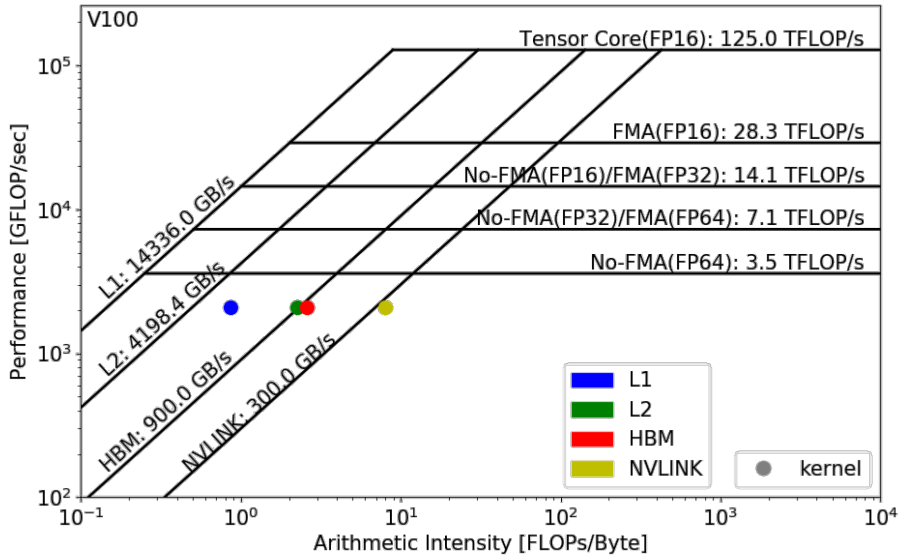
Mechanism behind Roofline Data Collection

Charlene Yang
NERSC, July 8 2020

https://www.nersc.gov/users/training/events/roofline-on-nvidia-gpus-hackathon/

The Goal

- Hierarchical Roofline
  - L1, L2, device memory, system memory (NVLINK or PCIe), peer-to-peer GPU, NIC, …
- Data precisions
  - double, single, half, quarter, …
- Instruction types
  - floating-point, integer, …
  - FMA, mul/add, …
- Execution units
  - CUDA core/Tensor core



Methodology

1. **Roofline ceilings**
   - theoretical values for peak bandwidth/computational throughput,
   - or, empirical values from Empirical Roofline Toolkit (ERT)
     https://bitbucket.org/berkeleylab/cs-roofline-toolkit/

2. **Application data**
   - measure three quantities: **time**, **FLOPs**, **data movement (Bytes)**
   - calculate AI and throughput:

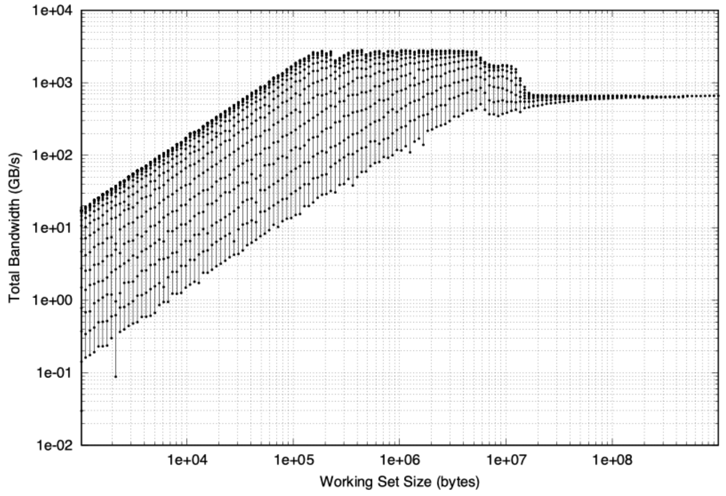
$$\text{Arithmetic Intensity} = \frac{\text{FLOPs}}{\text{data movement}} \qquad \text{Performance} = \frac{\text{FLOPs}}{\text{time}}$$

