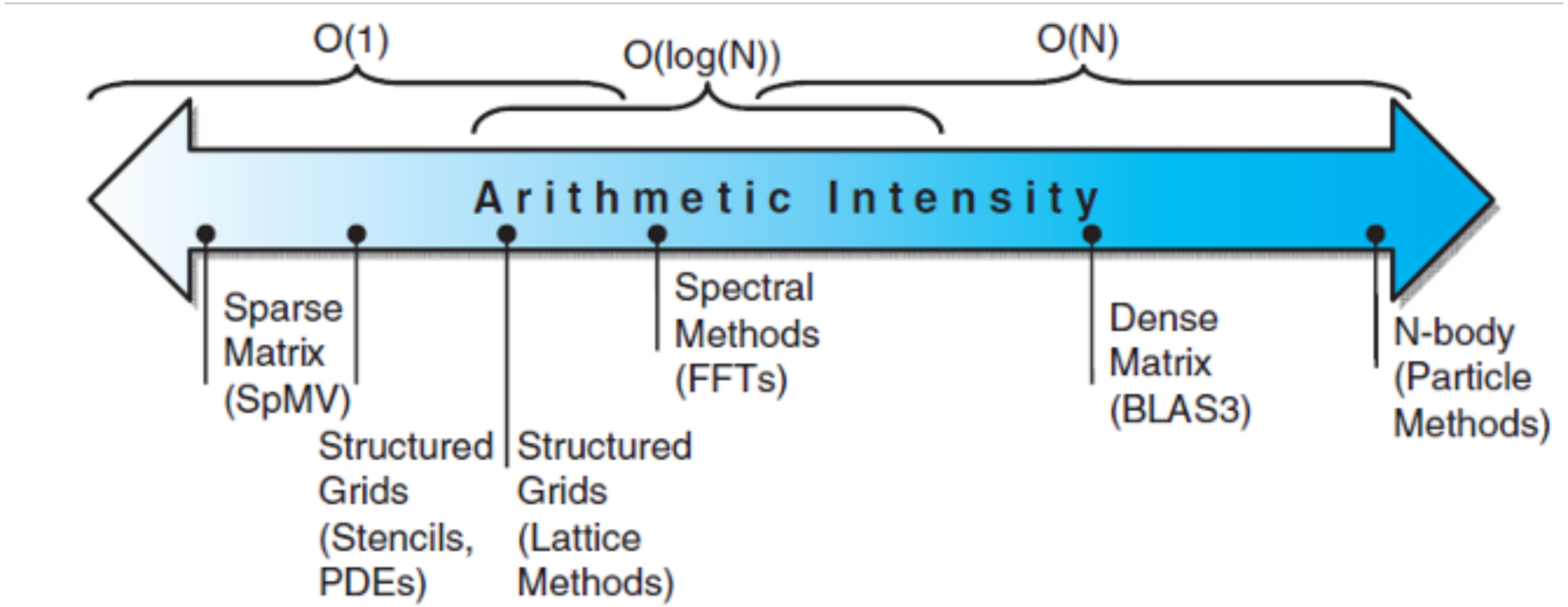
GPU COMPUTING
LECTURE 06 - PROFILING

Kazem Shekofteh
Kazem.shekofteh@ziti.uni-heidelberg.de
Institute of Computer Engineering
Ruprecht-Karls University of Heidelberg
Inspired from lectures by Holger Fröning

UNDERSTANDING PERFORMANCE

"There is no lower bound how bad a baseline can be."

UNDERSTANDING PERFORMANCE



Arithmetic intensity r of an application: FLOPs (or OPs) per byte of memory accessed

$$r = \frac{\text{FLOPs}}{\text{Byte}}$$

ROOFLINE MODEL

For a given processor

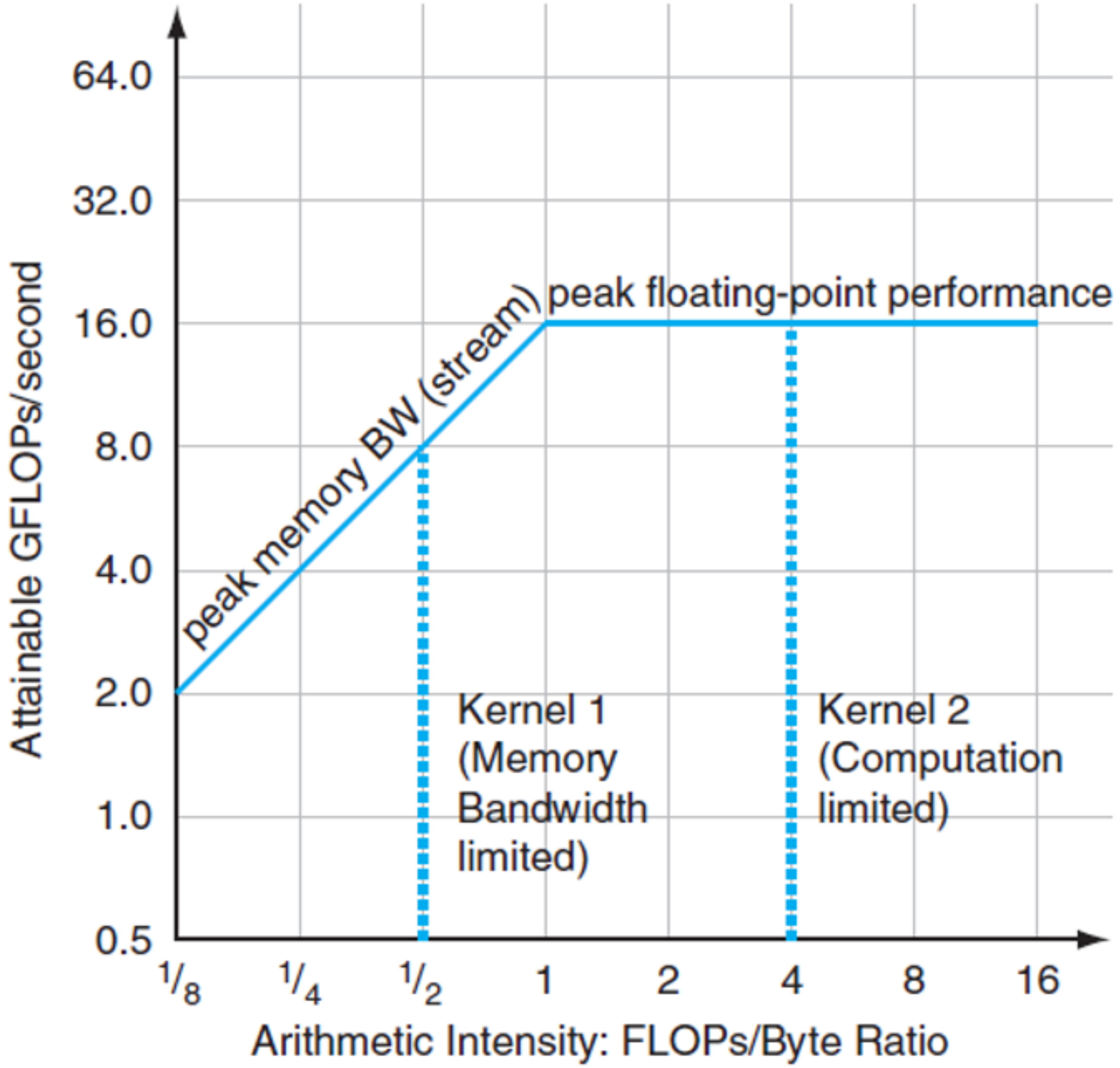
- Determine peak compute performance (GFLOP/s) (= `f`)
- Determine peak memory performance (GB/s) (= `m`)

Slope-intercept form $y = w \cdot x + z$

Attainable GFLOP/s performance `a` is then $a = min(m \cdot r, f)$

Boundness determines target metric

Boundness determines choice of optimizations



Patterson, Hennessy: Computer Organization and Design RISC-V Edition: The Hardware Software Interface, MK, 2017

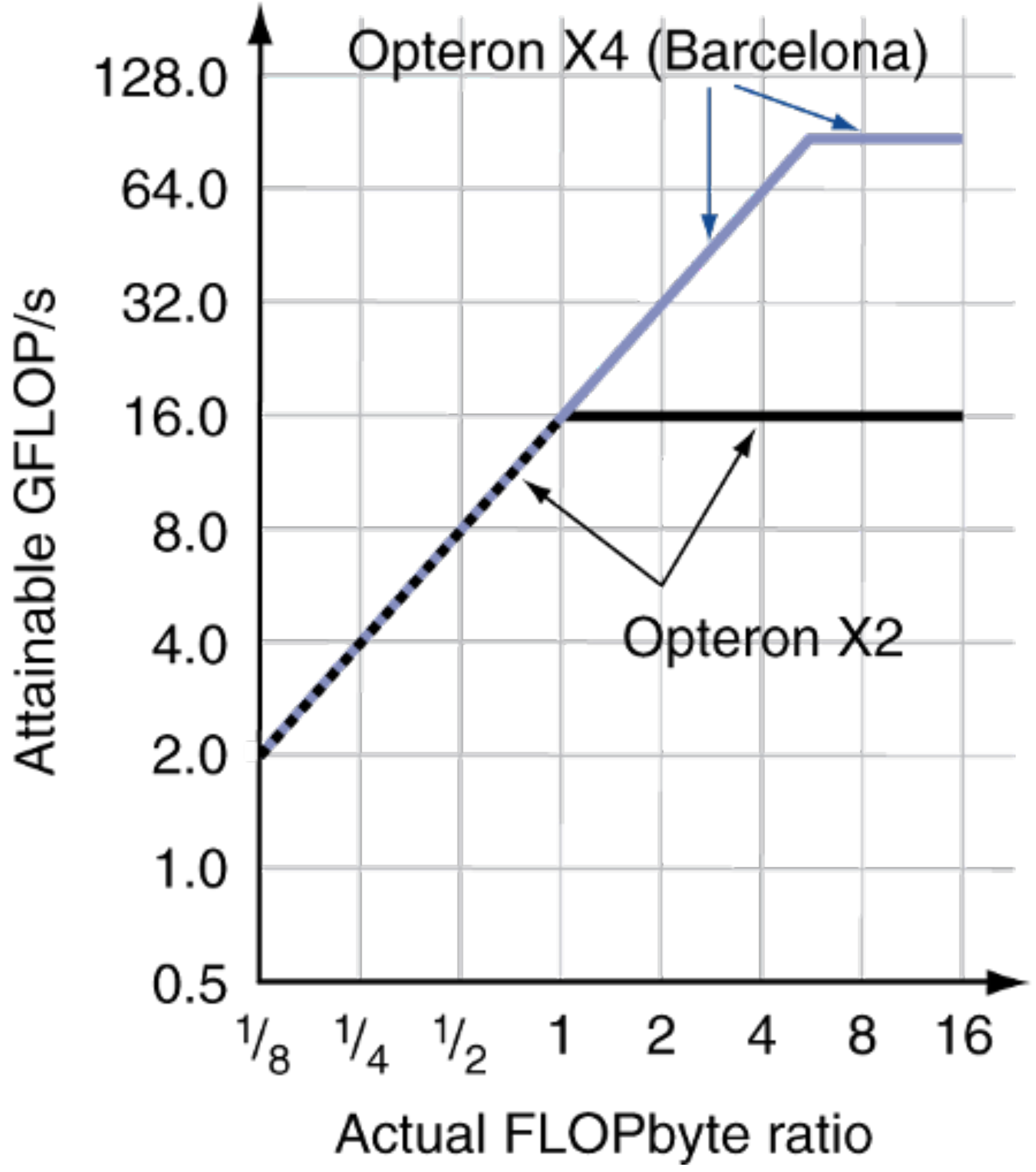
COMPARING SYSTEMS

Example: Opteron X2 vs. Opteron X4

- 2-core vs. 4-core, 2× FP performance/ core, 2.2GHz vs. 2.3GHz
- Same memory system

To get higher performance on X4 than X2

- Need high arithmetic intensity
- Or working set must fit in X4's 2MB L-3 cache



Patterson, Hennessy: Computer Organization and Design RISC-V Edition: The Hardware Software Interface, MK, 2017

