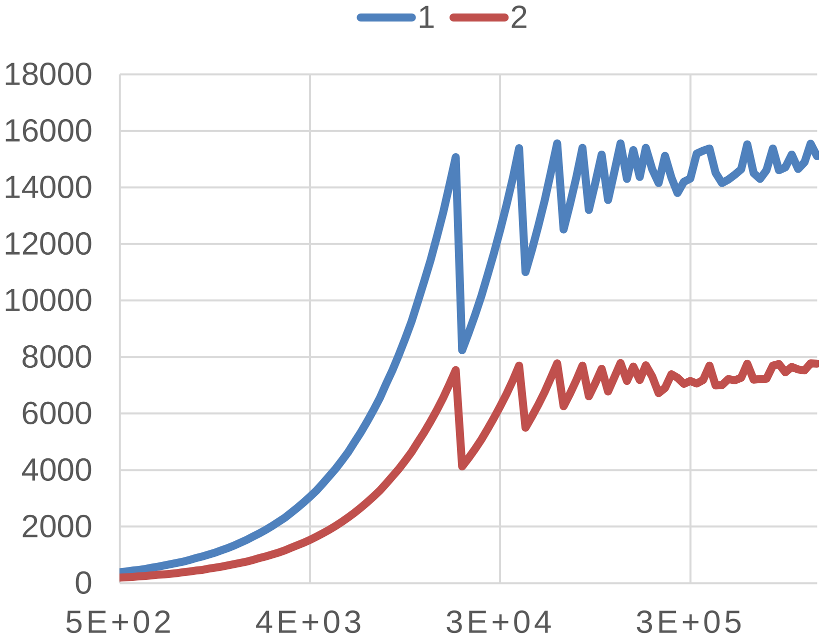
V100, strided FMA, GFLOP/s over number of computational elements
CUDA (left) and OpenMP target (right)

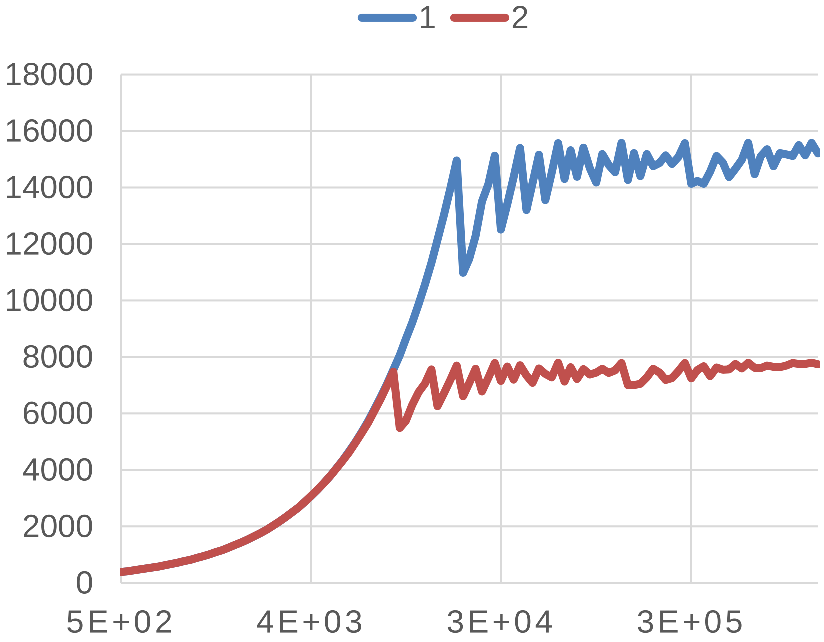




GPU Performance Issues

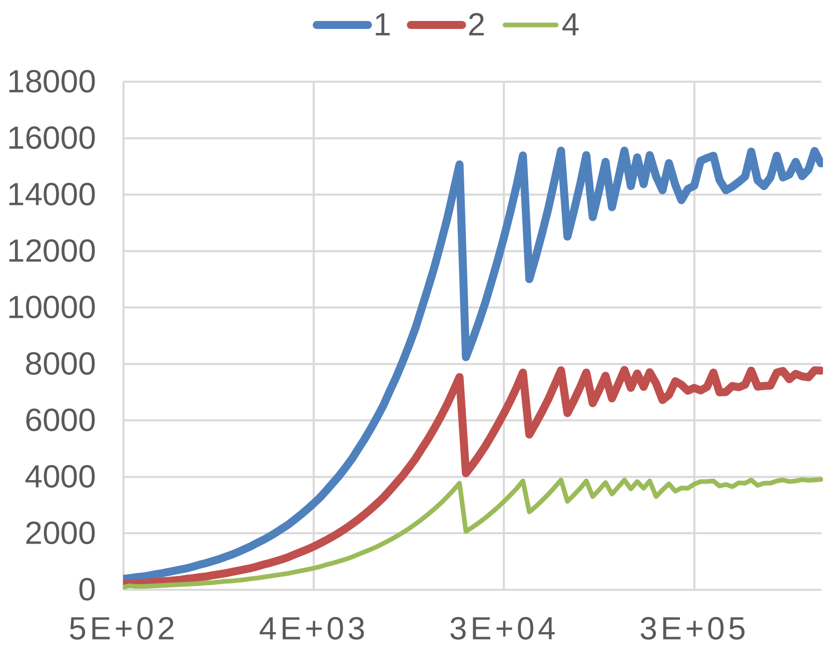
V100, strided FMA, GFLOP/s over number of computational elements
CUDA (left) and OpenMP target (right)