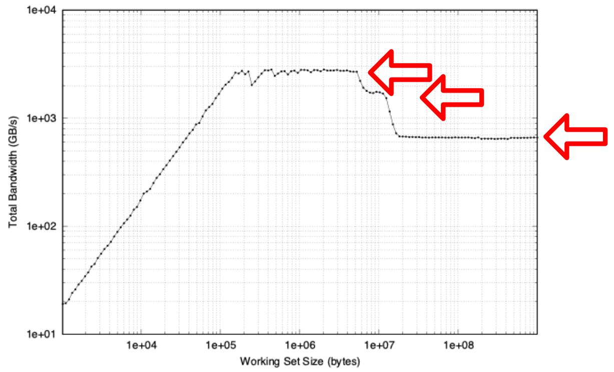
(x: FLOPs/Byte) **(y: GFLOP/s)**

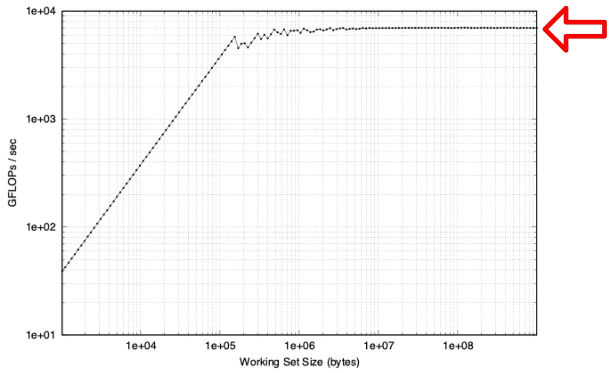
3. **Roofline chart**
   - Nsight Compute
   - or, customized scripts: https://gitlab.com/NERSC/roofline-on-nvidia-gpus/

1. Empirical Roofline Toolkit (ERT)

- Sweeps through a range of configurations
- Produces a more realistic set of peaks in real environment







(left) working set sizes
(top right) bandwidths, (top bottom) FLOP/s

2. Application Data (manual)

For tighter integration with your workflow, you can collect relevant metrics and plot Roofline charts manually:

srun -n1 nv-nsight-cu-cli -k <*kernels*> --metrics <*metrics*> -o output ./app

	Nsight Compute metrics and arithmetics
Time	`sm__cycles_elapsed.avg / sm__cycles_elapsed.avg.per_second`

FLOPs	CUDA core [`x`: `d` for DP, `f` for SP, and `h` for HP]
	`sm__sass_thread_inst_executed_op_xadd_pred_on.sum +`
	`sm__sass_thread_inst_executed_op_xfma_pred_on.sum * 2 +`
	`sm__sass_thread_inst_executed_op_xmul_pred_on.sum`
	Tensor core
	`sm__inst_executed_pipe_tensor.sum * 512`

Bytes	device memory: `dram__bytes.sum` L2: `lts__t_bytes.sum` L1: `l1tex__t_bytes.sum`

https://gitlab.com/NERSC/roofline-on-nvidia-gpus/

OLD: nvprof-based

Runtime:

Time per invocation of a kernel

```
nvprof --print-gpu-trace ./application
```

Average time over multiple invocations

```
nvprof --print-gpu-summary ./application
```

FLOPs:

CUDA Core: Predication aware and complex-operation aware (such as divides)

`nvprof --kernels ‘kernel_name’ --metrics ‘flop_count_xx’ ./application` **e.g.** `flop_count_{dp/dp_add/dp_mul/dp_fma, sp*, hp*}`

Tensor Core: (more details later)

```
--metrics tensor_precision_fu_utilization
```

0-10 integer range, 0-0, 10-125TFLOP/s;

multiply by run time -> FLOPs

Bytes: **on different cache levels**

Bytes = (read transactions + write transactions) x transaction size

```
nvprof --kernels ‘kernel_name’ --metrics ‘metric_name’./application
```

Level	Metrics	Transaction Size
First Level Cache*	`gld_transactions, gst_transactions, atomic_transactions, local_load_transactions, local_store_transactions, shared_load_transactions, shared_store_transactions`	32B
Second Level Cache	`l2_read_transactions, l2_write_transactions`	32B
Device Memory	`dram_read_transactions, dram_write_transactions`	32B
System Memory	`system_read_transactions, system_write_transactions`	32B

OLD: NCU-based (CUDA 10)

Quantities	Metrics	Arithmetics

Runtime	`smsp__cycles_elapsed.sum,`
	`smsp__cycles_elapsed.sum.per_second,`
	`smsp__pipe_tensor_op_hmma_cycles_active.sum,`
	`smsp__pipe_tensor_op_hmma_cycles_active.sum.per_second`
		`sum/sum.per _second`

FLOPs	`smsp__sass_thread_inst_executed_op_dadd_pred_on.sum,`
	`smsp__sass_thread_inst_executed_op_dmul_pred_on.sum,`
	`smsp__sass_thread_inst_executed_op_dfma_pred_on.sum,`
	`smsp__sass_thread_inst_executed_op_fadd_pred_on.sum,`
	`smsp__sass_thread_inst_executed_op_fmul_pred_on.sum,`
	`smsp__sass_thread_inst_executed_op_ffma_pred_on.sum,`
	`smsp__sass_thread_inst_executed_op_hadd_pred_on.sum,`
	`smsp__sass_thread_inst_executed_op_hmul_pred_on.sum,`
	`smsp__sass_thread_inst_executed_op_hfma_pred_on.sum,`
		`#d: double precision`
		`#f: signle precision`
		`#h: half precision`
		