OPTIMIZING PERFORMANCE

Optimize FP performance

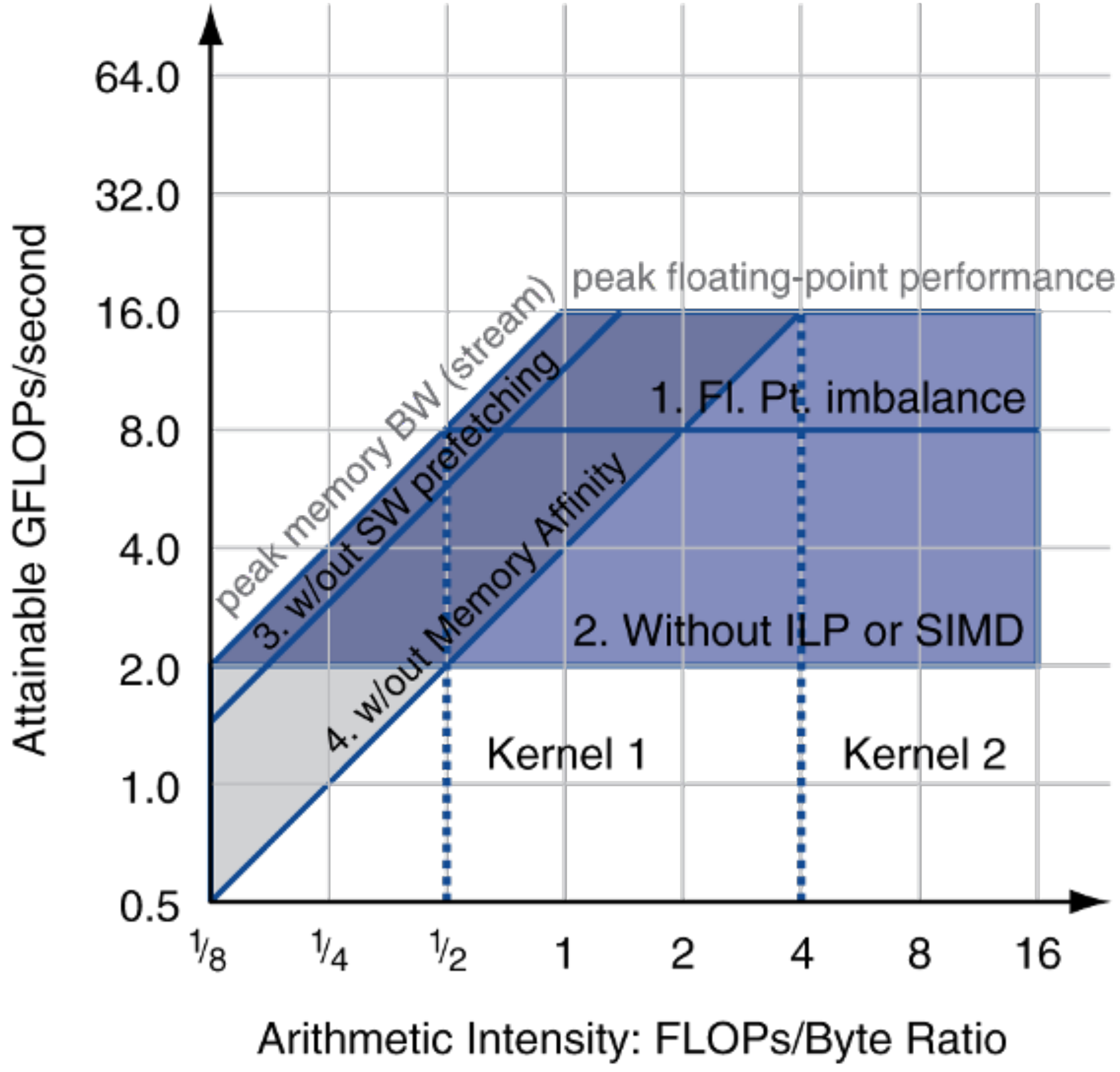
- Balance adds & multiplies
- Improve superscalar ILP
- Use of SIMD instructions

Optimize memory usage

- Software prefetch
- Avoid load stalls
- Memory affinity
- Avoid non-local data accesses

Optimization depends on `r`, but `r` can vary

- May scale with problem size
- Caching reduces memory accesses => increases arithmetic intensity



Patterson, Hennessy: Computer Organization and Design RISC-V Edition: The Hardware Software Interface, MK, 2017

ALGORITHMS CAN BE DIVIDED INTO THREE CLASSES

Memory-bound: limited in performance by access to memory

- Algorithm includes plenty of memory accesses, but for each memory access only few calculations are performed
- Execution time dominated by memory accesses

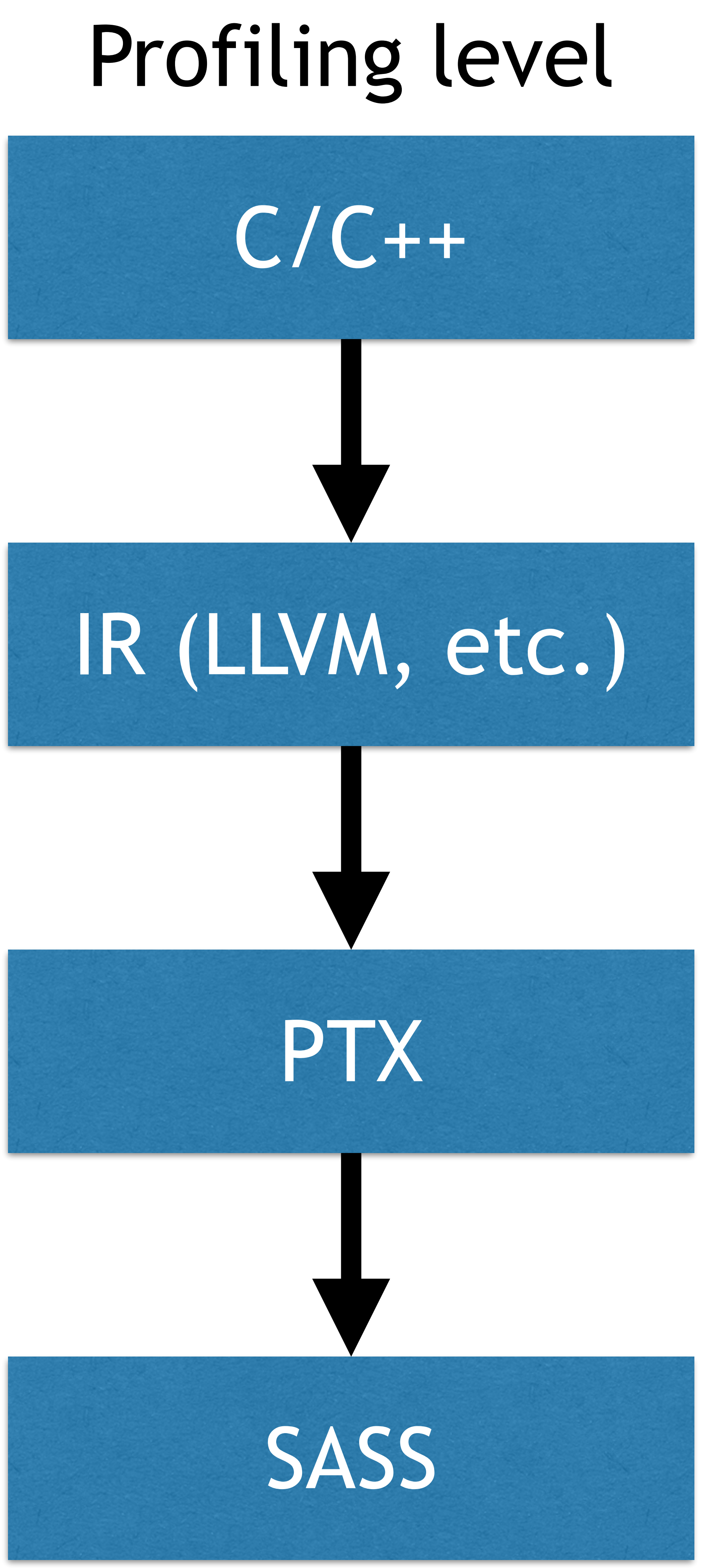
Compute-bound: limited in performance by computations

- Algorithm includes plenty of integer and floating point operations; for each memory access many calculations are performed
- Execution time dominated by computations

IO-bound: limited in performance by IO operations

- Usually disk or network access
- In the context of GPUs: PCIe bottleneck affecting host-device data movements

PROFILING AT SASS LEVEL



UNDERSTANDING GPU PERFORMANCE

Profiling: understanding application behavior in terms of static and dynamic behavior

- Static: instruction count, possibly separated for different classes
- Dynamic: cache behavior, scheduling, occupancy, memory stalls

Hardware performance counters: expensive resource, limited in capacity, costly in access => profiling will affect the performance of your code

Ensure:

- Correctness before profiling, e.g., `cuda-memcheck` for segmentation faults and memory leaks
- Compiler optimizations (`nvcc -O2 ...`)
- Debug information (`nvcc -lineinfo ...`)

NSIGHT COMPUTE

Records and analyzes kernel performance metrics

- Pretty detailed: ~1000 metrics

Two user interfaces

- Command line interface (CLI): `ncu`
- GUI: `nv-nsight-cu`

Recording and analyzing can be separated

- Record into file using `ncu`, download for local use with `nv-nsight-cu`
- `ncu` results are printed to stdout by default, use `--export/-o` to save results to a report file (`.ncu-rep`)

METRICS

List all metrics

`ncu —query-metrics <—chip tu102>` (`wc -l` reports 1687 lines :/)

Better use sets …

`ncu —list-sets`

… or custom combinations of sets, sections, and metrics

```
ncu --set default --section SourceCounters --metrics
sm__sass_inst_executed_op_shared <app>
```

```
$ ncu --list-sets
---------- --------------------------------------------------------------------------- ------- -----------------
Identifier Sections                                                                    Enabled Estimated Metrics
---------- --------------------------------------------------------------------------- ------- -----------------
default    LaunchStats, Occupancy, SpeedOfLight                                        yes     36
detailed   ComputeWorkloadAnalysis, InstructionStats, LaunchStats, MemoryWorkloadAnaly no      172
           sis, Occupancy, SchedulerStats, SourceCounters, SpeedOfLight, SpeedOfLight_
           RooflineChart, WarpStateStats
full       ComputeWorkloadAnalysis, InstructionStats, LaunchStats, MemoryWorkloadAnaly no      177
           sis, MemoryWorkloadAnalysis_Chart, MemoryWorkloadAnalysis_Tables, Nvlink_Ta
           bles, Nvlink_Topology, Occupancy, SchedulerStats, SourceCounters, SpeedOfLi
           ght, SpeedOfLight_RooflineChart, WarpStateStats
source     SourceCounters                                                              no      58
```

NSIGHT SYSTEM

Records and analyzes system performance metrics

- In particular, CPU-GPU interactions

Host code annotations to mark code for later reference

- `#include <nvToolsExt.h>` and link with `-lnvToolsExt`

Two user interfaces

- Command line interface (CLI): `nsys`
- GUI: `nsight-sys`

Recording and analyzing can be separated

- Record into file using `nsys profile <app>`, download for local use

```
...
nvtxRangePush(“sleeping”);
sleep(100);
nvtxRangePop();
...
```

EXAMPLE: PROFILING MATRIX MULTIPLY

NCU PROFILING

```
$ module load nvhpc/21.9

$ ./cuBLAS-test-sm75 1024 1024 1024
SGEMM (  1024 x   1024 x   1024):     0.0002 sec,    8363.55 GFLOP/s

$ ncu -f --set default -o <file> ./cuBLAS-test-sm75 1024 1024 1024
<snip>
==PROF== Profiling "volta_sgemm_128x64_nn" - 2: 0%....50%....100% - 8 passes
SGEMM (  1024 x   1024 x   1024):     0.5779 sec,       3.46 GFLOP/s
<snip>

$ ncu -f --set full --section ComputeWorkloadAnalysis -o <file> ./cuBLAS-test-
sm75 1024 1024 1024
<snip>
==PROF== Profiling "volta_sgemm_128x64_nn" - 2: 0%....50%....100% - 33 passes
SGEMM (  1024 x   1024 x   1024):     1.7117 sec,       1.17 GFLOP/s
<snip>
```

"SPEED OF LIGHT" ANALYSIS

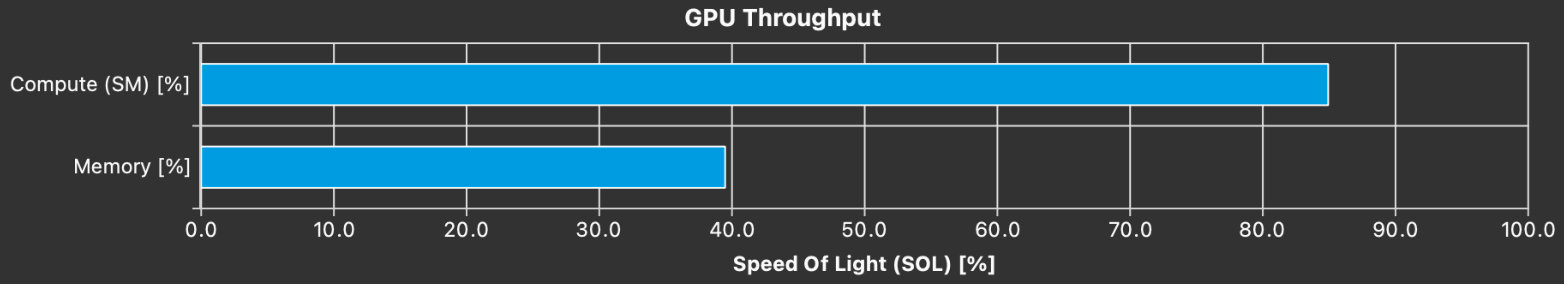
GPU Speed Of Light Throughput All

High-level overview of the throughput for compute and memory resources of the GPU.For each unit,the throughput reports the achieved percentage of utilization with respect to the theoretical maximum.Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor.High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

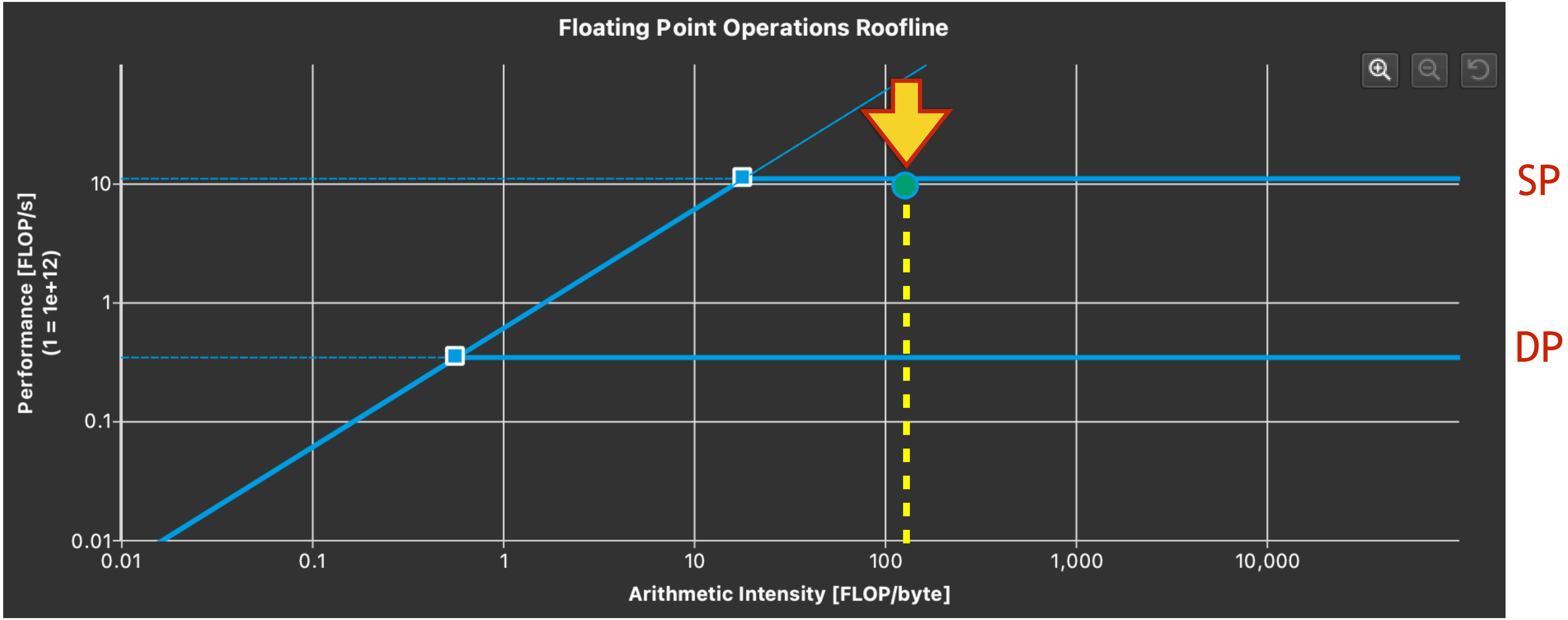
Compute (SM) Throughput [%]	84.93	Duration [usecond]	228.10
Memory Throughput [%]	39.48	Elapsed Cycles [cycle]	292771
L1/TEX Cache Throughput [%]	71.13	SM Active Cycles [cycle]	268708.31
L2 Cache Throughput [%]	29.00	SM Frequency [cycle/nsecond]	1.28
DRAM Throughput [%]	11.89	DRAM Frequency [cycle/nsecond]	6.39

High Throughput The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device.To further improve performance,work will likely need to be shifted from the most utilized to another unit.Start by analyzing workloads in the Compute Workload Analysis section.