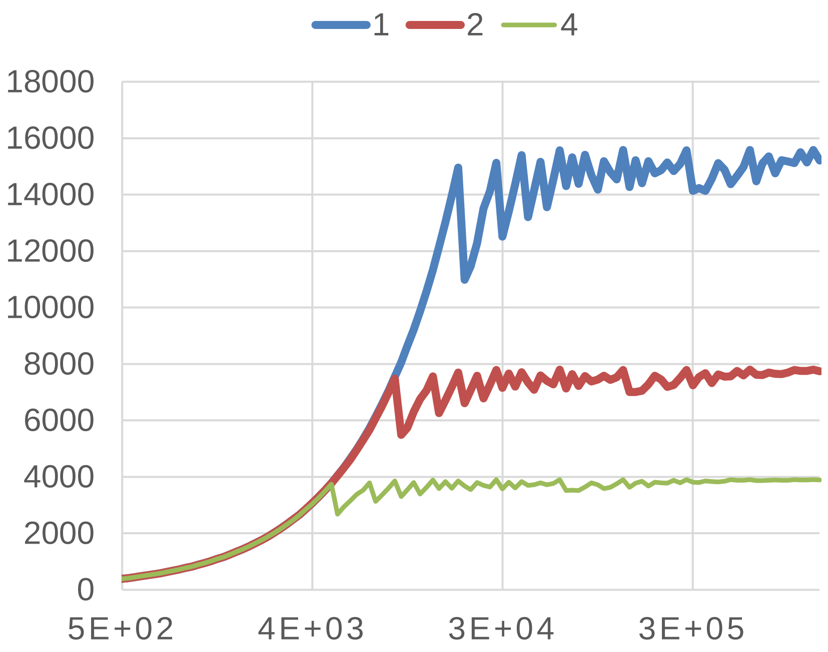




GPU Performance Issues

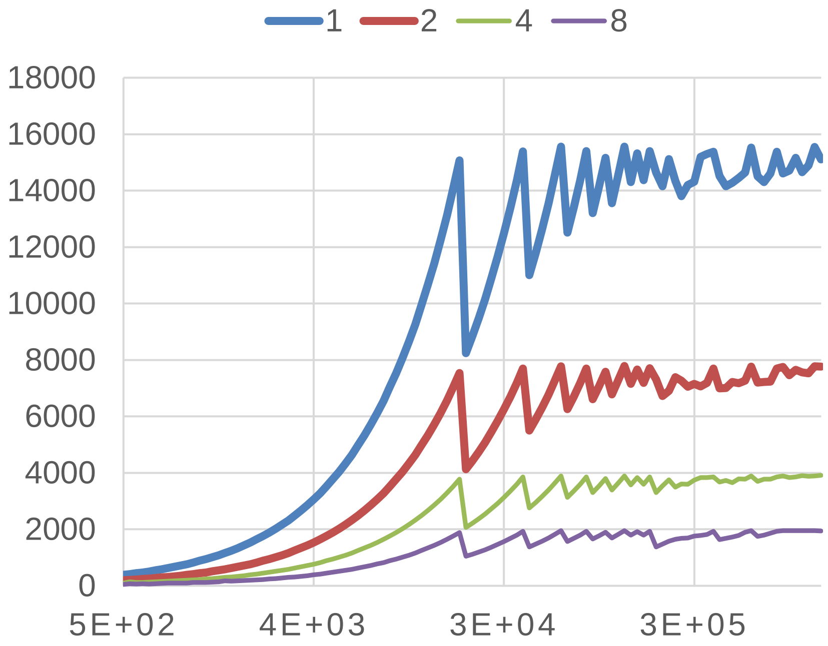
V100, strided FMA, GFLOP/s over number of computational elements
CUDA (left) and OpenMP target (right)