		`fma x 2 + mul + add`

Runtime and FLOPs:

```
nv-nsight-cu-cli -k `kernel_name' --metrics
`metrics_name' ./application
```

Similar to before, Bytes for different cache levels:

Bytes = (read transactions + write transactions) x transaction size)

Level	Metrics	Transaction Size

First Level Cache*	`l1tex__t_sectors_pipe_lsu_mem_global_op_ld.sum,`
	`l1tex__t_sectors_pipe_lsu_mem_global_op_st.sum,`
	`l1tex__t_set_accesses_pipe_lsu_mem_global_op_red.sum,`
	`l1tex__t_sectors_pipe_lsu_mem_local_op_ld.sum,`
	`l1tex__t_sectors_pipe_lsu_mem_local_op_st.sum,`
	`smsp__inst_executed_op_shared_ld.sum,`
	`smsp__inst_executed_op_shared_st.sum`
		32B

Second Level Cache	`lts__t_sectors_op_read.sum,`
	`lts__t_sectors_op_write.sum`
		32B

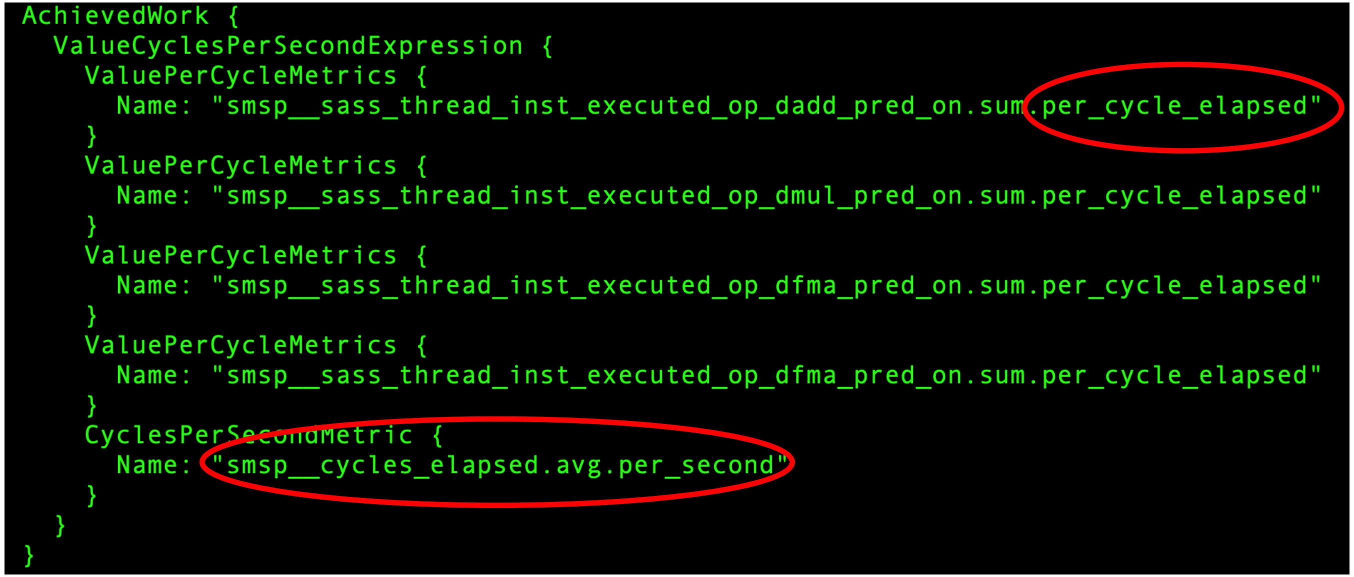
Device Memory	`dram__sectors_write.sum, dram__sectors_read.sum`	32B

System Memory	`lts__t_sectors_aperture_sysmem.op_read.sum,`
	`lts__t_sectors_aperture_sysmem.op_write.sum`
		32B

2. Application Data (automatic)

Otherwise, Nsight Compute provides a set of section files for automatic Roofline data collection