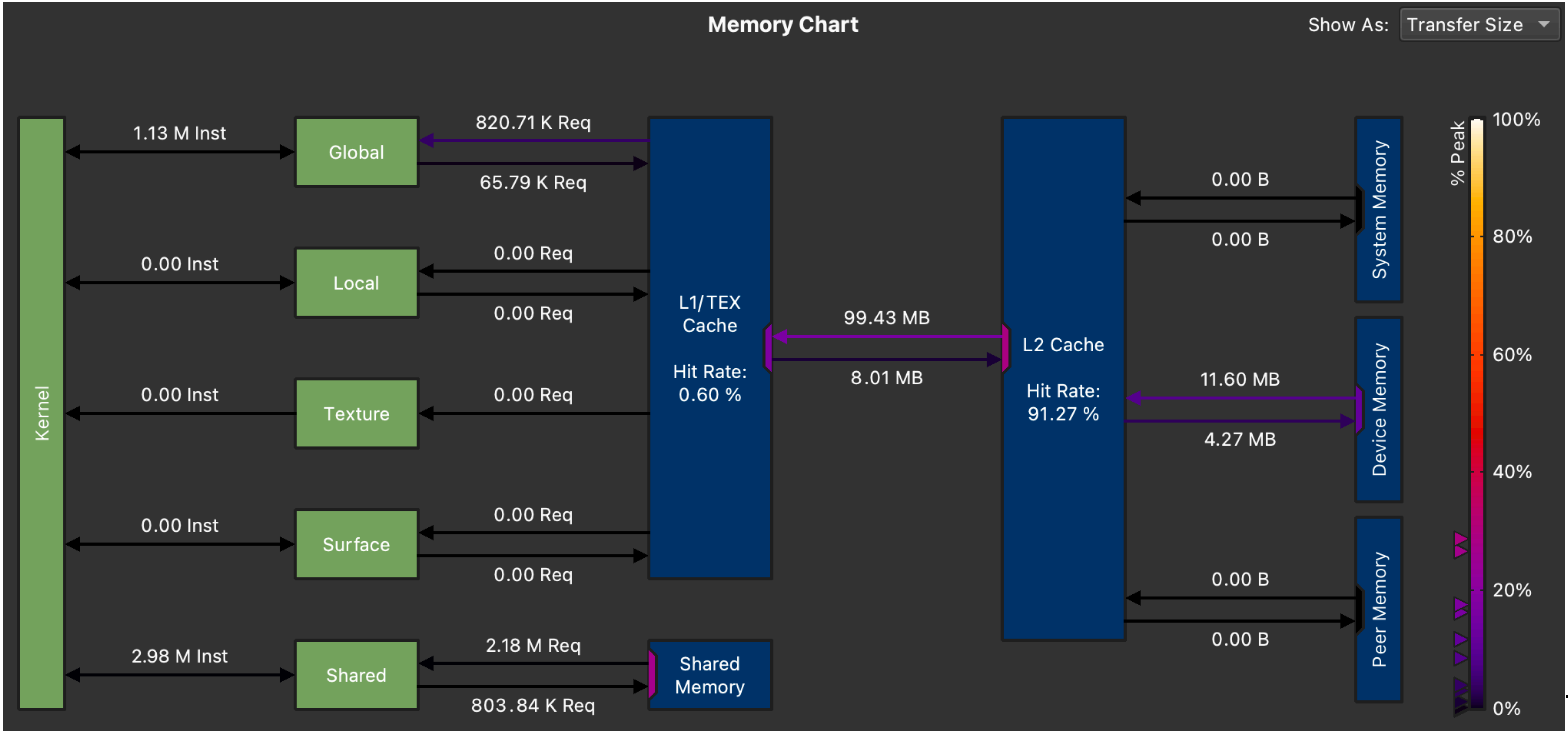
Roofline Analysis The ratio of peak float (fp32) to double (fp64) performance on this device is 32:1.The kernel achieved 85% of this device's fp32 peak performance and 0% of its fp64 peak performance.



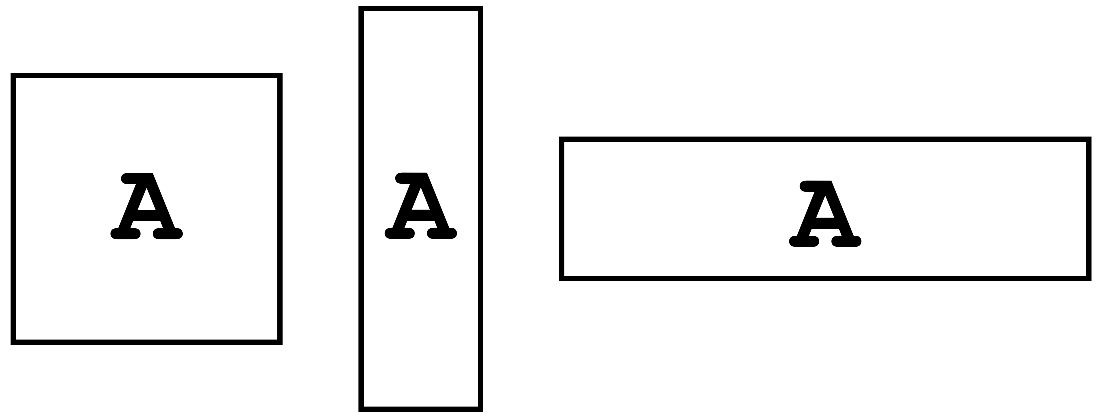
ROOFLINE ANALYSIS



MEMORY ANALYSIS



SKEWED MATRICES

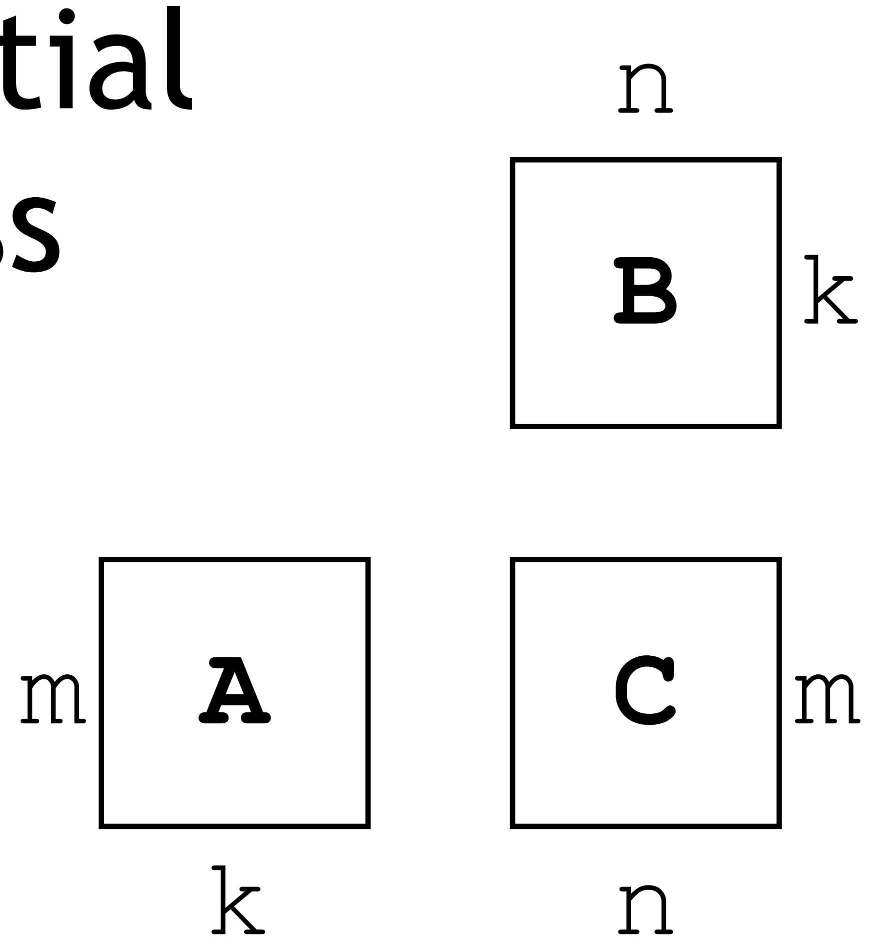


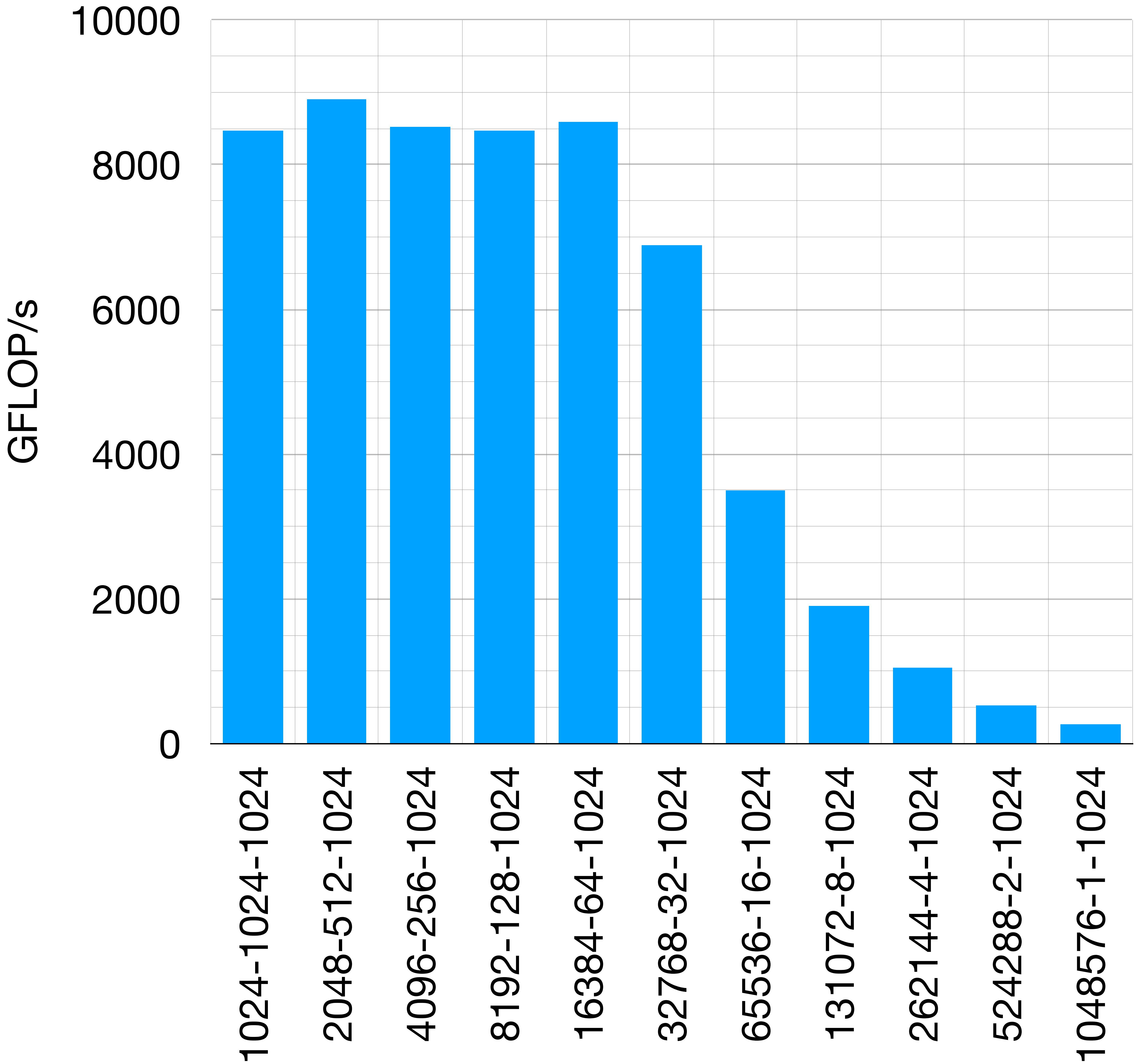
Peak performance assumption only holds true for square matrices