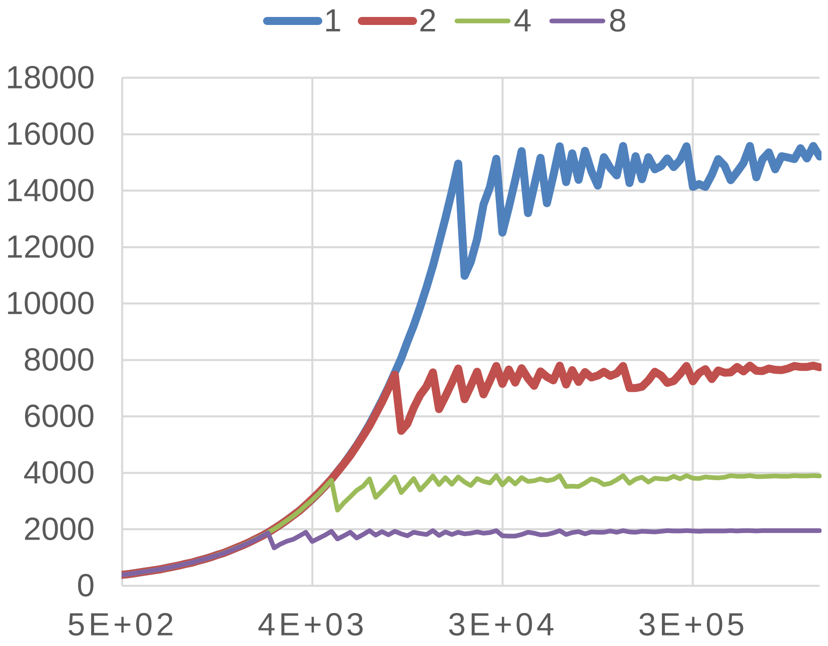




GPU Performance Issues

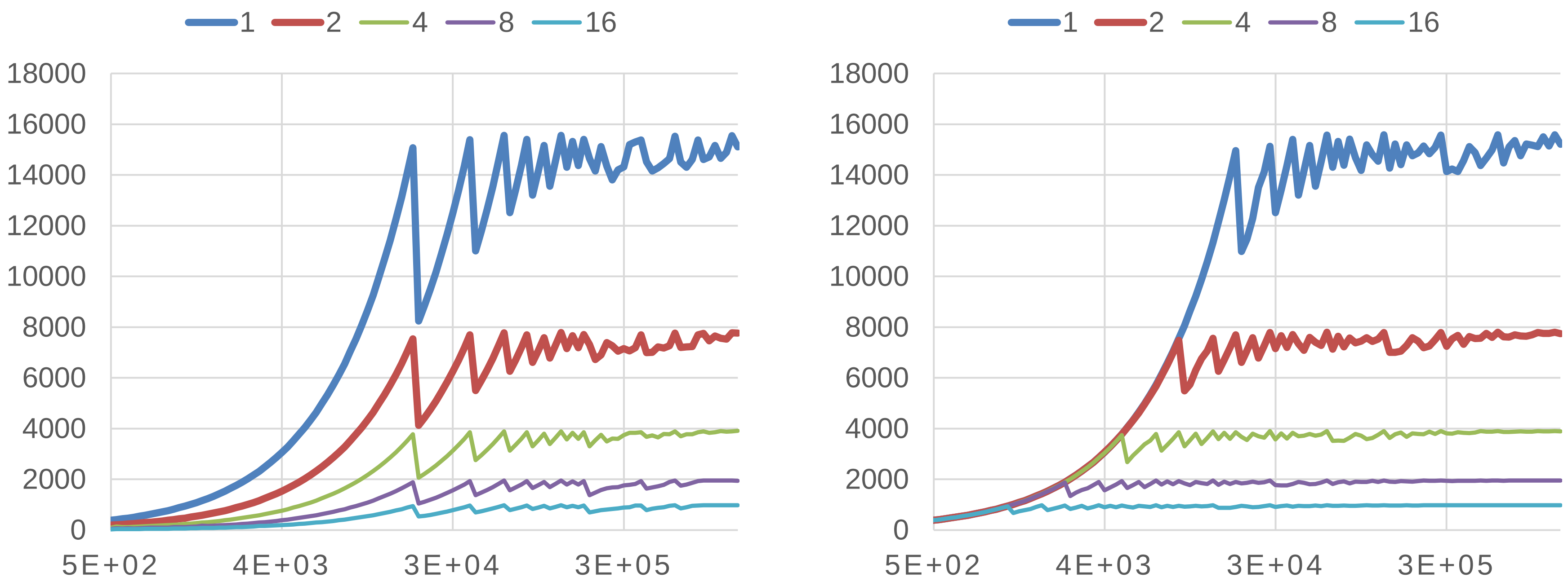
V100, strided FMA, GFLOP/s over number of computational elements
CUDA (left) and OpenMP target (right)