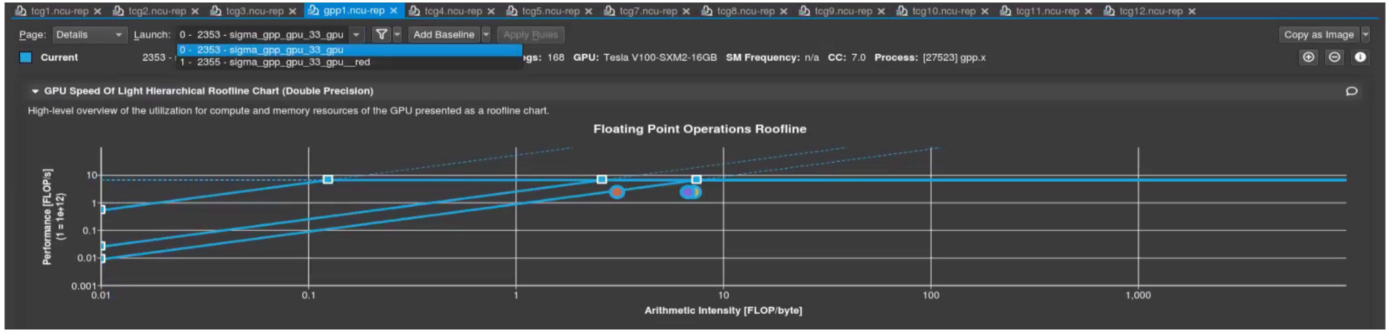
- metrics used are a bit different than in the previous table, but they will produce the same results
- SpeedOfLight_Hierarchical{Double,Single,Half,Tensor}RooflineChart.section



```
AchievedWork {
  ValueCyclesPerSecondExpression {
    ValuePerCycleMetrics {
      Name: "smsp__sass_thread_inst_executed_op_dadd_pred_on.sum.per_cycle_elapsed"
    }
    ValuePerCycleMetrics {
      Name: "smsp__sass_thread_inst_executed_op_dmul_pred_on.sum.per_cycle_elapsed"
    }
    ValuePerCycleMetrics {
      Name: "smsp__sass_thread_inst_executed_op_dfma_pred_on.sum.per_cycle_elapsed"
    }
    ValuePerCycleMetrics {
      Name: "smsp__sass_thread_inst_executed_op_dfma_pred_on.sum.per_cycle_elapsed"
    }
    CyclesPerSecondMetric {
      Name: "smsp__cycles_elapsed.avg.per_second"
    }
  }
}
```

3. Roofline Charts

- Automatic if you are using the Roofline feature in Nsight Compute



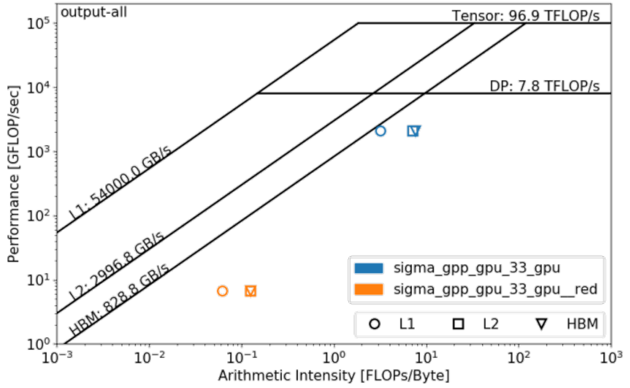
Page: Details Launch: 0 - 2353 - sigma_gpp_gpu_33_gpu

Current 2353 - … Regs: 168 GPU: Tesla V100-SXM2-16GB SM Frequency: n/a CC: 7.0 Process: [27523] gpp.x

GPU Speed Of Light Hierarchical Roofline Chart (Double Precision)

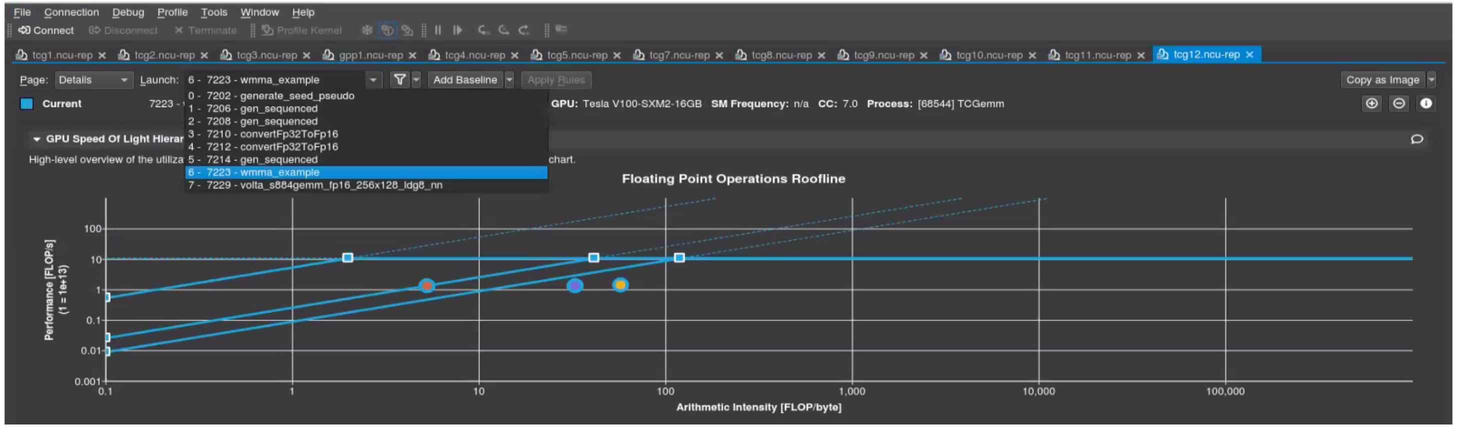
High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