- Notation: `m-n-k` parameters of `cublasSgemm`
- Total work identical

Reality: substantial performance loss

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$$





USING NCU TO UNDERSTAND MORE

Profile your parametrized application and record to file

```
for ((i=1;i<=1024;i*=2)); do ncu -f --set full -o cuBLAS-skewed-$((1024*$i))-$
((1024/$i))-$((1024)) ./cuBLAS-test-sm75 $((1024*$i)) $((1024/$i)) $((1024));
done
```

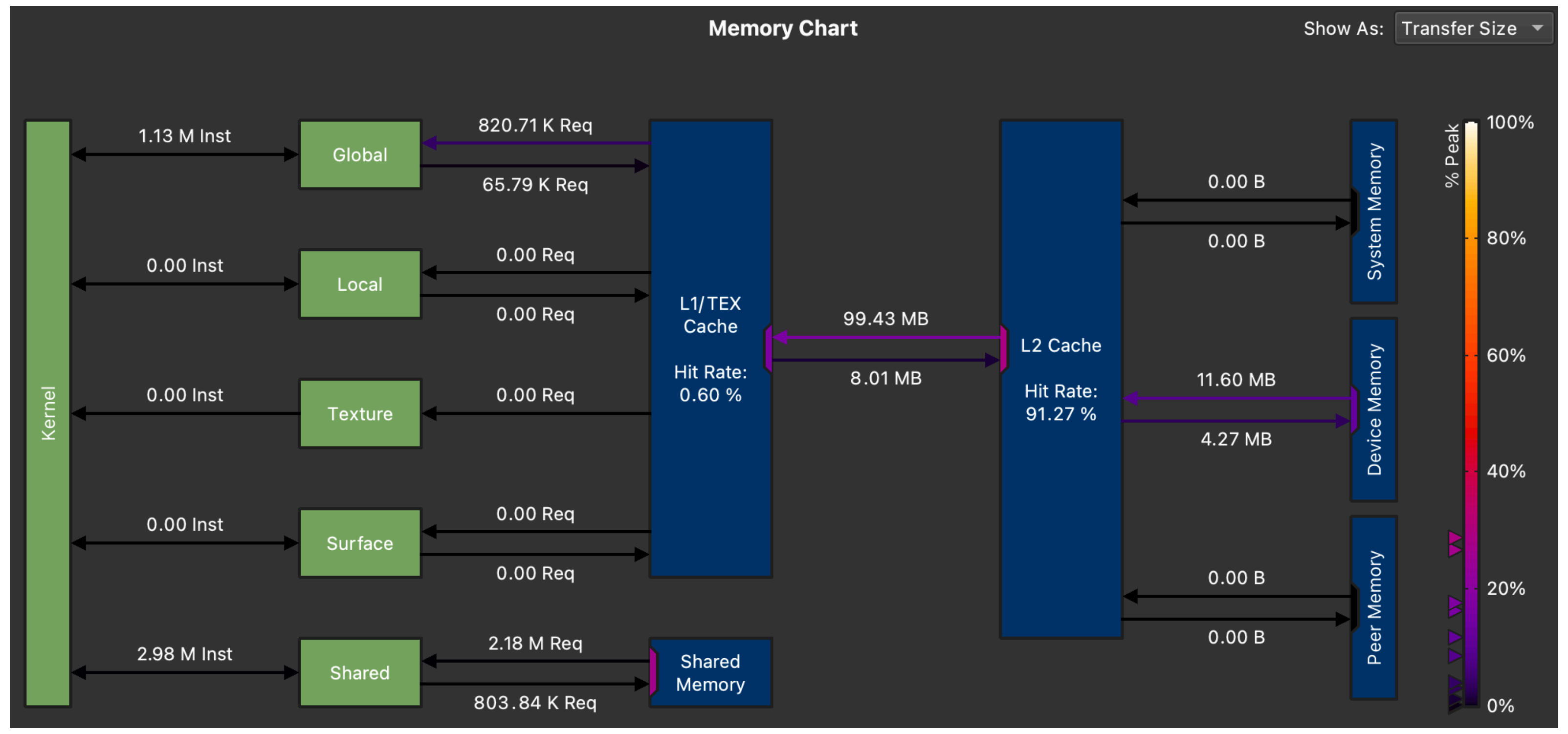
Find metrics of interest

Postprocess record file

```
ncu --import <file.ncu-rep> —details-all
```

```
L1 hit rate:     l1tex__t_sector_hit_rate.pct
L2 hit rate:     lts__t_sector_hit_rate.pct
Shared memory:   sass__inst_executed_shared_loads
                 sass__inst_executed_shared_stores
SM -> L1:        l1tex__t_output_wavefronts_pipe_lsu_mem_global_op_ld.sum
                 l1tex__t_output_wavefronts_pipe_lsu_mem_global_op_st.sum
L1 -> L2:        l1tex__t_sectors_pipe_lsu_mem_global_op_ld.sum
                 l1tex__t_sectors_pipe_lsu_mem_global_op_st.sum
L2 -> DRAM:      dram__sectors_read.sum
                 dram__sectors_write.sum
```

SKEWED MATRICES - GLOBAL READ TRAFFIC



Traffic from L1 to L2 vs. traffic from L2 to DRAM

Reminder for matrix size of N*N

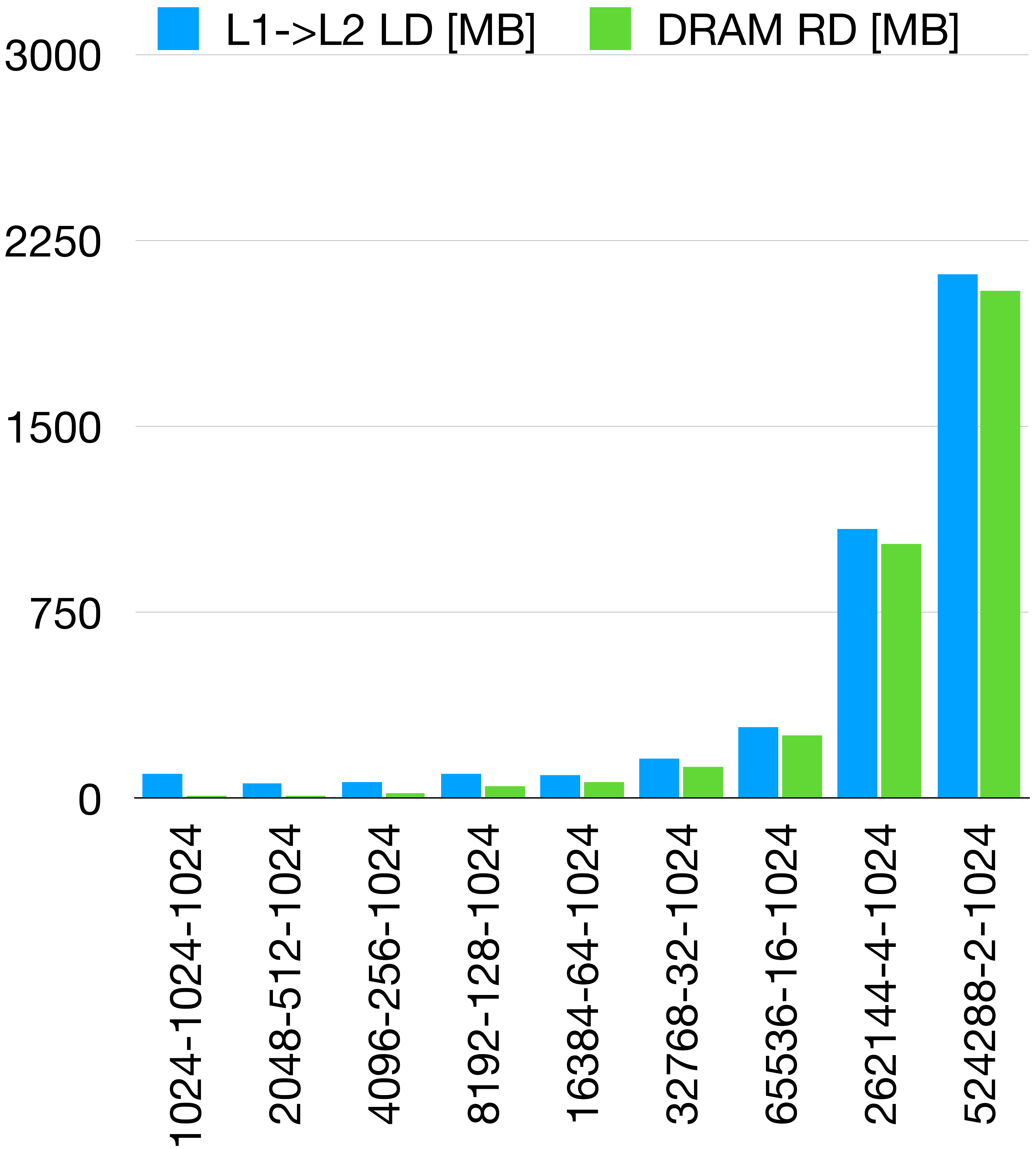
Unique memory accesses: $2N^2*4B$

(Assuming perfect caching)

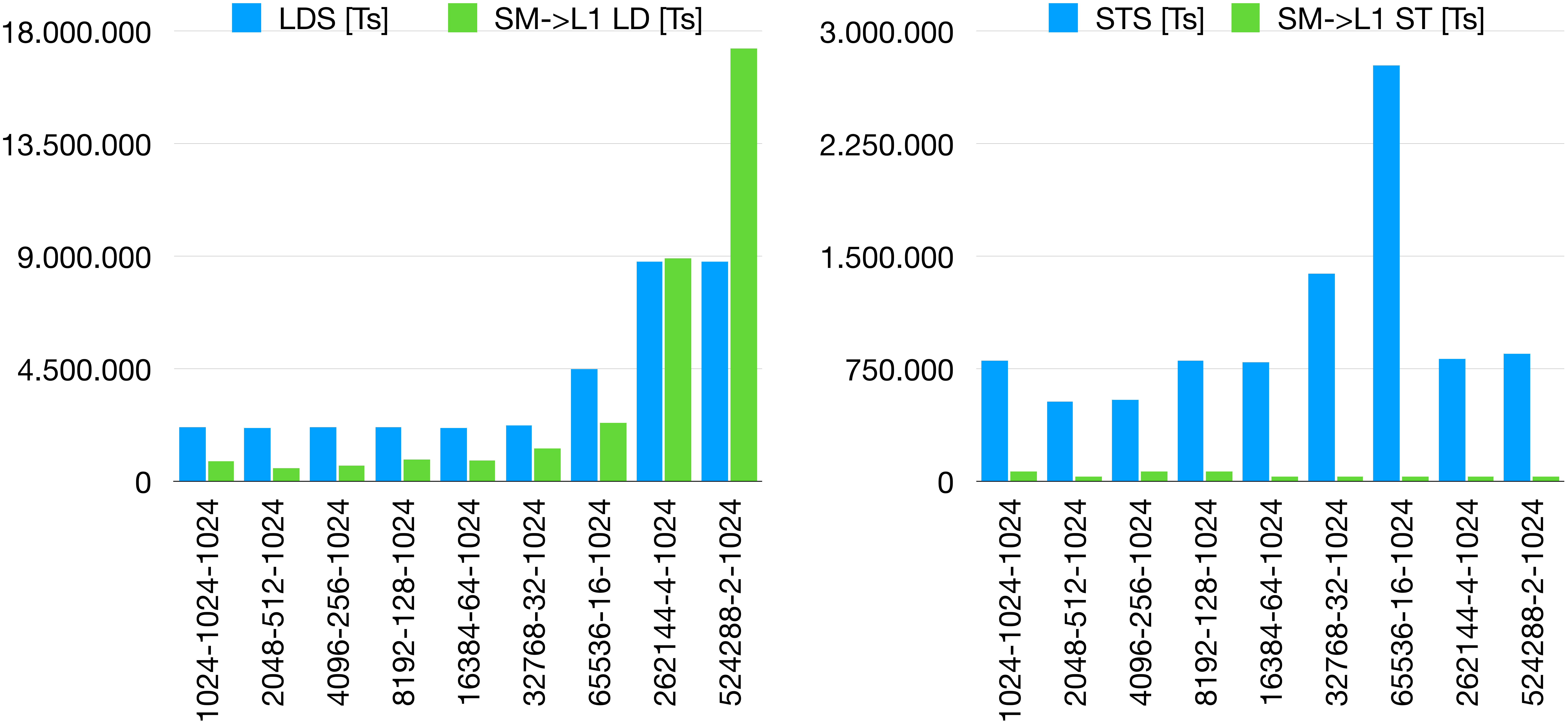
Resulting read traffic

From 1.25x (2048-512-1024)