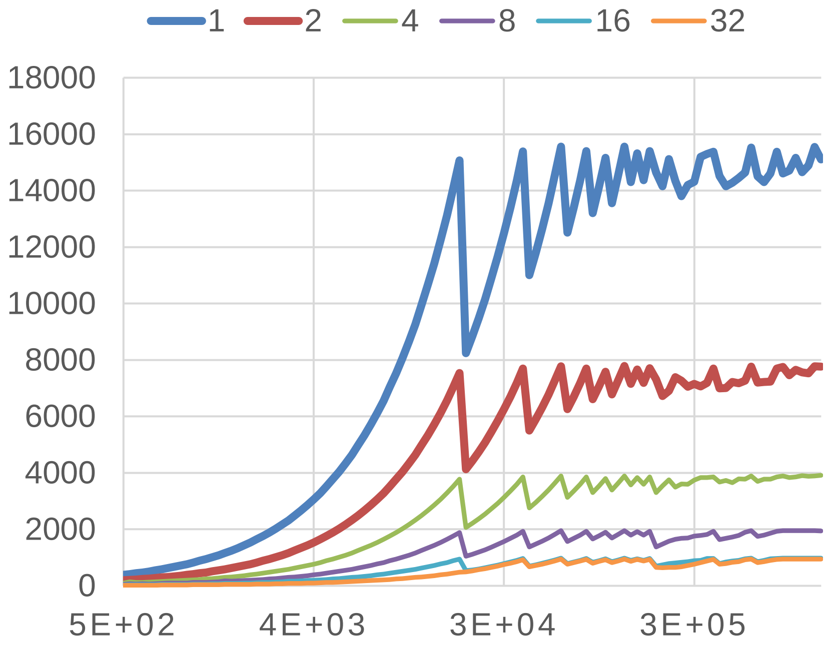
GPU Performance Issues

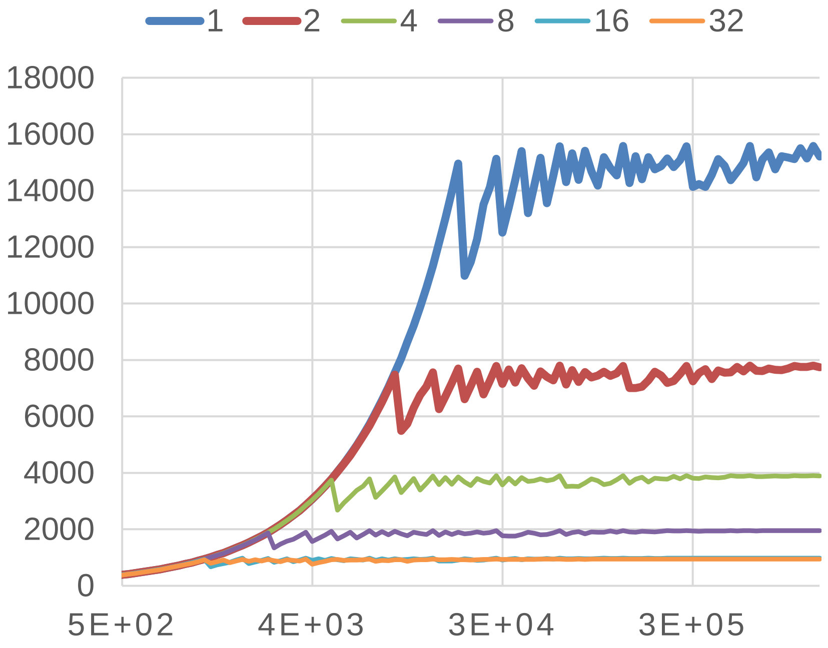
V100, strided FMA, GFLOP/s over number of computational elements
CUDA (left) and OpenMP target (right)



GPU Performance Issues

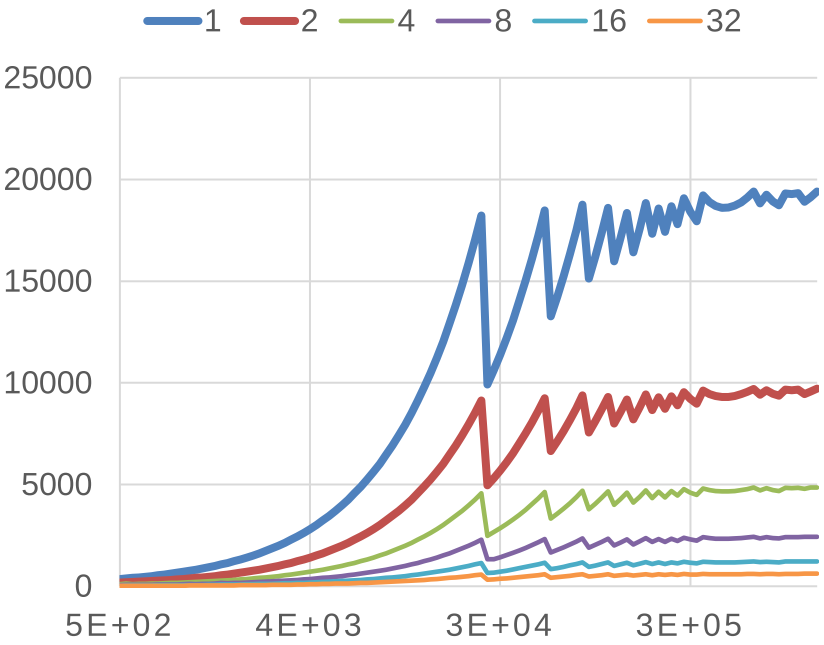
V100, strided FMA, GFLOP/s over number of computational elements
CUDA (left) and OpenMP target (right)

