GPUscout

Locating Data Movement-related Bottlenecks on GPUs

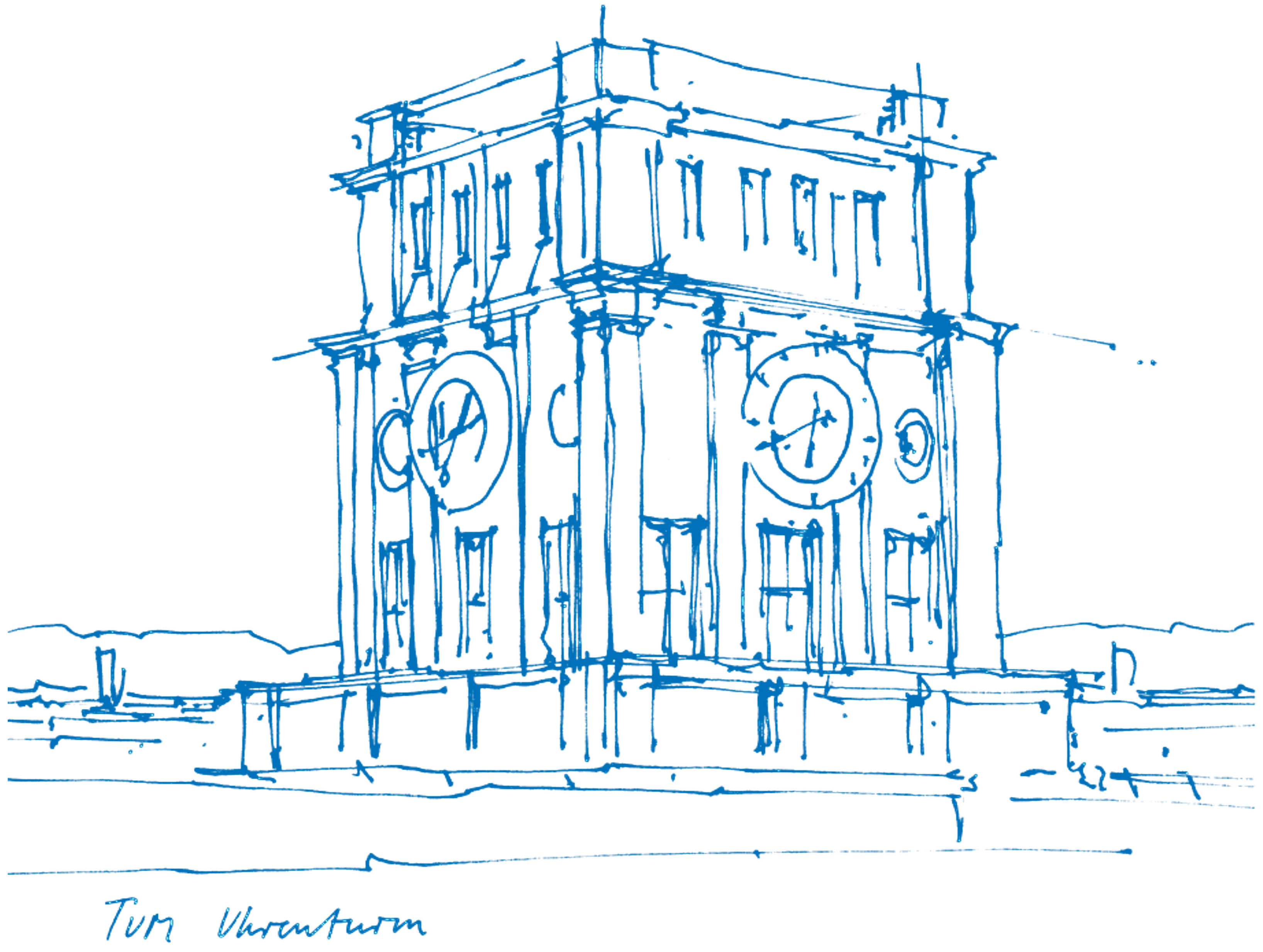
Soumya Sen
soumya.sen@tum.de

Stepan Vanecek
stepan.vanecek@tum.de

Martin Schulz
schulzm@in.tum.de

Chair of Computer Architecture and Parallel Systems
Technical University of Munich

ProTools’23, 12th November 2023



GPUscout

A new approach for locating data movement-related bottlenecks on NVidia GPUs

Combines **3 approaches**:

1. Static code analysis
2. Sampling PC stalls
3. Reading kernel-wide counters

Main objectives

- Scanning kernels for frequently-occurring data/memory-related bottlenecks,
- Providing information about the type and severity of the bottleneck,
- Pointing the user to the source code line.