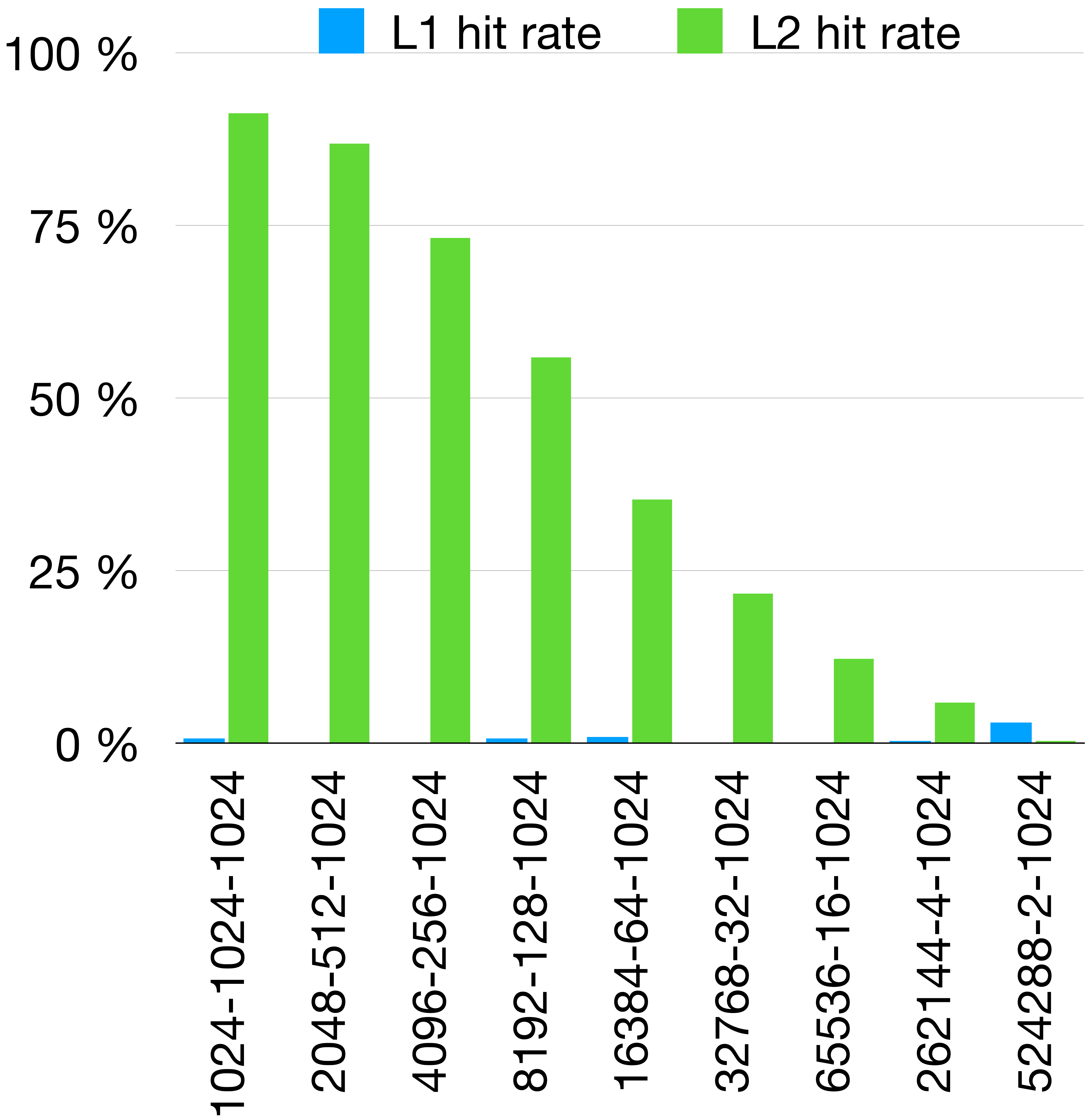
to 256.03x (524288-2-1024)



SKEWED MATRICES - SHARED MEMORY VS. GLOBAL MEMORY TRANSACTIONS

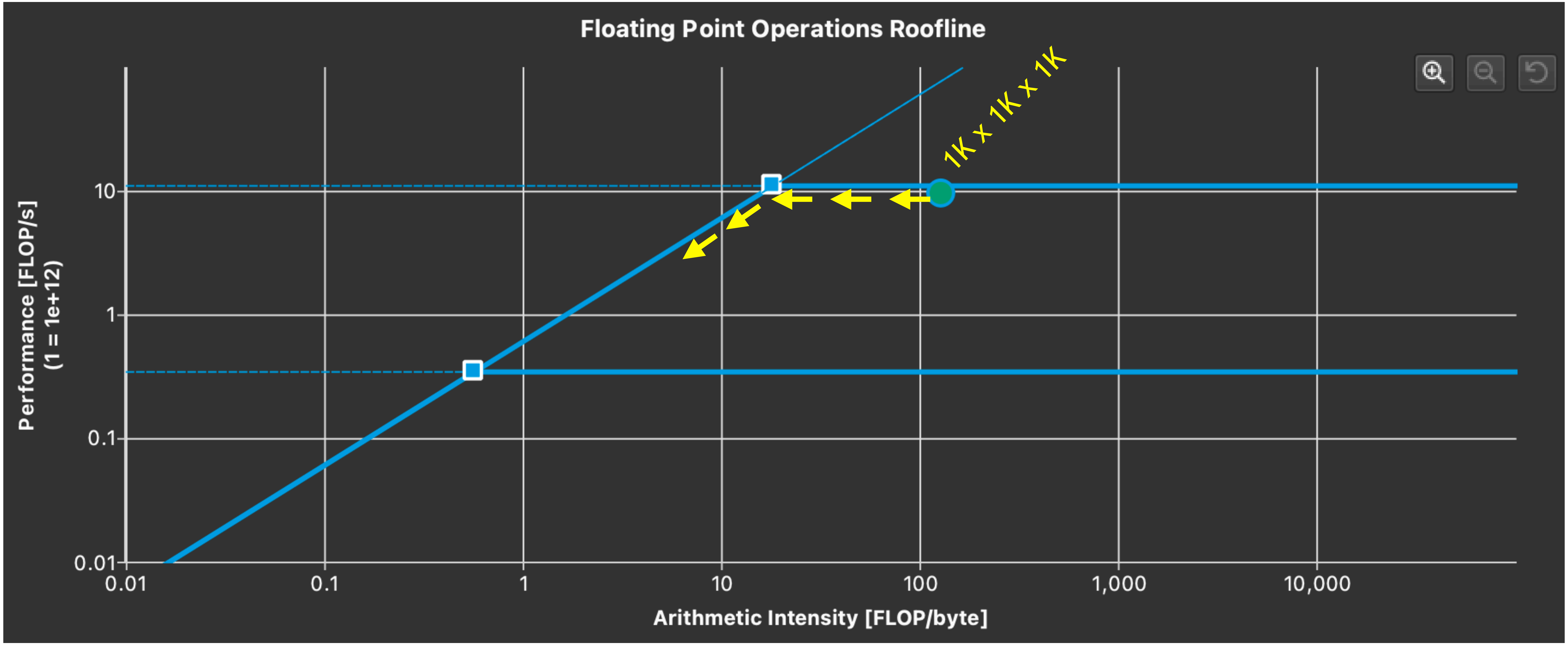


CACHE HIT RATES AND INTERNALS



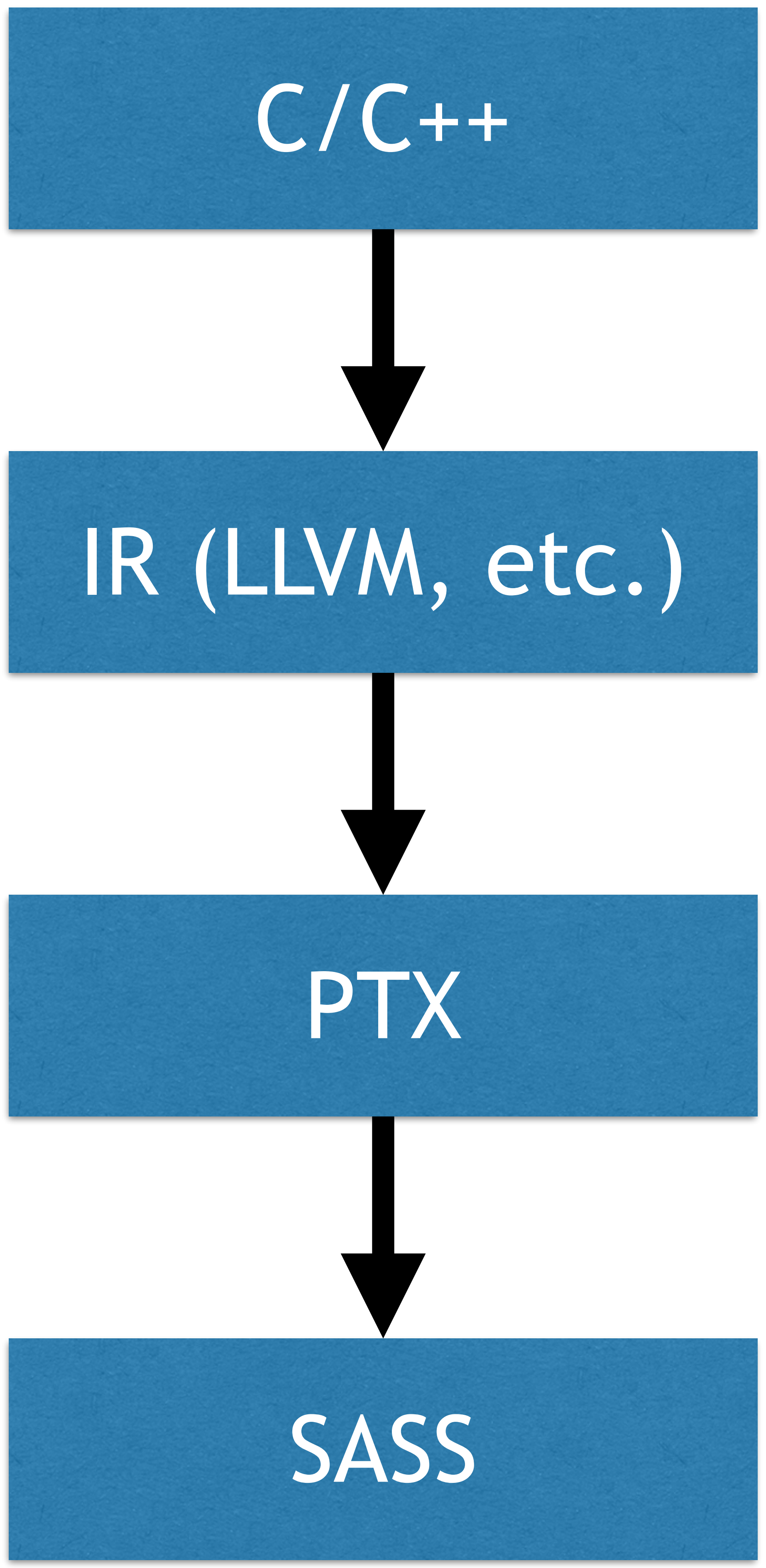
Operation	Kernel
1024-1024-1024	volta_sgemm_128x64_nn
2048-512-1024	volta_sgemm_128x128_nn
4096-256-1024	volta_sgemm_128x128_nn
8192-128-1024	volta_sgemm_128x64_nn
16384-64-1024	volta_sgemm_128x64_nn
32768-32-1024	volta_sgemm_128x32_sliced1x4_nn
65536-16-1024	volta_sgemm_128x32_sliced1x4_nn
131072-8-1024	scal_64addr_kernel
	scal_64addr_kernel
	scal_64addr_kernel
	sgemm_largek_lds64
262144-4-1024	gemmSN_NN_kernel
524288-2-1024	gemmSN_NN_kernel
1048576-1-1024	kernel
	kernel
	splitKreduce_kernel

ROOFLINE ANALYSIS



PROFILING AT PTX LEVEL

Profiling level



Lorenz Braun, Holger Fröning, CUDA Flux: A Lightweight Instruction Profiler for CUDA Applications, Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS19), held as part of ACM/IEEE Supercomputing 2019 (SC19), Denver, CO, USA.

CURRENTLY AVAILABLE TOOLS FOR PROFILING

Hardware performance-counter based: nvprof & NSight

- CUDA API trace
- Light to heavy performance impact
- Slowdown due to kernel replays

GPU simulators: GPGPU-Sim, Multi2Sim, Barra

- Very detailed analyses possible
- Very slow (10^5 - 10^6)
- Usually behind currently available hardware

Instrumentation based: GPU Ocelot/Lynx, SASSI, NVBit (Research Prototype)

- Custom profiling
- No hardware metrics such as cache hit-rate
- Fast, low overhead
- Lifetime often limited

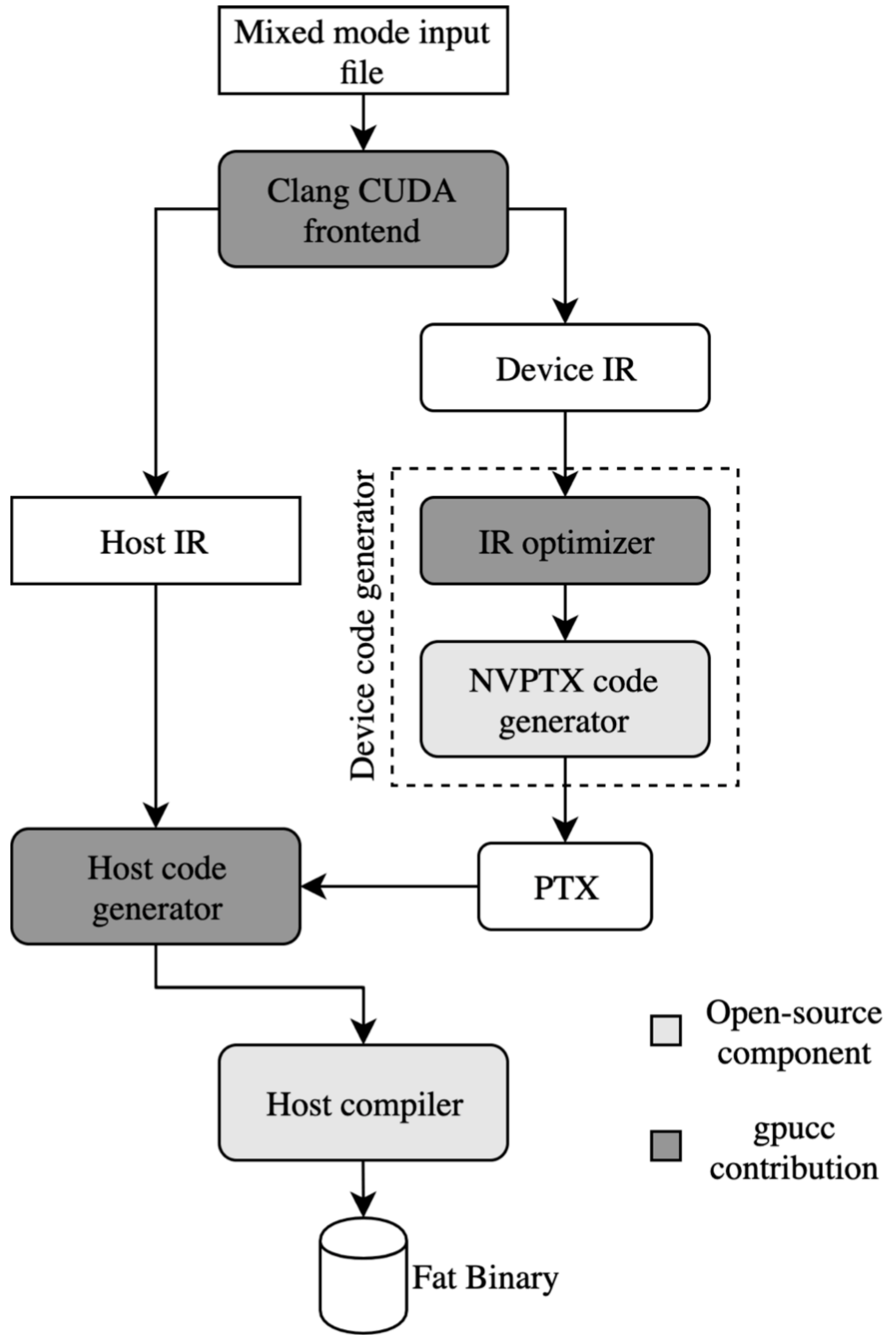
THE LLVM COMPILER FRAMEWORK AND CUDA

Since integration of gpucc [1], CUDA code is natively supported

Framework can be split up in front-end, 'middle-end' (optimizer) and back-end

Middle-end can be easily extended by registering custom transformation passes

CUDA compilation is implemented using mixed mode compilation flow



[1] Jingyue Wu, et al. 2016. Gpucc: an open-source GPGPU compiler. CGO. DOI:https://doi.org/10.1145/2854038.2854041