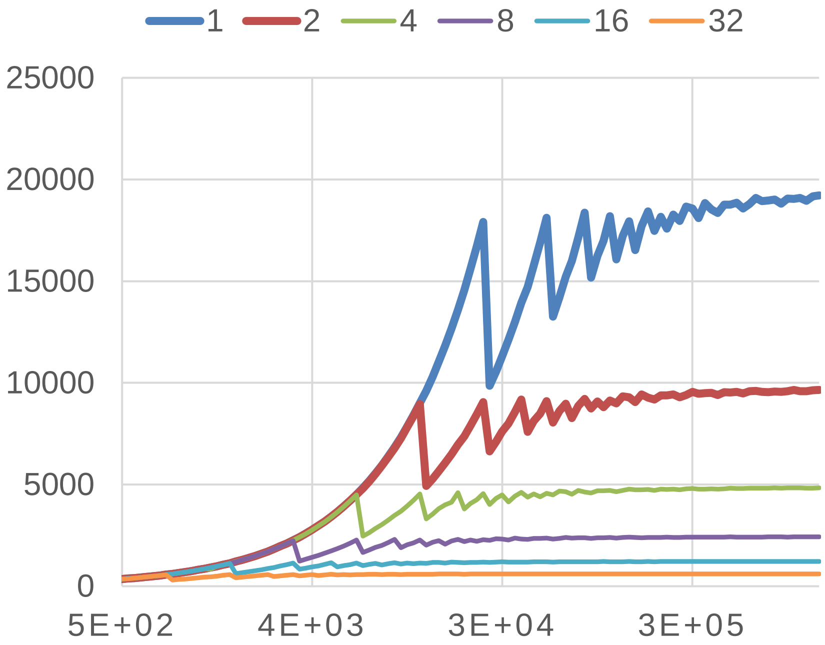




GPU Performance Issues

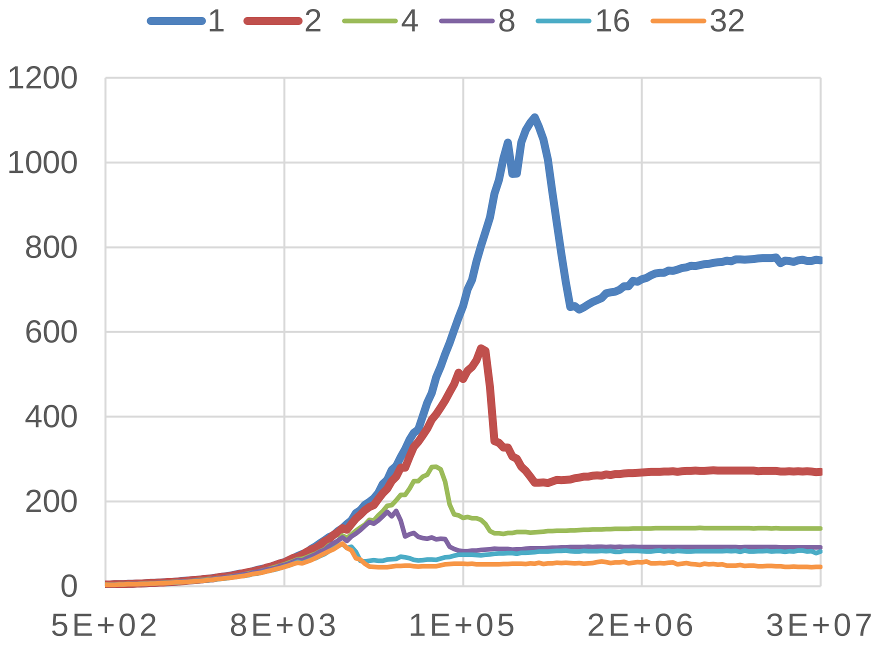
A100, strided FMA, GFLOP/s over number of computational elements
CUDA (left) and OpenMP target (right)