- For more customized plots, please try scripts here
- https://gitlab.com/NERSC/roofline-on-nvidia-gpus/
- TWEAK them to fit your needs!



Example 1. GEMM

- A Tensor Core example using WMMA and cuBLAS
  - convertFp32ToFp16(__half*, float*, int)
  - generate_seed_pseudo(unsigned long long, unsigned long long, unsigned long long, curandOrdering, curandStateXORWOW*, unsigned int*)
  - void gen_sequenced<curandStateXORWOW, float, int, &(float curand_uniform_noargs<curandStateXORWOW>(curandStateXORWOW*, int)), rng_config<curandStateXORWOW, (curandOrdering)101> >(curandStateXORWOW*, float*, unsigned long, unsigned long, int)
  - volta_s884gemm_fp16_256x128_ldg8_nn
  - wmma_example(__half*, __half*, float*, int, int, int, float, float)



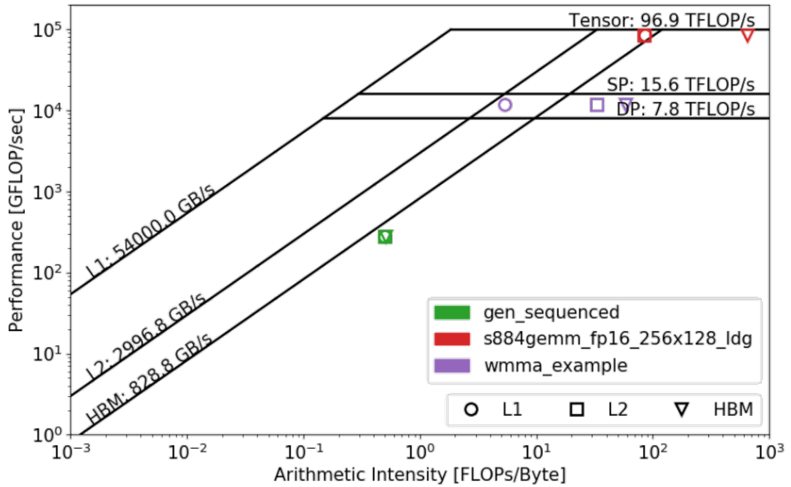
```
Page: Details    Launch: 6 -  7223 - wmma_example
0 -  7202 - generate_seed_pseudo
1 -  7206 - gen_sequenced
2 -  7208 - gen_sequenced
3 -  7210 - convertFp32ToFp16
4 -  7212 - convertFp32ToFp16
5 -  7214 - gen_sequenced
6 -  7223 - wmma_example
7 -  7229 - volta_s884gemm_fp16_256x128_ldg8_nn

Current    7223 -    GPU: Tesla V100-SXM2-16GB  SM Frequency: n/a  CC: 7.0  Process: [68544] TCGemm

GPU Speed Of Light Hierar
High-level overview of the utiliza                    chart.
```

https://github.com/NVIDIA-developer-blog/code-samples/tree/master/posts/tensor-cores