CUDA FLUX: LLVM-BASED CODE INSTRUMENTATION FOR PROFILING

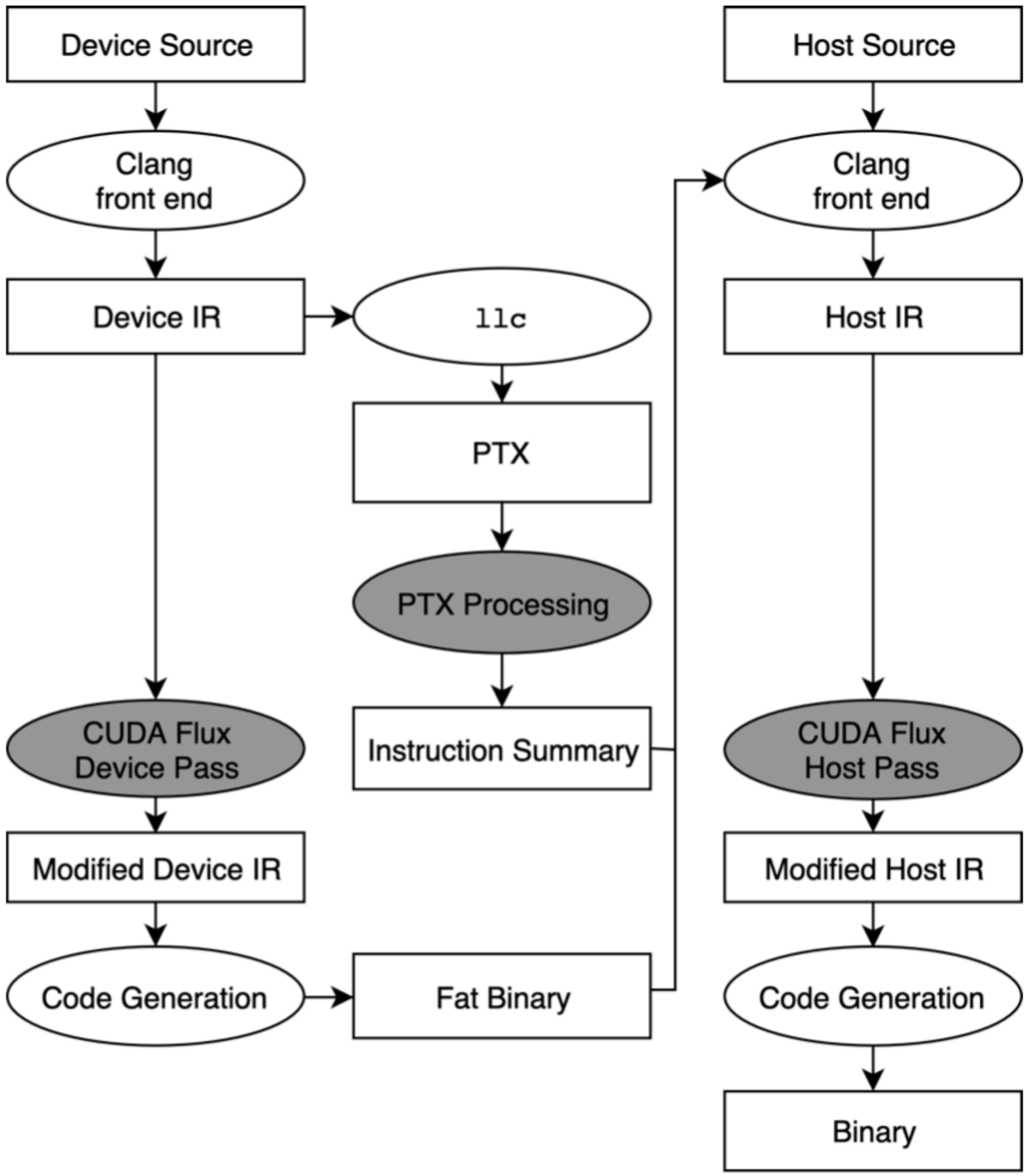
Static runtimes manage instrumentation counters

- Device pass: link device code to runtime
- Host pass: link host code to runtime

PTX processing

- Iterates over all kernels
- Produces a PTX block summary containing instructions counts of all basic blocks

Flexible: instrumentation on either warp-level, CTA-level or full thread-grid



COMPUTING INSTRUCTIONS ON PTX LEVEL

Each basic block (BB) is instrumented

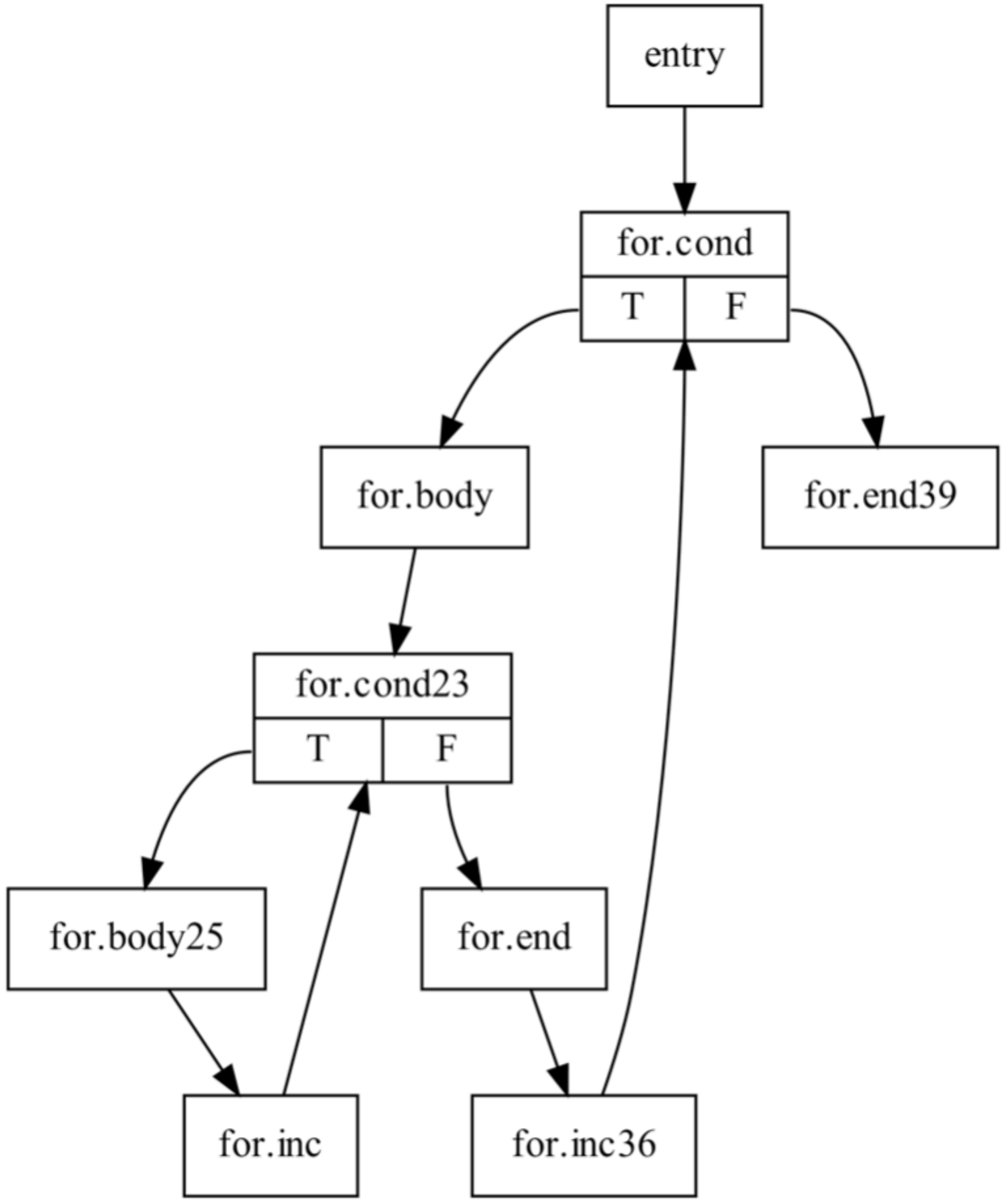
- Begin of BB = branch target, no branches/jumps inside a BB except for end of BB

On entering a BB the corresponding counter for the block is increased

After kernel execution: PTX instruction counters are calculated using BB counter and the PTX instruction summary

Advantages

- Fine grained profiling
- Time does not depend on number of metrics monitored
- PTX is an accessible intermediate assembly for CUDA GPUs



CFG for 'matrixMul' function

LIMITATIONS

Profiling on PTX level, not SASS

- Closer to high-level code, farer away from hardware

Kernel definition and kernel launch need to be in the same compilation module

Modification of build system needed (in majority of cases):

- Change `nvcc` to `clang++`
- Non compatible compiler flags
- Easy on good/simple build systems, error-prone on complicated build systems

Instrumentation takes place at IR level

Texture memory is not supported (clang limitation)

PERFORMANCE EVALUATION

CUDA Flux vs. nvprof using the Polybench-GPU Benchmark

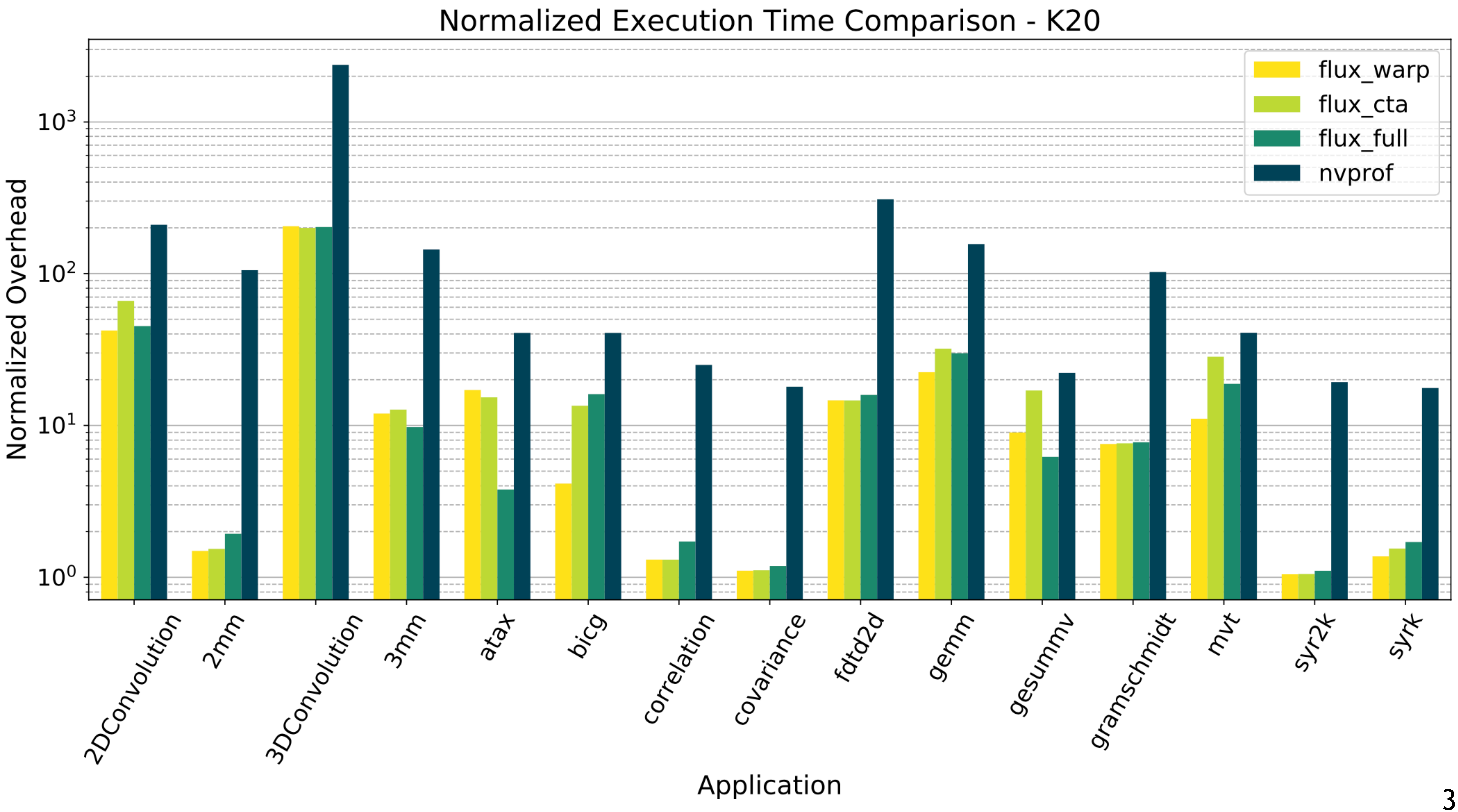
- Measurements on NVIDIA Tesla K20 and Titan Xp
- Only kernel time is measured using a median of five executions