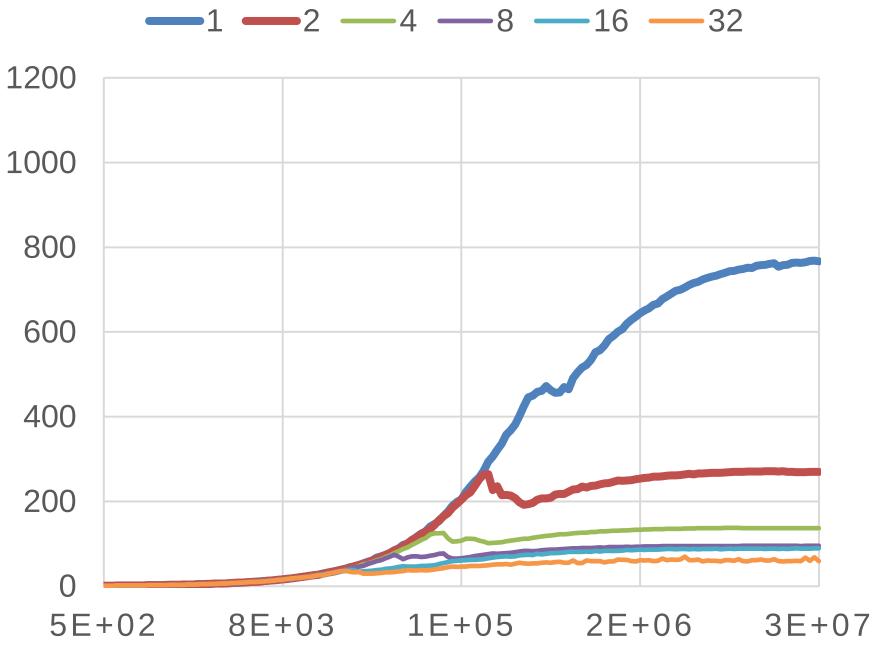




GPU Performance Issues

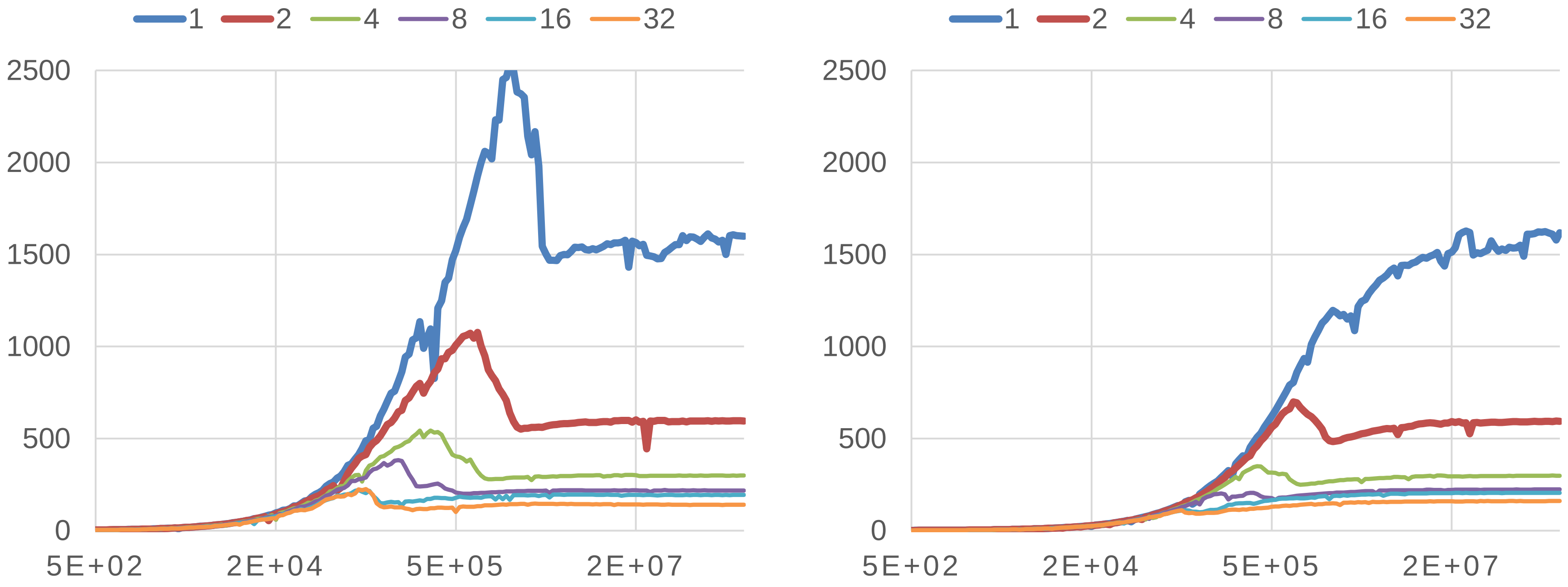
V100, strided stream, bandwidth in GB/s over number of array elements
CUDA (left) and OpenMP target (right)





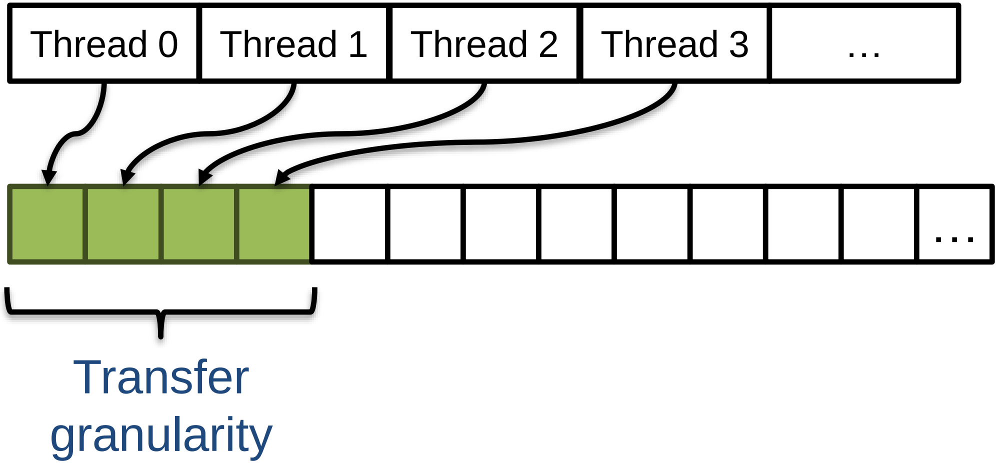
GPU Performance Issues

A100, strided stream, bandwidth in GB/s over number of array elements
CUDA (left) and OpenMP target (right)