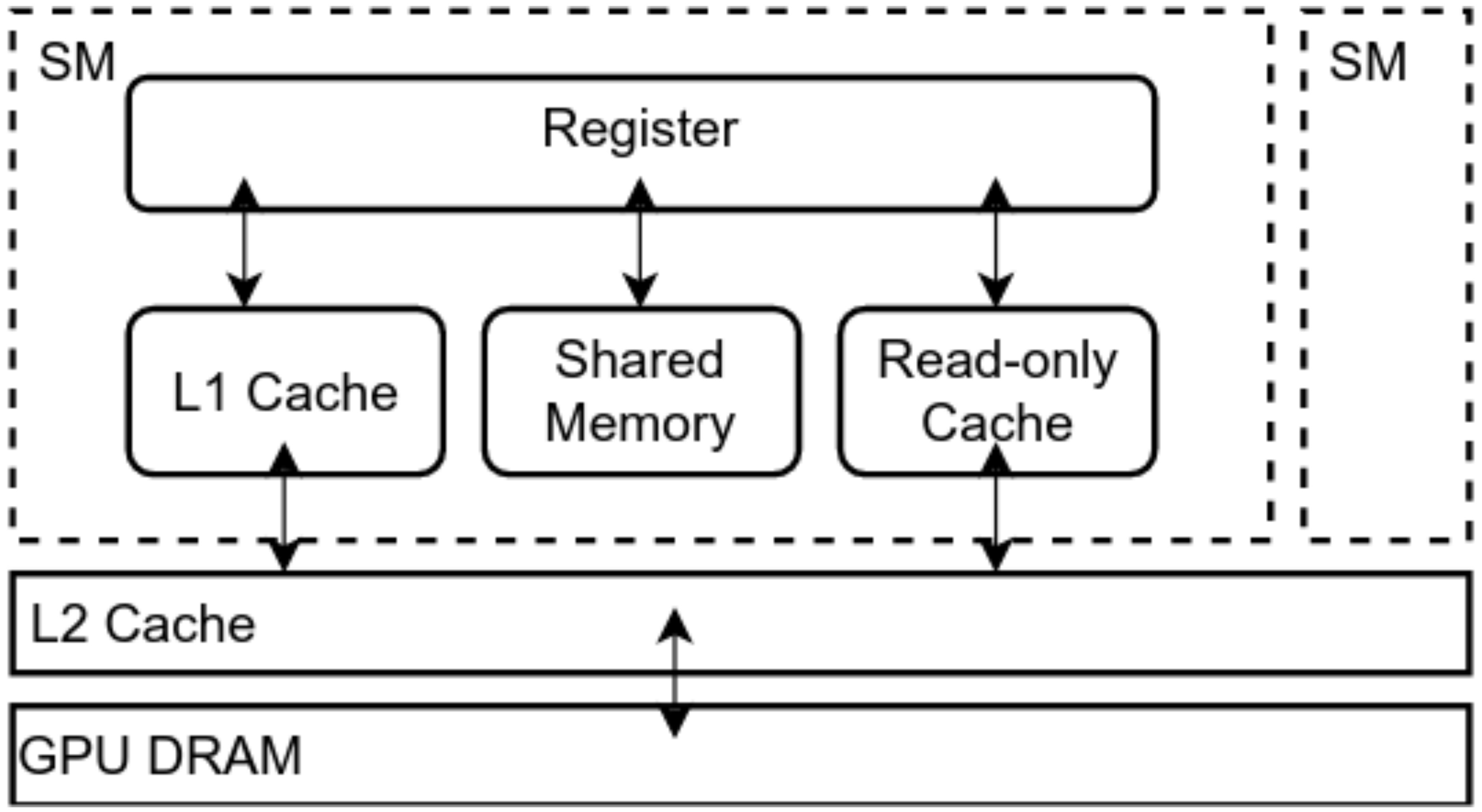
GPU Architecture

Very complex GPU architecture

Parallel design amplifies bottlenecks

The behaviour of GPU kernels is rather hidden

➡ Optimizing performance of GPU kernels is therefore a challenge



Existing Approaches

- Focus on analyzing **kernel-wide metrics**
- Provide finer-granularity data, however **without further guidance**

➡ We need a solution which

1. Discovers the problematic behaviour,
2. Points the user at the exact place in code where the problem originates,
3. Provides means to verify user’s improvements.

Background

What is SASS, Warp stalls, or NCU metrics?

2 assemblies in CUDA

1. PTX
2. SASS

CUPTI

- Provides data for profiling and tracing tools
- GPUscout uses the PC Sampling API of CUPTI (Warp stalls)
  - Stall reasons, line number

Nsight Compute CLI (ncu)

- Kernel-wide performance counters

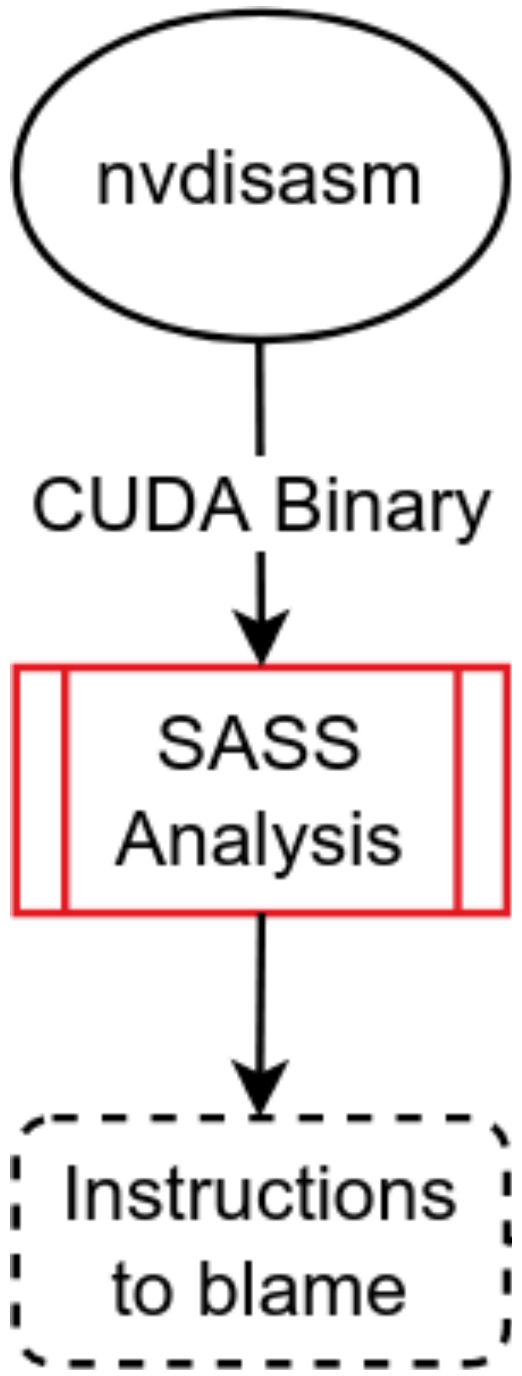
Bottlenecks Analysis

- **SASS analysis** at heart of GPUscout
    - Searching for specific code patterns
- **Warp stalls** for identified code line
- Kernel-wide **metrics** provide overview of data movements
- Additional metrics displayed for specific bottlenecks