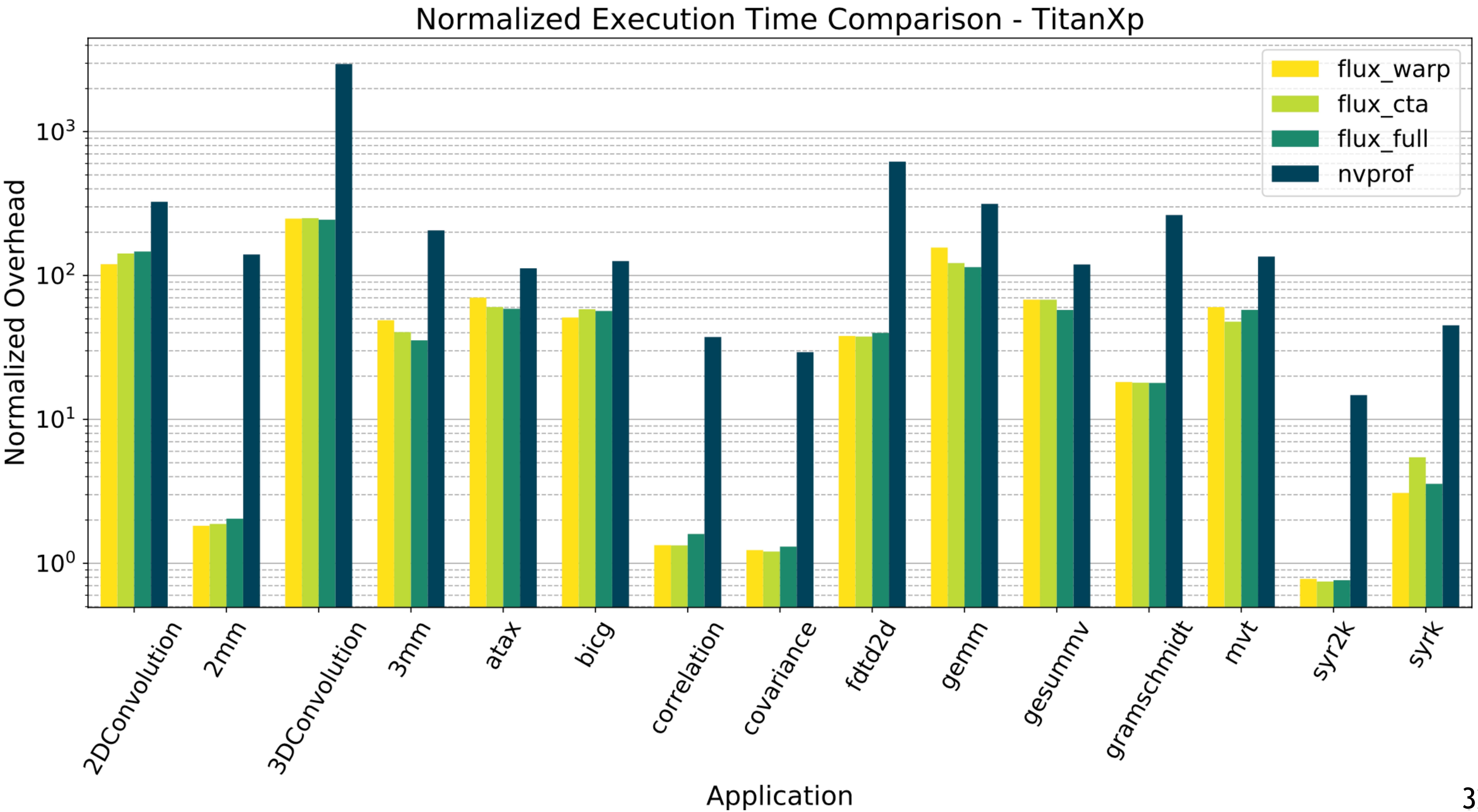
Four different profiling configurations

- flux_warp: all threads of one single warp
- flux_cta: all threads of one single CTA (aka. thread block)
- flux_full: all threads of the complete thread grid
- nvprof: measurement with 8 different metrics instruction counter metrics

Baseline measurement without any instrumentation or profiling is used to normalize the results

Open-Source: available on github: https://github.com/UniHD-CEG/cuda-flux and available in the lab (`module load cuda_flux`)


Normalized Execution Time Comparison - K20
flux_warp
flux_cta
flux_full
nvprof
Normalized Overhead
10^3
10^2
10^1
10^0
2DConvolution
2mm
3DConvolution
3mm
atax
bicg
correlation
covariance
fdtd2d
gemm
gesummv
gramschmidt
mvt
syr2k
syrk
Application



Normalized Execution Time Comparison - TitanXp
Normalized Overhead
10^3
10^2
10^1
10^0
flux_warp
flux_cta
flux_full
nvprof
2DConvolution
2mm
3DConvolution
3mm
atax
bicg
correlation
covariance
fdtd2d
gemm
gesummv
gramschmidt
mvt
syr2k
syrk
Application

WRAPPING UP

SUMMARY

Models help us to understand performance based on application behavior

- Roofline model
- 3C model

Profiling helps us to understand code behavior in detail

- Usually based on hardware performance counters, but that’s expensive
- Tools: Nsight, nvprof, CUDA Flux, etc.
- Methodology: Model/Intuition/Hypothesis -> Experiment design -> Profiling -> Analysis

Excursion: Predictive performance modeling

- Reasoning about performance of application and/or processor without executing it (at least not on every combination of the tuple)
- Execution statistics & HW characterization = performance (time, power, energy) prediction

EXCURSION:
(PREDICTIVE) PERFORMANCE MODELING

PERFORMANCE MODELING

	Speed	Ease	Flexibility	Accurracy	Scalability
Ad-hoc Analytical Models	1	3	2	4	1
Structured Analytical Models	1	2	1	4	1
Functional Simulation	3	2	2	3	3
Cycle accurate Simulation	4	2	2	2	4
HW Emulation (FPGA)	3	3	3	2	3
Similar hardware measurement	2	1	4	2	2
Node Prototype	2	1	4	1	4
Prototype at Scale	2	1	4	1	2
Final System	-	-	-	-	-
Learning-based Models	1	2	1	2	1

Adapted from: Kyle L. Spafford and Jeffrey S. Vetter. 2012. Aspen: a domain specific language for performance modeling. SC '12

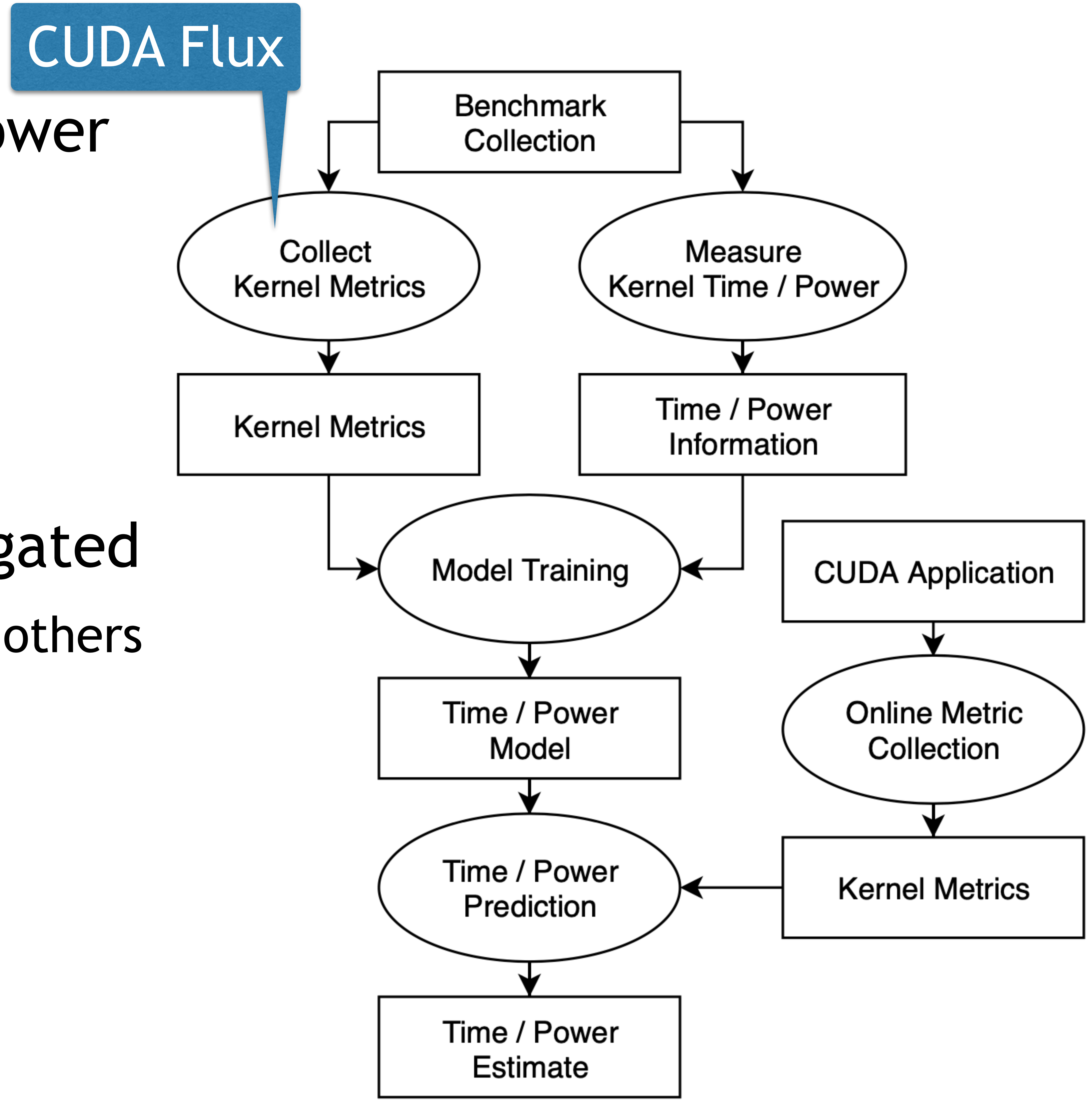
HETEROGENEITY AND PORTABILITY

Predictions about execution time and power consumption

- Runtime/scheduling decisions
- Provisioning decisions
- Performance portability explorations

State-of-the-art: 25 publications investigated

- Methods: analytical (9) vs. learning (10) vs. others
- Representativeness (1-169 kernels/apps)
- Portability (1-9 GPUs)
- Availability (only 2 models published)
- DVFS support (6)
- Time (21); power consumption (10)

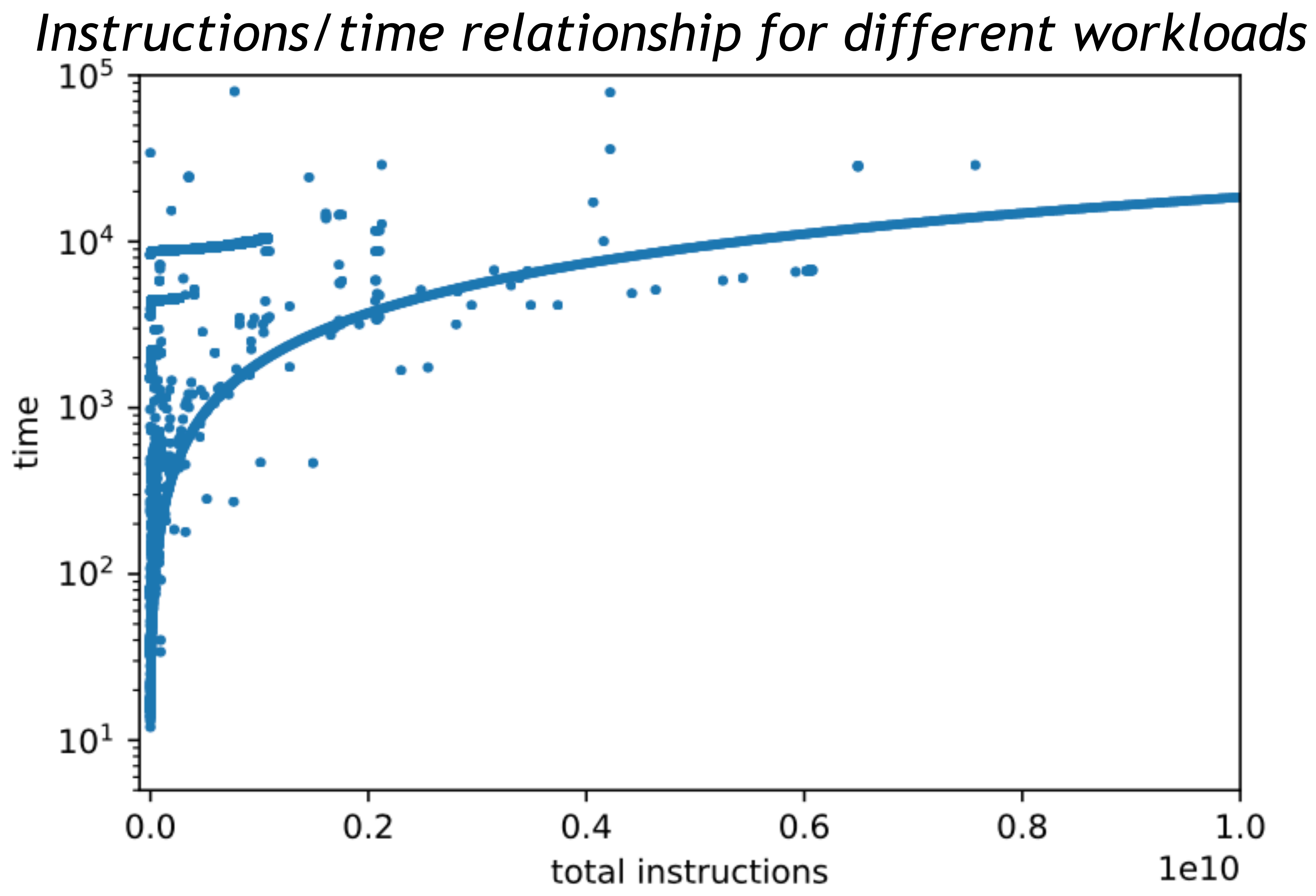


Lorenz Braun, Holger Fröning, CUDA Flux: A Lightweight Instruction Profiler for CUDA Applications, PMBS Workshop@SC2019
https://github.com/UniHD-CEG/cuda-flux

GPU MANGROVE: PORTABLE, FAST, SIMPLE

Which metrics make good features?