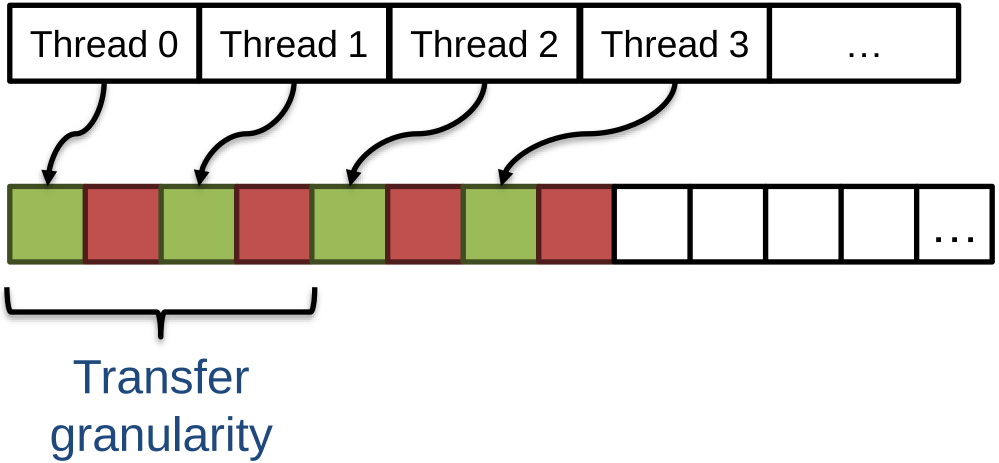
Memory Coalescing

- Coalesced access: consecutive threads access consecutive memory locations



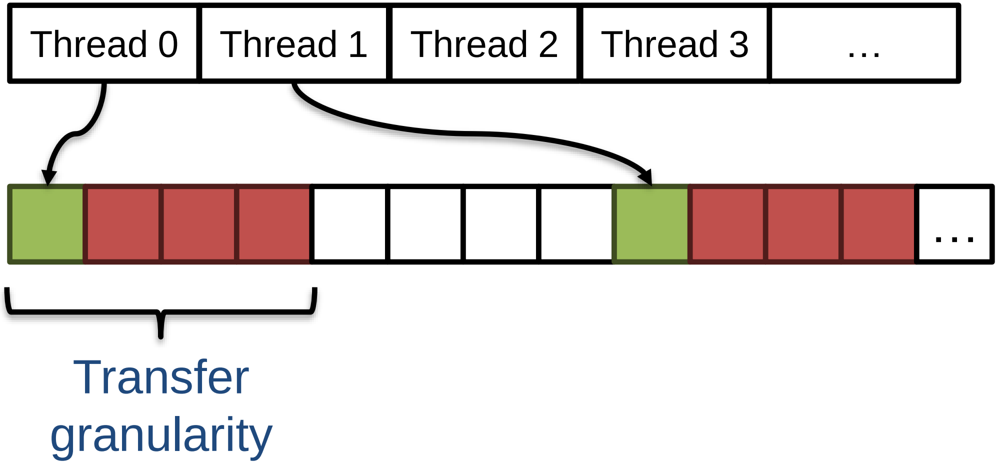
Memory Coalescing

- Uncoalesced access: additional, unused data has to be transferred
- Granularity depends on GPU and on memory space (DRAM, L2, L1, …)



Memory Coalescing

- Uncoalesced access: additional, unused data has to be transferred
- Granularity depends on GPU and on memory space (DRAM, L2, L1, …)



GPU Performance Issues – Summary

- If the observed performance is ‘not good enough’
  - Check for lack of parallelism
  - Check for diverging code paths
  - Investigate sensitivity to parameter changes
  - Compare against micro-benchmarks

- If ‘bad’ performance still cannot be explained, there is most likely a different limiter in play, e.g.
  - L1/ L2 cache bandwidth
  - L1/ shared memory bank conflicts
  - Atomic congestion
  - …

GPU Performance Engineering

- GPU Architecture
- Performance Modelling
- Profiling Tools
- Use Cases
- Outlook

Tool support

- Manual performance engineering can be challenging
  - Hard to isolate ‘hot spots’ relevant for profiling
  - Compilers may apply additional optimizations
  - Often requires in-depth knowledge about GPU architecture

- One remedy: profiling tools
  - *NVIDIA nsight systems* provides a *whole application* view
  - *NVIDIA nsight compute* investigates in-depth *kernel* performance

Nsight Systems

- Systems gives a first application-level overview

- Two main workflows

1. Profile application on target machine and print aggregated information directly on the command line

2. Profile application on target machine, open resulting profile in GUI locally (Both machines may be the same)

Nsight Systems

1. Profile application on target machine and print aggregated information directly on the command line

```
nsys profile \
    -o /tmp/my-profile --force-overwrite=true \
    --stats=true \
    my-app my-parameters
```

Also available as make target: `make nsys-stats`

Example outputs: /profiles/*gpu*/*test*/*test-backend*-nsys-stats

Nsight Systems

1. Profile application on target machine and print aggregated information directly on the command line

```
nsys profile \
    -o /tmp/my-profile --force-overwrite=true \
    --stats=true \
    my-app my-parameters
```

Live Demo

Also available as make target: `make nsys-stats`

Example outputs: /profiles/*gpu*/*test*/*test-backend*-nsys-stats

Nsight Systems

2. Profile application on target machine, open resulting profile in GUI locally
   (Both machines may be the same)

```
nsys profile \
    -o my-profile --force-overwrite=true \
    my-app my-parameters
```

Also available as make target: `make nsys`

Example outputs: `/profiles/`*`gpu/test/test-backend`*`.qdrep`

`/profiles/`*`gpu/test/test-backend`*`.nsys-rep`

Nsight Systems

2. Profile application on target machine, open resulting profile in GUI locally (Both machines may be the same)

```
nsys profile \
    -o my-profile --force-overwrite=true \
    my-app my-parameters
```

Live Demo

Also available as make target: `make nsys`

Example outputs: `/profiles/gpu/test/test-backend.qdrep`
`/profiles/gpu/test/test-backend.nsys-rep`

Nsight Systems

- Systems gives a first application-level overview
  - Allows identifying regions of interest

- No instrumentalization necessary – simply profile your binary
  - Can also be used with closed source projects

- Documentation

  https://docs.nvidia.com/nsight-systems/UserGuide/index.html

Nsight Systems

- Useful command line arguments

```
--delay=...
--duration=...

--trace=mpi --mpi-impl=...

--trace=...
--sample=...
```

https://docs.nvidia.com/nsight-systems/UserGuide/index.html#cli-profile-command-switch-options

Nsight Compute

- After using nsight systems to isolate kernels of interest, nsight compute can give more insight into their performance characteristics

- Different workflows are available, each can be customized

- Collects specific *metrics*, e.g.
  - Bytes transferred from DRAM to L2
  - Double precision FMAs performed
  - …