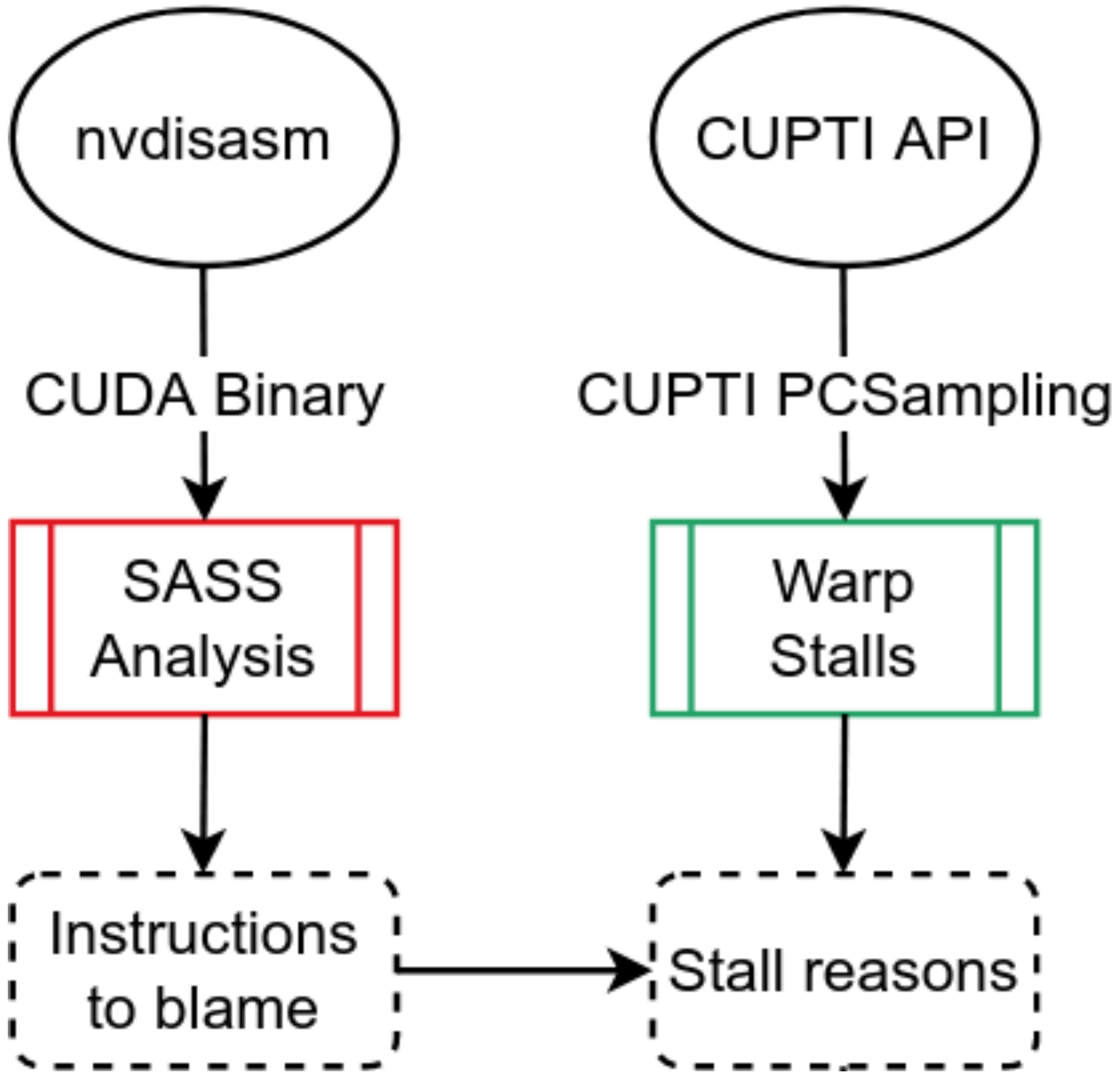
Analyses

1. Vectorized Loads
2. Register Spilling
3. Shared Memory
4. Shared Atomics
5. Read-only Cache
6. Texture Memory
7. Datatype Conversions