Example 1. GEMM

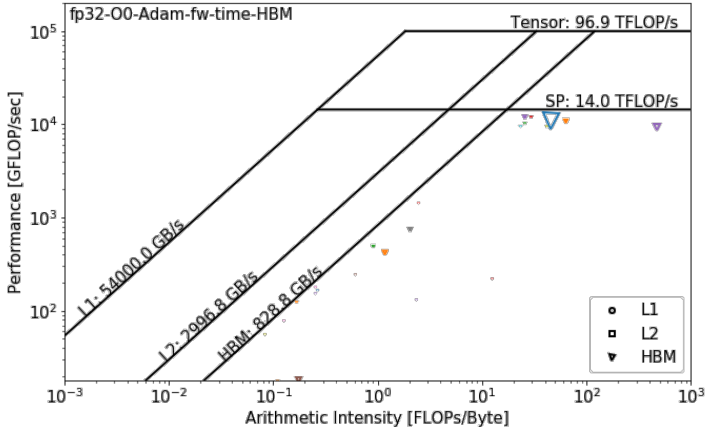
- A Tensor Core example using WMMA and cuBLAS
  - convertFp32ToFp16(__half*, float*, int)
  - generate_seed_pseudo(unsigned long long, unsigned long long, unsigned long long, curandOrdering, curandStateXORWOW*, unsigned int*)
  - void gen_sequenced<curandStateXORWOW, float, int, &(float curand_uniform_noargs<curandStateXORWOW>(curandStateXORWOW*, int)), rng_config<curandStateXORWOW, (curandOrdering)101> >(curandStateXORWOW*, float*, unsigned long, unsigned long, int)
  - volta_s884gemm_fp16_256x128_ldg8_nn
  - wmma_example(__half*, __half*, float*, int, int, int, float, float)

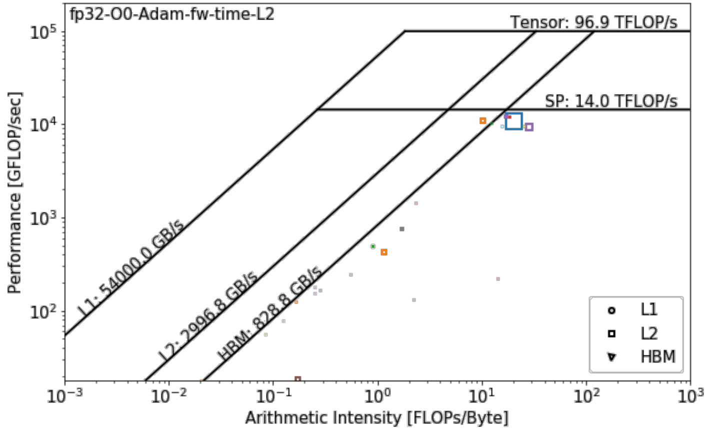
  - Multiple kernels in one plot
  - Shorten/reformat kernel names
  - Adjust the xmin/xmax and ymin/ymax
  - Prune 0-AI kernels (first 2)

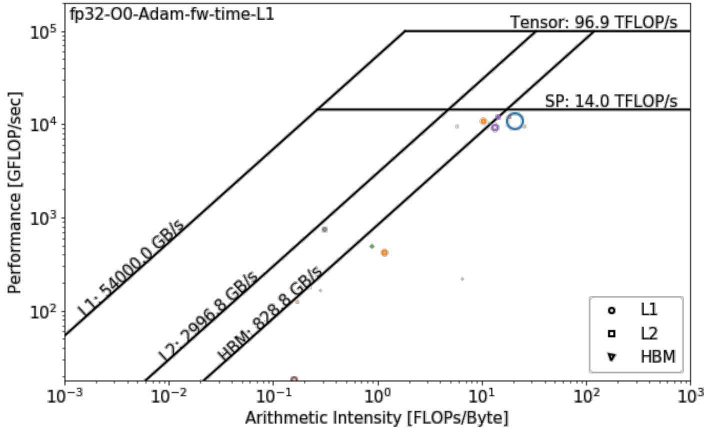


https://github.com/NVIDIA-developer-blog/code-samples/tree/master/posts/tensor-cores

Example 2. PyTorch climate seg.