- Aims to provide guidance in optimization

Nsight Compute

- Can be used from the GUI itself
  - Works best when profiling on the same machine

- Extensive command line interface

  https://docs.nvidia.com/nsight-compute/NsightComputeCli

- Detailed explanation of metrics

  https://docs.nvidia.com/nsight-compute/ProfilingGuide/index.html#metrics-structure

Nsight Compute

1. Basic usage

```
ncu \
    my-app my-parameters
```

- Prints general information on the command line for *each kernel*
- Will ‘replay’ kernels multiple times to obtain different metrics
- Output can be quite lengthy and difficult to analyze

Nsight Compute

2. Basic usage + filters

Skip first two kernels …

… then profile one kernel

```
ncu \
    --kernel=myKernel --launch-skip=2 --launch-count=1 \
    my-app my-parameters
```

Also available as make target: `make ncu-cl`

Example outputs: `/profiles/`*`gpu`*`/`*`test`*`/`*`test-backend`*`-ncu-cl`

Further options including filtering by kernel name

https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html#profile

Nsight Compute

2. Basic usage + filters

Skip first two kernels …

… then profile one kernel

```
ncu \
    --kernel=myKernel --launch-skip=2 --launch-count=1 \
    my-app my-parameters
```

Live Demo

Also available as make target: `make ncu-cl`

Example outputs: /profiles/*gpu*/*test*/*test-backend*-ncu-cl

Further options including filtering by kernel name

https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html#profile

Nsight Compute

3. Basic usage + filters + specific metrics

```
ncu \
    --kernel=myKernel --launch-skip=2 --launch-count=1 \
    --metrics \
      dram__bytes_write.sum,dram__bytes_read.sum.per_second \
    my-app my-parameters
```

Also available as make target: `make ncu-metrics`

Example outputs: /profiles/*gpu*/*test*/*test-backend*-ncu-metrics

Nsight Compute

3. Basic usage + filters + specific metrics

Short summary over key metrics: `/docs/metrics.md`

Detailed documentation:

https://docs.nvidia.com/nsight-compute/ProfilingGuide/index.html#metrics-structure

Nsight Compute

3. Basic usage + filters + specific metrics

Live Demo

```
ncu \
    --kernel=myKernel --launch-skip=2 --launch-count=1 \
    --metrics \
      dram__bytes_write.sum,dram__bytes_read.sum.per_second \
    my-app my-parameters
```

Also available as make target: `make ncu-metrics`

Example outputs: /profiles/*gpu*/*test*/*test-backend*-ncu-metrics

Nsight Compute

4. Profile application on target machine, open resulting profile in GUI locally
   (Both machines may be the same)

```
ncu \
    -o my-profile --force-overwrite \
    --kernel=myKernel --launch-skip=2 --launch-count=1 \
    --set=full
    my-app my-parameters
```

Also available as make target: `make ncu-metrics`

Example outputs: /profiles/*gpu*/*test*/*test-backend*.ncu-rep

Nsight Compute

4. Profile application on target machine, open resulting profile in GUI locally (Both machines may be the same)

Customization is either done via sets or sections

```
ncu --list-sets            ncu --set=... ...
ncu --list-sections        ncu --section=... ...
```

Nsight Compute

4. Profile application on target machine, open resulting profile in GUI locally (Both machines may be the same)

```
ncu \
    -o my-profile --force-overwrite \
    --kernel=myKernel --launch-skip=2 --launch-count=1 \
    --set=full
    my-app my-parameters
```

Live Demo

Also available as make target: `make ncu-metrics`

Example outputs: /profiles/*gpu*/*test*/*test-backend*.ncu-rep

GPU Performance Engineering

GPU Architecture

Performance Modelling

Profiling Tools

Use Cases

Outlook

Use Case – Inefficient Stream Benchmark

- Code: `~/src/stream-slow`
- Issue: Bandwidth benchmark reports a bandwidth that is way to low

Use Case – Inefficient Stream Benchmark

- Code: `~/src/stream-slow`
- Issue: Bandwidth benchmark reports a bandwidth that is way to low
- Profiling:
  - Nsight compute shows reasonable performance numbers
  - Nsight systems reveals superfluous memory operations
- Identified cause: unnecessary PCIe transfers in measurement region
- Side note: this code version can be used to benchmark PCIe bandwidth

Use Case – Stencil Code

- Code: `~/src/stencil-2d`

- Issue: OpenMP target version is slower than a CUDA implementation

Use Case – Stencil Code

- Code: `~/src/stencil-2d`
- Issue: OpenMP target version is slower than a CUDA implementation
- Profiling: nsight compute shows additional L2 traffic
- Identified cause: additional memory traffic (1D vs 2D thread decomposition)

GPU Programming & Performance

GPU Architecture

Performance Modelling

Profiling Tools

Use Cases

Outlook

Outlook

- New HPC GPU systems with shared physical memory
  - NVIDIA GraceHopper super chip
  - AMD MI300A APU
  - Might require different ways of programming (for full performance)

Outlook

- Additional resources
  - Courses for European participants (note: shameless self-promotion)

    https://hpc.fau.de/teaching/tutorials-and-courses/

    https://www.nhr-verein.de/en/courses-and-workshops

  - Most high performance computing centers offer interesting courses

  - NVIDIA GTC conference – recordings and slides from past years are available

    https://www.nvidia.com/gtc/