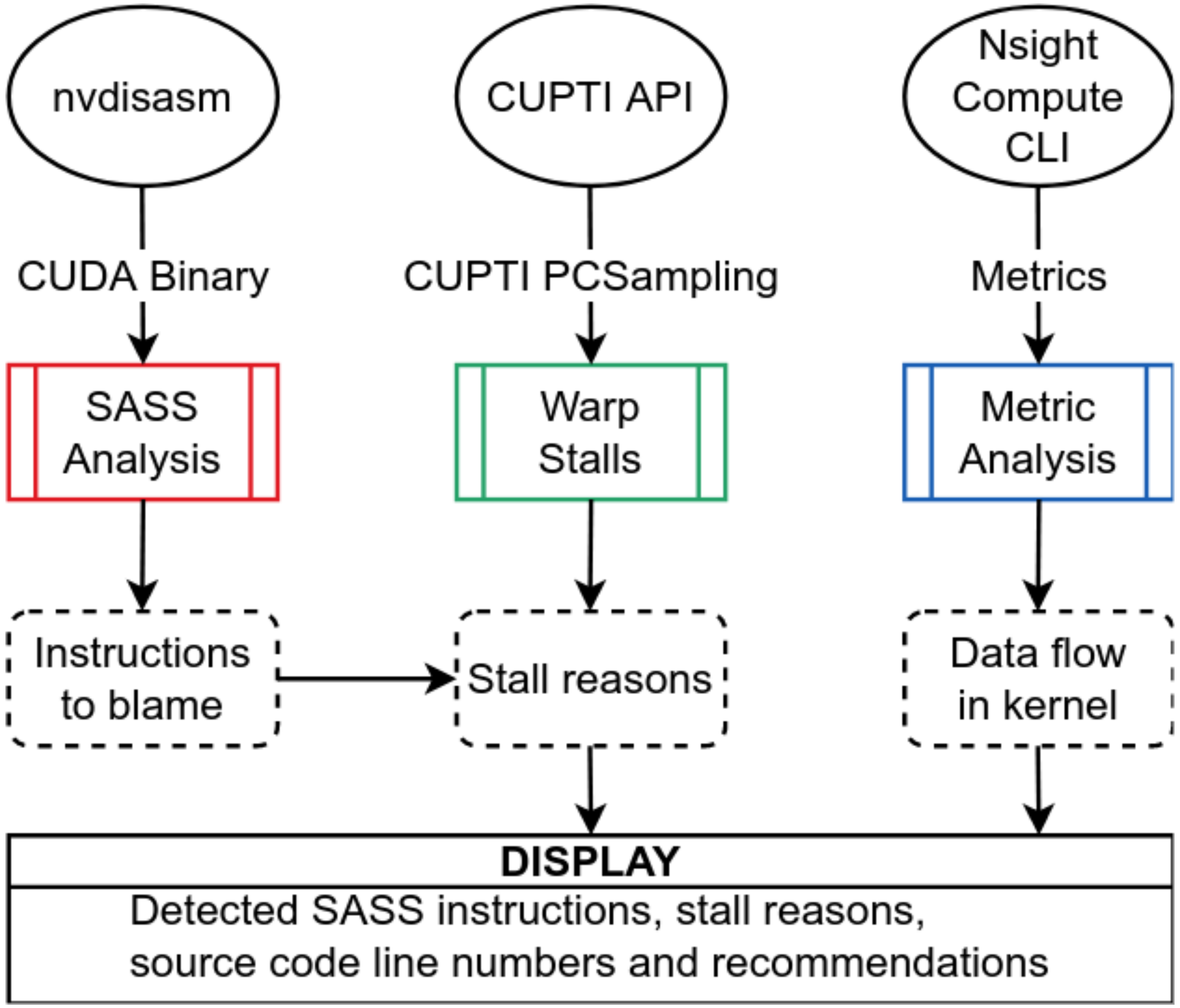
Architecture of GPUscout



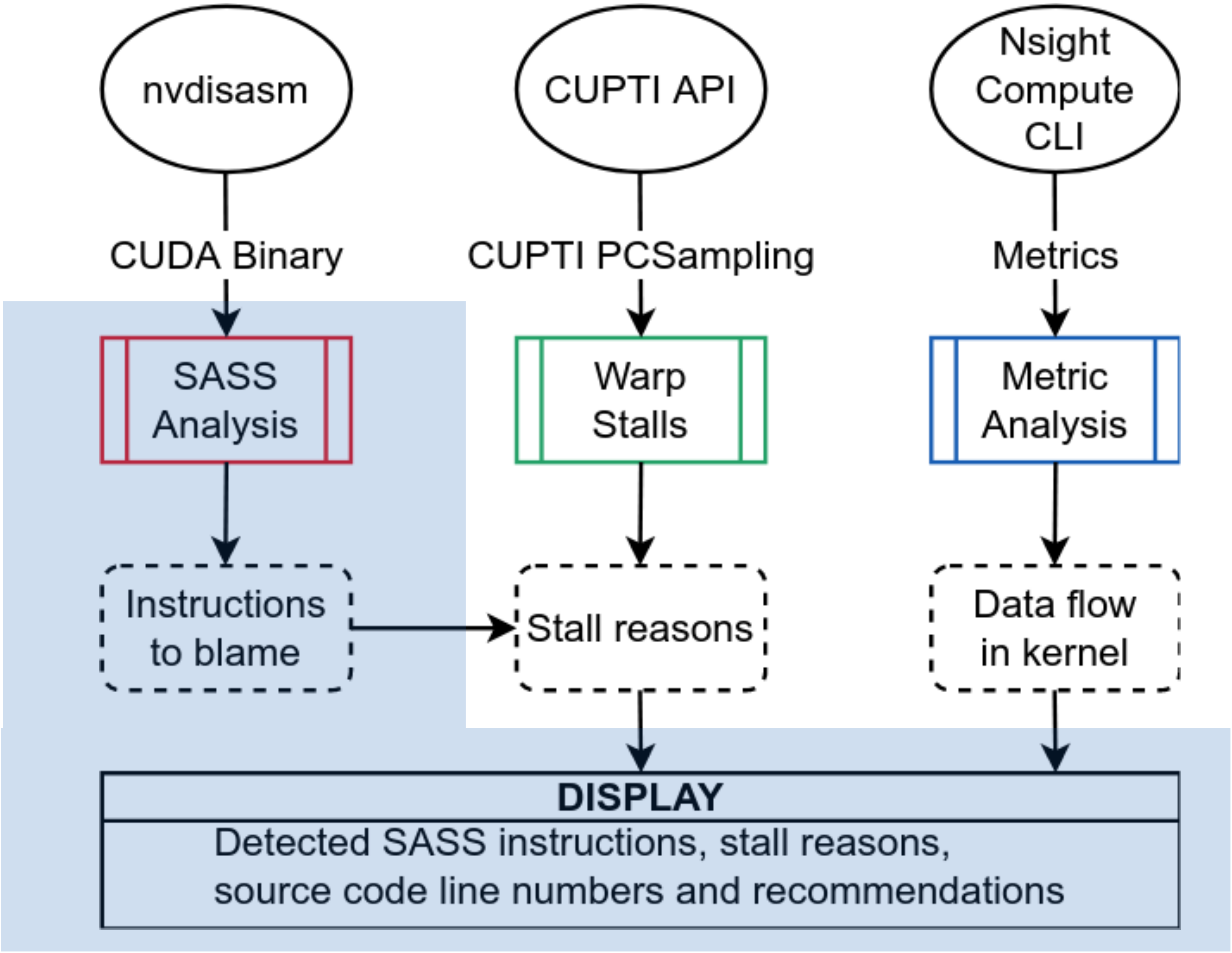
Architecture of GPUscout



Architecture of GPUscout



Architecture of GPUscout



GPUscout Analysis

Source code line number

Stall contribution

Kernel name (mangled)

Extra information related to the register

Bottleneck type

Detected bottleneck
SASS Analysis

```
-- Register spilling analysis for kernel: _Z3dotPiS_S_ --

WARNING  :: A spill detected in line number 144 of your code.
Register R19 spilled in LOAD operation
Previous instruction of R19 before spilling : IADD at line number 129 of code.
INFO  :: Total current registers for the SASS instruction: 17
```

Warp Stalls
CUPTI API

```
Stalls are detected with % of occurrence for the SASS instruction

stalled_lg_throttle (79.4996 %),          stalled_selected (12.6628 %),
stalled_not_selected (0.0255297 %),       stalled_mio_throttle (2.62956 %)
stalled_wait (5.18254 %)
```

Metric Analysis
Nsight Compute CLI

```
INFO  :: Data flow in memory for load operations

Local memory ---- request load data ----> L1 cache : 4.17837e+06 bytes
L1 Cache miss % (due to local memory load request) : 0.02
L1 cache ---- request load data ----> L2 cache : 835.674 bytes
L2 Cache miss % (due to L1 load data request) : 99.81
L2 cache ---- request load data ----> DRAM : 3.14027e+06 bytes

For register spilling, check Long Scoreboard stalls: 35.13 % per warp active
For register spilling, check LG Throttle stalls: 17.08 % per warp active
Percentage of total L2 queries due to local memory: 1.64476 %
```