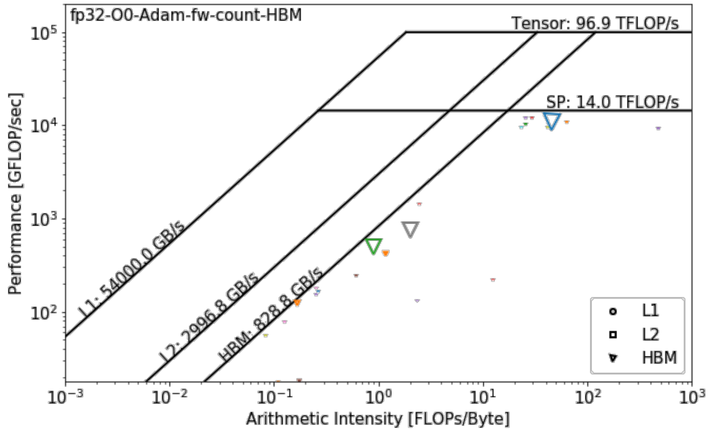


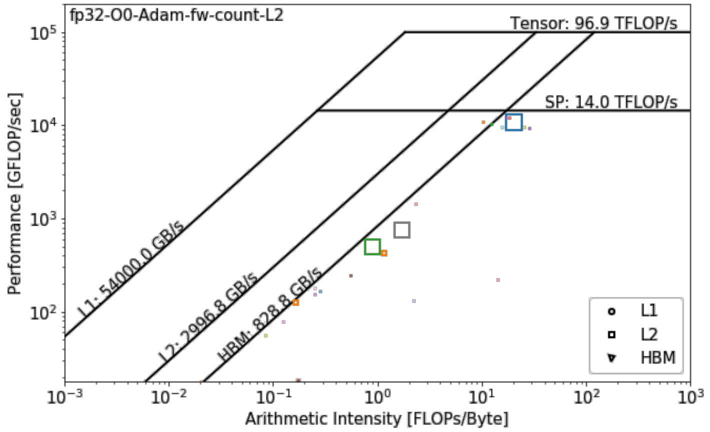


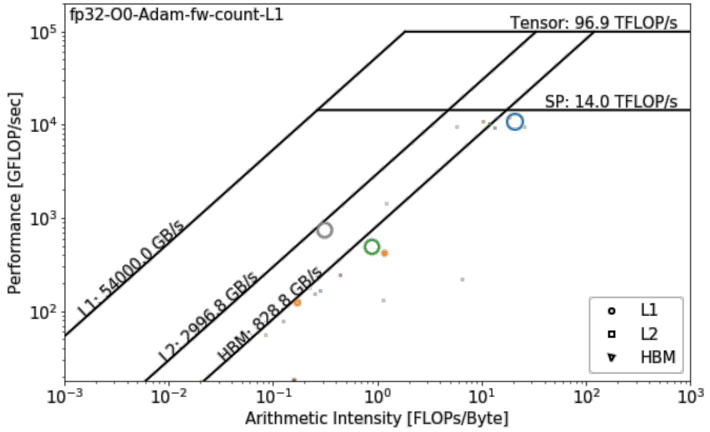


- **Separate HBM, L2, L1 charts**
- Marker size based on **percentage of time**, kernel count, FLOPs, etc
- AMP O0 vs O1 (CUDA core vs Tensor Core)

Example 2. PyTorch climate seg.