- Instructions executed
- FLOPs
- Memory footprint
- Kernel launch configuration
- Computational intensity
- Synchronizations



Portable code features only depend on the kernel and the data handed to it

- Hardware metrics like cache-hit rates not allowed
- Creation of models for new GPUs requires only time and power measurements
- Instruction statistics are essential; represent actual work of the processing units

GPU MANGROVE: GPU MANGROVE: PORTABLE, FAST, SIMPLE PERFORMANCE PREDICTION

RandomForests

Light computational workload

Likely to over-fit (but can be improved by training method)

Works well with even few samples

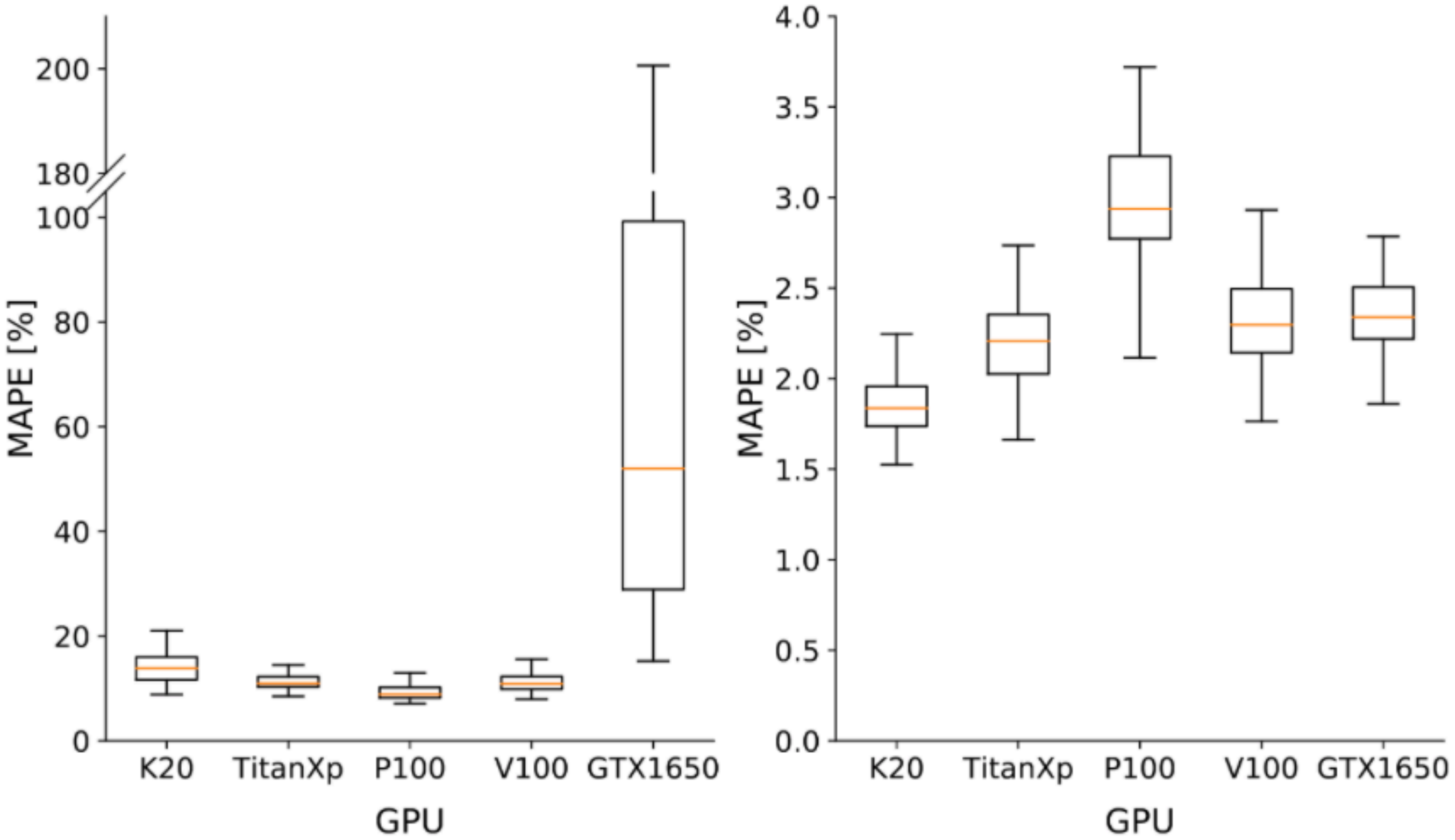
Interpolation outside range of training data is difficult

Methodology

189 unique kernels from Parboil, Rodinia, Polybench-GPU and SHOC

Prediction accuracy: 8.86–52.0% for time, 1.84–2.94% for power, across five different GPU

Prediction latency: 15-108ms (not optimized)



Time prediction

Power prediction

TACO 2021 article: https://doi.org/10.1145/3431731, https://github.com/UniHD-CEG/gpu-mangrove
Tutorial @ HiPEAC2021: https://www.hipeac.net/2021/budapest/#/program/sessions/7856/

BACKUP / VOLTA

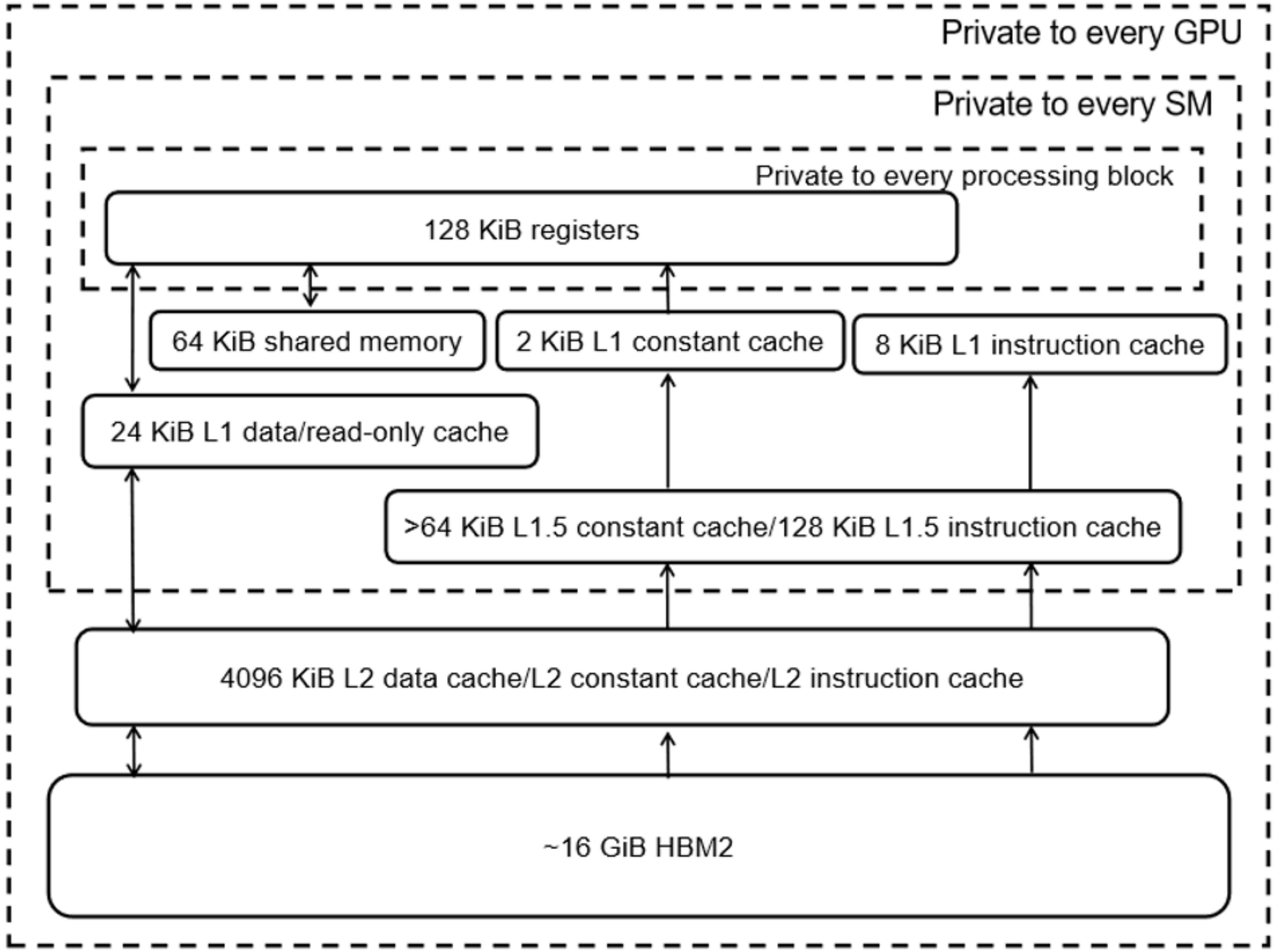


Figure 3.3: Memory hierarchy of the Pascal P100 GPU (GP104).

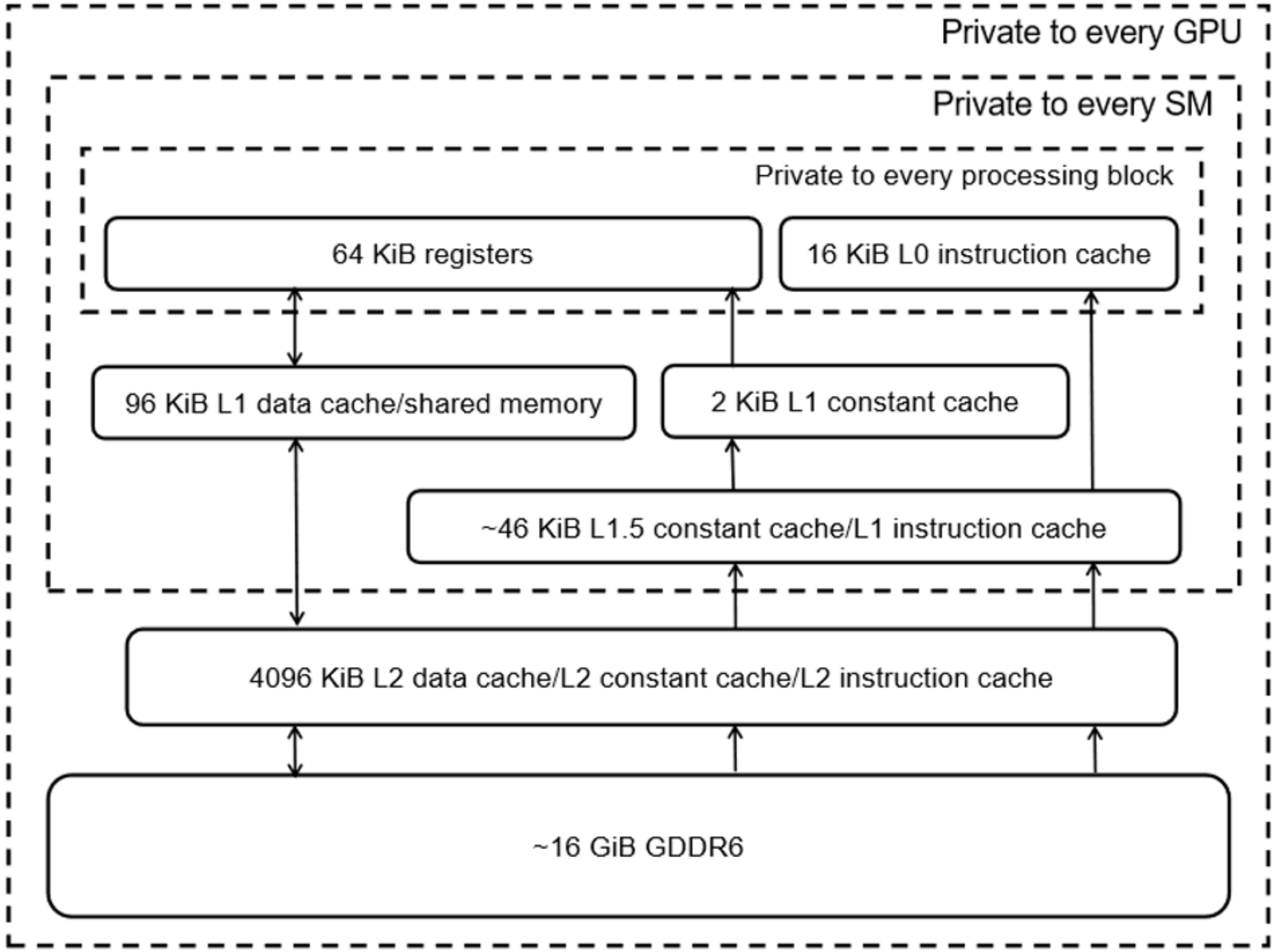


Figure 3.1: Memory hierarchy of the Turing T4 GPU (TU104).