Kernel-wide metrics

Mixbench

Use-case 1

Mixbench

Use-case 1

- Benchmarking suite for mixed operational intensity kernels
- CUDA implementation `mixbench-cuda`
- Executes MAD operations

<u>GPUscout analysis:</u>

1. Use Shared Memory
2. Use Vectorized Loads

Mixbench

Use-case 1

- Benchmarking suite for mixed operational intensity kernels
- CUDA implementation `mixbench-cuda`
- Executes MAD operations

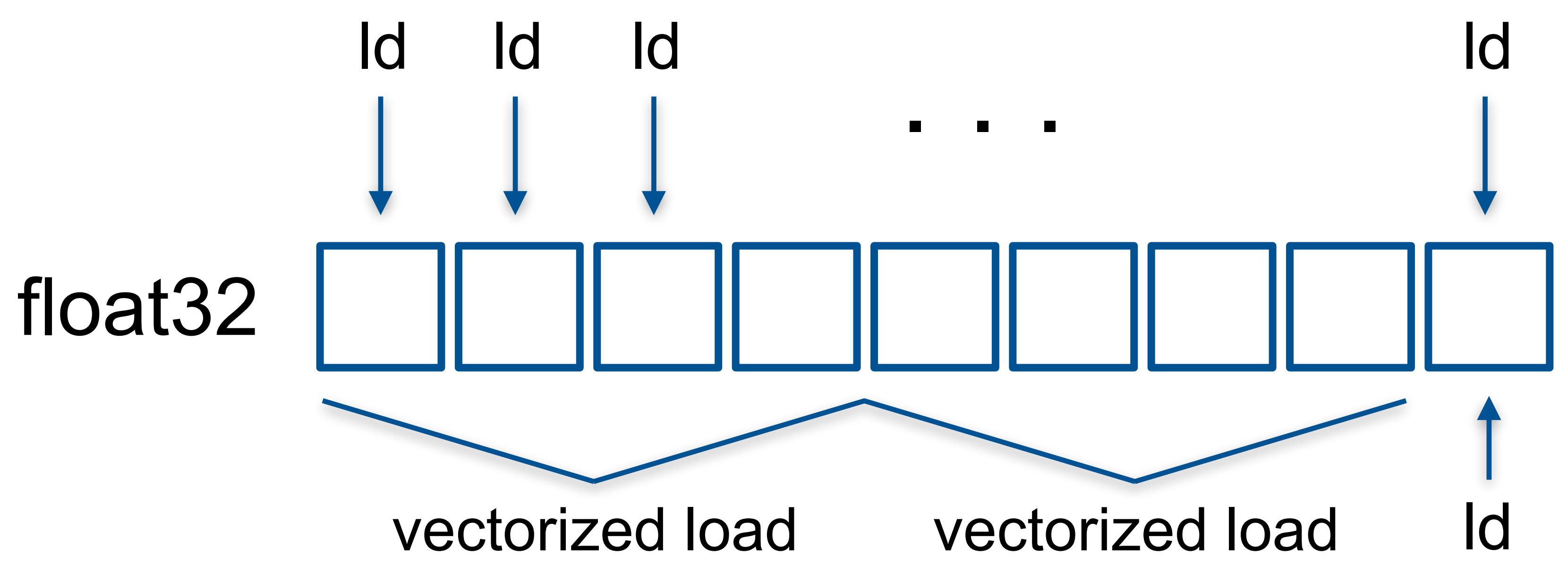
GPUscout analysis:

1. Use Shared Memory
2. Use Vectorized Loads

Mixbench

Use-case 1

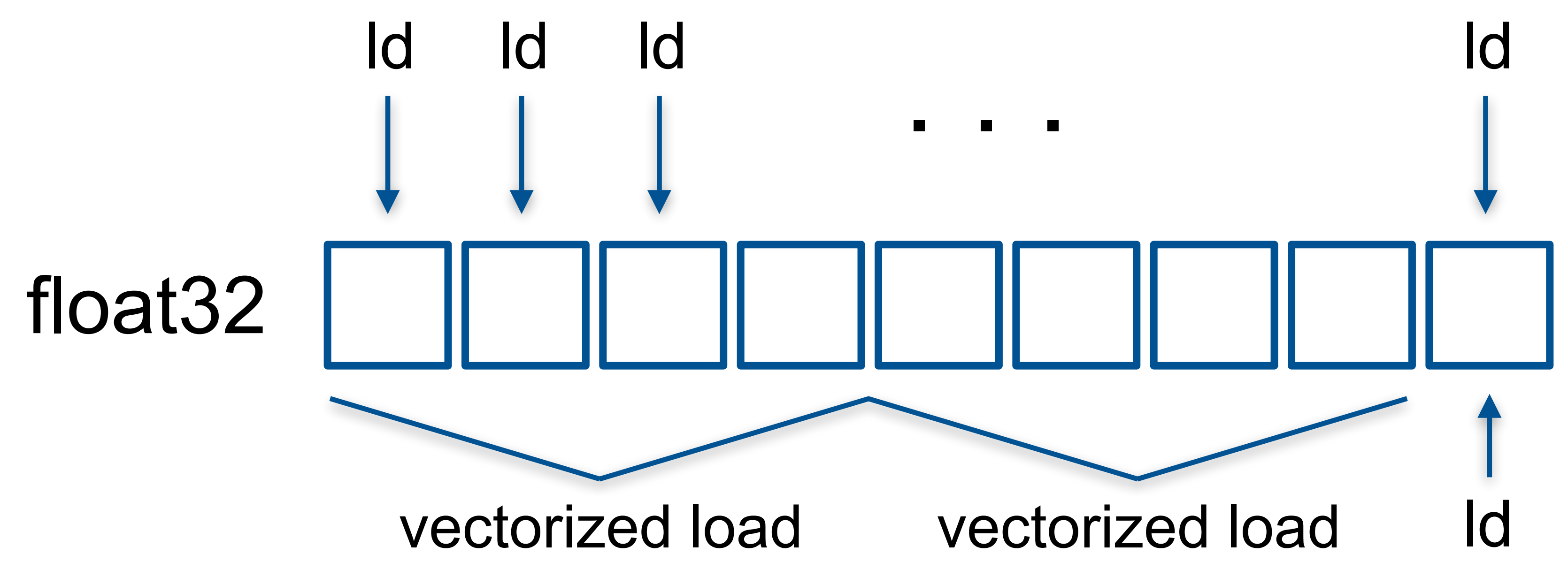
2. Use Vectorized Loads



Mixbench

Use-case 1

2. Use Vectorized Loads




```
for(int j=0; j<granularity; j++)
 tmps[j] = g_data[...];
 ...
 for(int i=0; i<compute_iterations; i++)
  tmps[j] = mad(tmps[j],tmps[j],seed);
```

```
for(int j=0; j<granularity/4; j++)
 reinterpret_cast<float4*>(tmps)[j] =
  reinterpret_cast<float4*>(g_data)[...];
 ...
 for(int i=0; i<compute_iterations; i++)
  reinterpret_cast<float4*>(tmps)[j] =
   mad(reinterpret_cast<float4*>(tmps)[j],
    reinterpret_cast<float4*>(tmps)[j],seed);
```

Mixbench

Use-case 1

2. Use Vectorized Loads

Warp stalls:

\+ Long scoreboard ↓ **62%** (originally 70%)

Metric Analysis:

– SM Occupancy ↓ **83%** (originally 92%)

➡ **Speedup of 3.77x**

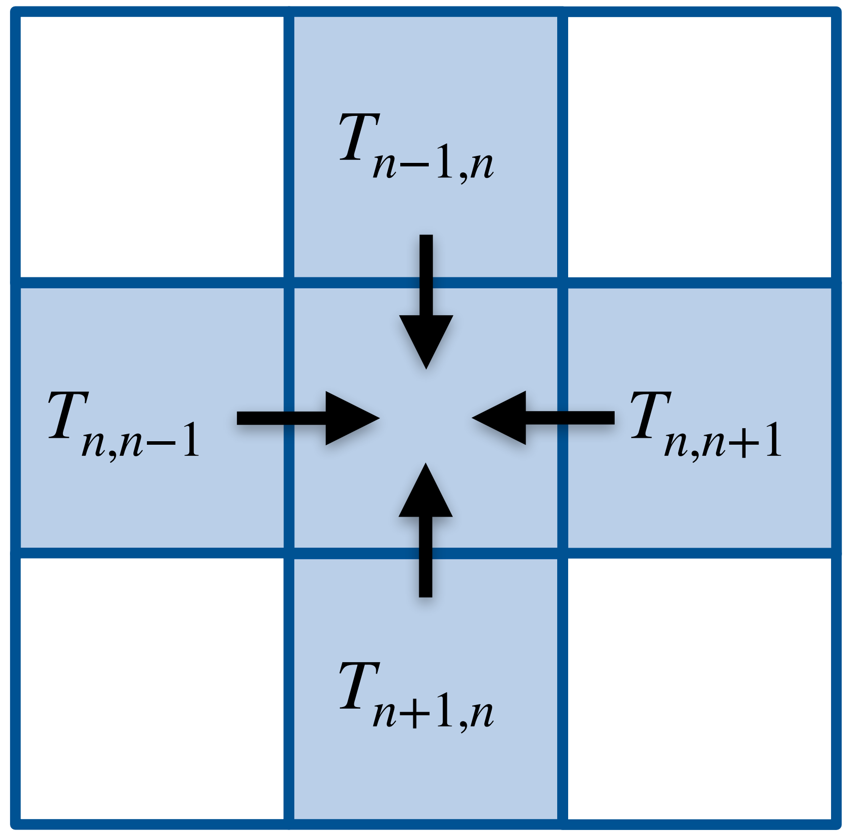
Heat Transfer Simulation

Use-case 2

Heat Transfer Simulation

Use-case 2

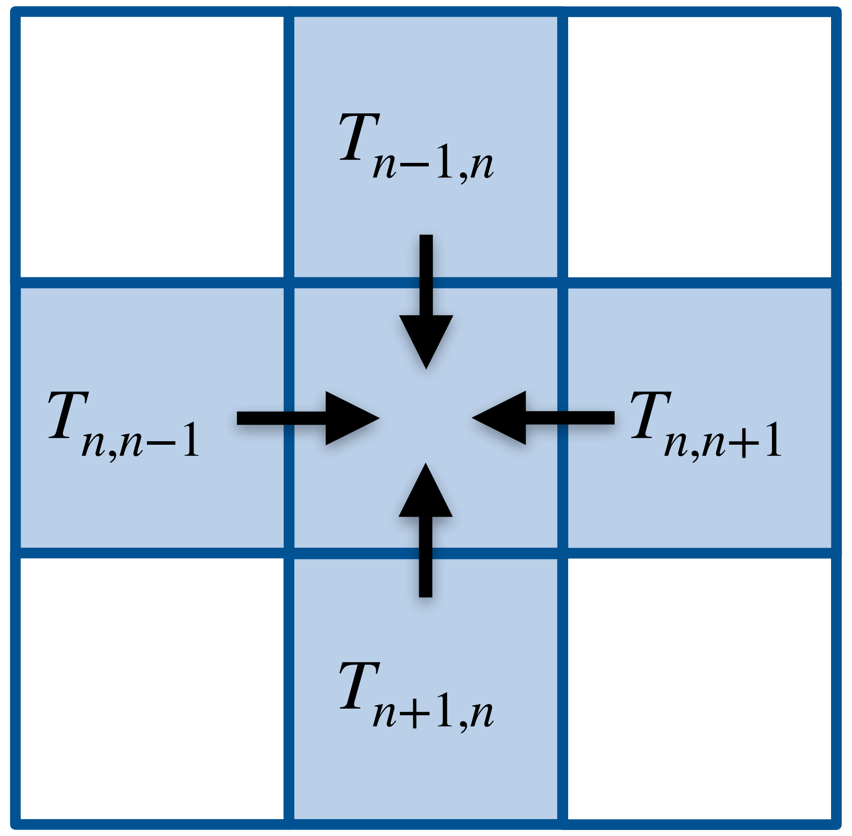
- Jacobi iterative solver for 2D
- $T_{NEW} = T_{OLD} + k * (T_{TOP} + T_{BOTTOM} + T_{LEFT} + T_{RIGHT} - 4 * T_{OLD})$
- GPUscout analysis:
  1. Use Texture Memory (or Use Shared memory)
  2. Use Vectorized Loads
  3. Using __restrict__ keyword
  4. Minimizing Datatype Conversions



Heat Transfer Simulation

Use-case 2

- Jacobi iterative solver for 2D
- $T_{NEW} = T_{OLD} + k * (T_{TOP} + T_{BOTTOM} + T_{LEFT} + T_{RIGHT} - 4 * T_{OLD})$
- GPUscout analysis:
  1. Use Texture Memory (or Use Shared memory)
  2. Use Vectorized Loads
  3. Using __restrict__ keyword
  4. Minimizing Datatype Conversions



Heat Transfer Simulation

Use-case 2

1. Use Texture Memory

```
== texture memory analysis for kernel: 2D-stencil-naive  ==
WARNING  ::  Use texture memory for register number (written-to): R4 at line
↪  number 6 of your code. The data is read from register number: R4
No spatial locality found for the register data
Stalls are detected with % of occurence for the SASS instruction
stalled_wait (66.6667 %), stalled_selected (33.3333 %)

WARNING  ::  Use texture memory for register number (written-to): R28 at line
↪  number 6 of your code. The data is read from register number: R2
Spatial locality found for the register data
Stalls are detected with % of occurence for the SASS instruction
stalled_wait (14.2857 %), stalled_lg_throttle (85.7143 %)

INFO  ::  Check data flow in texture memory, if you modify your code to use
↪  textures
Kernel ---- request load data ----> Texture Memory 0 instructions
Texture memory ---- request load data ----> L1 cache 0 bytes
L1 Cache miss % (due to texture memory load request) 100
L1 cache ---- request load data ----> L2 cache (due to texture memory load
↪  request) 0 bytes
L2 Cache miss % (due to L1 load data request) 23.89
L2 cache ---- request load data ----> DRAM 0 bytes
If using texture memory, check Tex Throttle: 0 %
If using texture memory, check Long Scoreboard: 37.79 %
```

Heat Transfer Simulation

Use-case 2

SASS Analysis

```
== texture memory analysis for kernel: 2D-stencil-naive  ==
WARNING  ::  Use texture memory for register number (written-to): R4 at line
↪  number 6 of your code. The data is read from register number: R4
No spatial locality found for the register data
Stalls are detected with % of occurence for the SASS instruction
stalled_wait (66.6667 %), stalled_selected (33.3333 %)

WARNING  ::  Use texture memory for register number (written-to): R28 at line
↪  number 6 of your code. The data is read from register number: R2
Spatial locality found for the register data
Stalls are detected with % of occurence for the SASS instruction
stalled_wait (14.2857 %), stalled_lg_throttle (85.7143 %)

INFO  ::  Check data flow in texture memory, if you modify your code to use
↪  textures
Kernel ---- request load data ----> Texture Memory 0 instructions
Texture memory ---- request load data ----> L1 cache 0 bytes
L1 Cache miss % (due to texture memory load request) 100
L1 cache ---- request load data ----> L2 cache (due to texture memory load
↪  request) 0 bytes
L2 Cache miss % (due to L1 load data request) 23.89
L2 cache ---- request load data ----> DRAM 0 bytes
If using texture memory, check Tex Throttle: 0 %
If using texture memory, check Long Scoreboard: 37.79 %
```

Heat Transfer Simulation

Use-case 2

SASS Analysis

Warp stalls

```
== texture memory analysis for kernel: 2D-stencil-naive  ==
WARNING  ::  Use texture memory for register number (written-to): R4 at line
↪  number 6 of your code. The data is read from register number: R4
No spatial locality found for the register data
Stalls are detected with % of occurence for the SASS instruction
stalled_wait (66.6667 %), stalled_selected (33.3333 %)

WARNING  ::  Use texture memory for register number (written-to): R28 at line
↪  number 6 of your code. The data is read from register number: R2
Spatial locality found for the register data
Stalls are detected with % of occurence for the SASS instruction
stalled_wait (14.2857 %), stalled_lg_throttle (85.7143 %)

INFO  ::  Check data flow in texture memory, if you modify your code to use
↪  textures
Kernel ---- request load data ----> Texture Memory 0 instructions
Texture memory ---- request load data ----> L1 cache 0 bytes
L1 Cache miss % (due to texture memory load request) 100
L1 cache ---- request load data ----> L2 cache (due to texture memory load
↪  request) 0 bytes
L2 Cache miss % (due to L1 load data request) 23.89
L2 cache ---- request load data ----> DRAM 0 bytes
If using texture memory, check Tex Throttle: 0 %
If using texture memory, check Long Scoreboard: 37.79 %
```

Heat Transfer Simulation

Use-case 2

SASS Analysis

Warp stalls

Kernel Metrics

```
== texture memory analysis for kernel: 2D-stencil-naive  ==
WARNING  ::  Use texture memory for register number (written-to): R4 at line
↪  number 6 of your code. The data is read from register number: R4
No spatial locality found for the register data
Stalls are detected with % of occurence for the SASS instruction
stalled_wait (66.6667 %), stalled_selected (33.3333 %)
```

```
WARNING  ::  Use texture memory for register number (written-to): R28 at line
↪  number 6 of your code. The data is read from register number: R2
Spatial locality found for the register data
Stalls are detected with % of occurence for the SASS instruction
stalled_wait (14.2857 %), stalled_lg_throttle (85.7143 %)
```

```
INFO  ::  Check data flow in texture memory, if you modify your code to use
↪  textures
Kernel ---- request load data ----> Texture Memory 0 instructions
Texture memory ---- request load data ----> L1 cache 0 bytes
L1 Cache miss % (due to texture memory load request) 100
L1 cache ---- request load data ----> L2 cache (due to texture memory load
↪  request) 0 bytes
L2 Cache miss % (due to L1 load data request) 23.89
L2 cache ---- request load data ----> DRAM 0 bytes
If using texture memory, check Tex Throttle: 0 %
If using texture memory, check Long Scoreboard: 37.79 %
```

Heat Transfer Simulation

Use-case 2

```
== texture memory analysis for kernel: 2D-stencil-naive  ==
WARNING  ::  Use texture memory for register number (written-to): R4 at line
↪  number 6 of your code. The data is read from register number: R4
No spatial locality found for the register data
Stalls are detected with % of occurence for the SASS instruction
stalled_wait (66.6667 %), stalled_selected (33.3333 %)

WARNING  ::  Use texture memory for register number (written-to): R28 at line
↪  number 6 of your code. The data is read from register number: R2
Spatial locality found for the register data
Stalls are detected with % of occurence for the SASS instruction
stalled_wait (14.2857 %), stalled_lg_throttle (85.7143 %)
```

⋮

Heat Transfer Simulation

Use-case 2

```
== texture memory analysis for kernel: 2D-stencil-naive ==
WARNING :: Use texture memory for register number (written-to): R4 at line
↪ number 6 of your code. The data is read from register number: R4
No spatial locality found for the register data
Stalls are detected with % of occurence for the SASS instruction
stalled_wait (66.6667 %), stalled_selected (33.3333 %)

WARNING :: Use texture memory for register number (written-to): R28 at line
↪ number 6 of your code. The data is read from register number: R2
Spatial locality found for the register data
Stalls are detected with % of occurence for the SASS instruction
stalled_wait (14.2857 %), stalled_lg_throttle (85.7143 %)
```

⋮

Heat Transfer Simulation

Use-case 2

```
== texture memory analysis for kernel: 2D-stencil-naive  ==
WARNING  ::  Use texture memory for register number (written-to): R4 at line
↪  number 6 of your code. The data is read from register number: R4
No spatial locality found for the register data
Stalls are detected with % of occurence for the SASS instruction
stalled_wait (66.6667 %), stalled_selected (33.3333 %)

WARNING  ::  Use texture memory for register number (written-to): R28 at line
↪  number 6 of your code. The data is read from register number: R2
Spatial locality found for the register data
Stalls are detected with % of occurence for the SASS instruction
stalled_wait (14.2857 %), stalled_lg_throttle (85.7143 %)
```

stalled_wait: Warp stalled waiting for a execution dependency of a fixed-latency instruction. Caused mostly because of an already highly optimized kernel.

stalled_lg_throttle: Warp stalled waiting for the L1 instruction queue for local and global (LG) memory operations. Caused mostly because of executing local or global memory instructions too frequently.

Heat Transfer Simulation

Use-case 2

⋮

```
INFO  ::  Check data flow in texture memory, if you modify your code to use
↪  textures
Kernel ---- request load data ----> Texture Memory 0 instructions
Texture memory ---- request load data ----> L1 cache 0 bytes
L1 Cache miss % (due to texture memory load request) 100
L1 cache ---- request load data ----> L2 cache (due to texture memory load
↪  request) 0 bytes
L2 Cache miss % (due to L1 load data request) 23.89
L2 cache ---- request load data ----> DRAM 0 bytes
If using texture memory, check Tex Throttle: 0 %
If using texture memory, check Long Scoreboard: 37.79 %
```

Heat Transfer Simulation

Use-case 2

⋮

```
INFO  ::  Check data flow in texture memory, if you modify your code to use
↪  textures
Kernel ---- request load data ----> Texture Memory 0 instructions
Texture memory ---- request load data ----> L1 cache 0 bytes
L1 Cache miss % (due to texture memory load request) 100
L1 cache ---- request load data ----> L2 cache (due to texture memory load
↪  request) 0 bytes
L2 Cache miss % (due to L1 load data request) 23.89
L2 cache ---- request load data ----> DRAM 0 bytes
If using texture memory, check Tex Throttle: 0 %
If using texture memory, check Long Scoreboard: 37.79 %
```

Heat Transfer Simulation

Use-case 2

- Jacobi iterative solver for 2D
- $T_{NEW} = T_{OLD} + k * (T_{TOP} + T_{BOTTOM} + T_{LEFT} + T_{RIGHT} - 4 * T_{OLD})$
- <u>GPUscout analysis:</u>
  1. Use Texture Memory

Heat Transfer Simulation

Use-case 2

- Jacobi iterative solver for 2D
- $T_{NEW} = T_{OLD} + k * (T_{TOP} + T_{BOTTOM} + T_{LEFT} + T_{RIGHT} - 4 * T_{OLD})$
- GPUscout analysis:

1. Use Texture Memory

    Warp Stalls:

    – TEX throttle ↑ **25%** (originally 0%)

    + Long Scoreboard ↓ **27%** (originally 38%)

    Metric Analysis:

    + Throughput ↑ **61%**

➡ **Performance improvement of 39.2%**

Heat Transfer Simulation

Use-case 2

- Jacobi iterative solver for 2D
- $T_{NEW} = T_{OLD} + k * (T_{TOP} + T_{BOTTOM} + T_{LEFT} + T_{RIGHT} - 4 * T_{OLD})$
- GPUscout analysis:

3. Using __restrict__ keyword

Heat Transfer Simulation

Use-case 2

- Jacobi iterative solver for 2D
- $T_{NEW} = T_{OLD} + k * (T_{TOP} + T_{BOTTOM} + T_{LEFT} + T_{RIGHT} - 4 * T_{OLD})$
- GPUscout analysis:

3. Using __restrict__ keyword

➡ Performance improvement of only **0.3%**

Heat Transfer Simulation

Use-case 2

- Jacobi iterative solver for 2D
- $T_{NEW} = T_{OLD} + k * (T_{TOP} + T_{BOTTOM} + T_{LEFT} + T_{RIGHT} - 4 * T_{OLD})$
- GPUscout analysis:

4. Minimizing Datatype Conversions

Heat Transfer Simulation

Use-case 2

- Jacobi iterative solver for 2D
- $T_{NEW} = T_{OLD} + k * (T_{TOP} + T_{BOTTOM} + T_{LEFT} + T_{RIGHT} - 4 * T_{OLD})$
- GPUscout analysis:

4. Minimizing Datatype Conversions
    - Impossible to avoid

GPUscout

- A new tool focussing on detecting memory-based bottlenecks on NVidia GPU kernels
- Builds on SASS analysis, sampling Warp Stalls, and providing additional kernel-wide metrics
- Points the user directly at the potentially problematic code line and provides additional information
- GPUscout recommendations bring a speedup of 3.77x and 1.64x on presented kernels

Try out GPUscout and get in touch with us!

https://github.com/caps-tum/GPUscout

stepan.vanecek@tum.de

Acknowledgements

SPONSORED BY THE

Federal Ministry of Education and Research

DEEP-SEA
EU Grant #955606
BMBF #16HPC014

DEEP-SEA