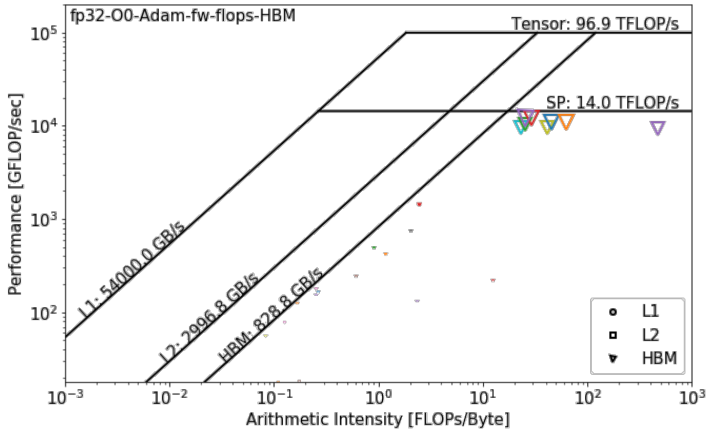


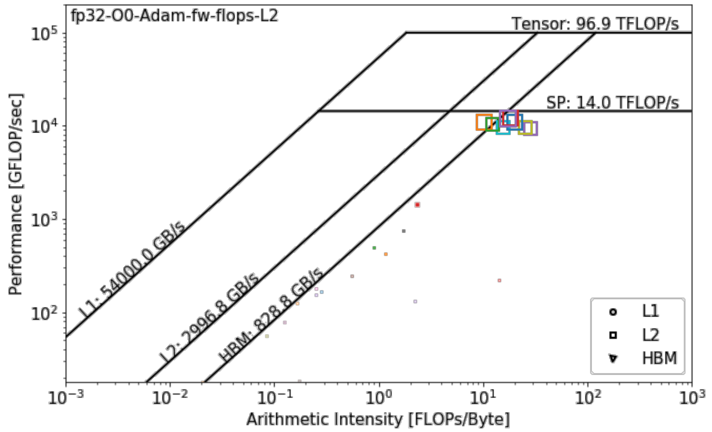


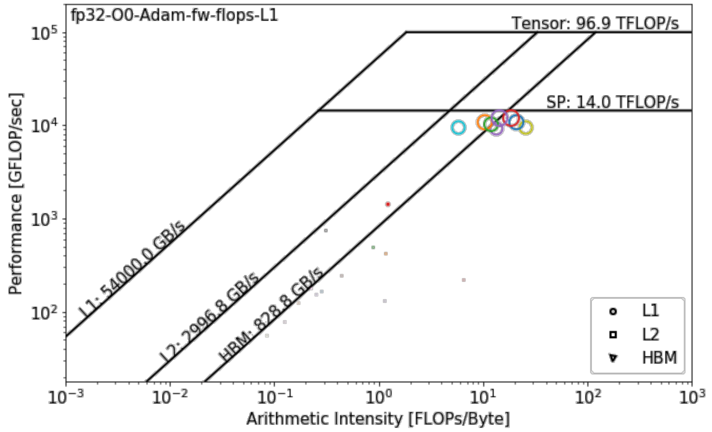


- **Separate HBM, L2, L1 charts**
- Marker size based on percentage of time, **kernel count**, FLOPs, etc
- AMP O0 vs O1 (CUDA core vs Tensor Core)

Example 2. PyTorch climate seg.

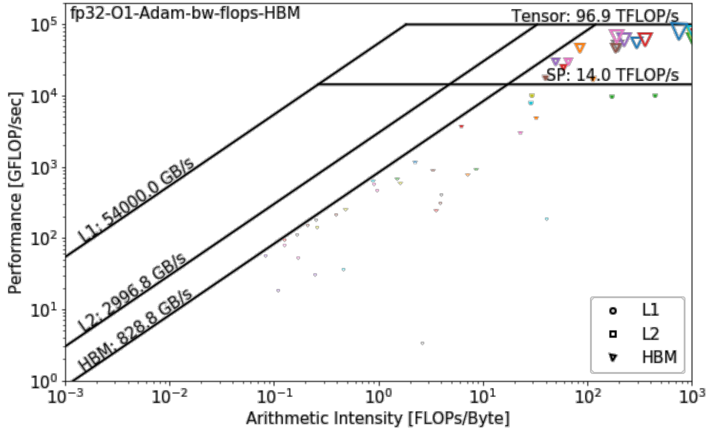


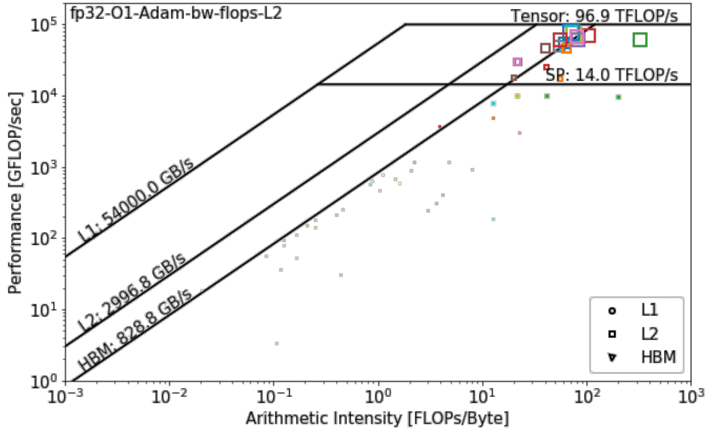


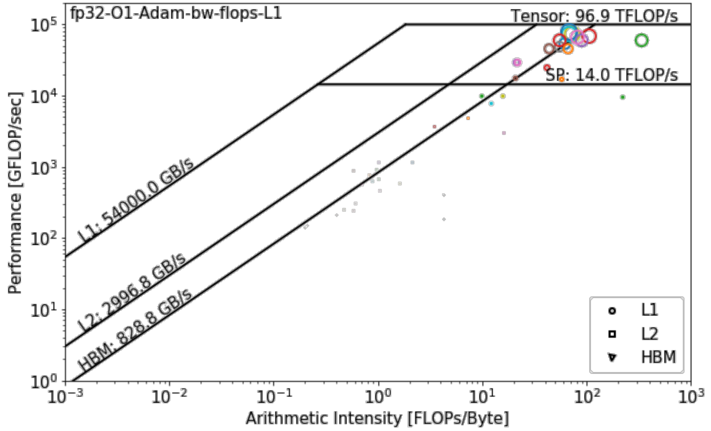


- **Separate HBM, L2, L1 charts**
- Marker size based on percentage of time, kernel count, **FLOPs**, etc
- AMP O0 vs O1 (CUDA core vs Tensor Core)

Example 2. PyTorch climate seg.





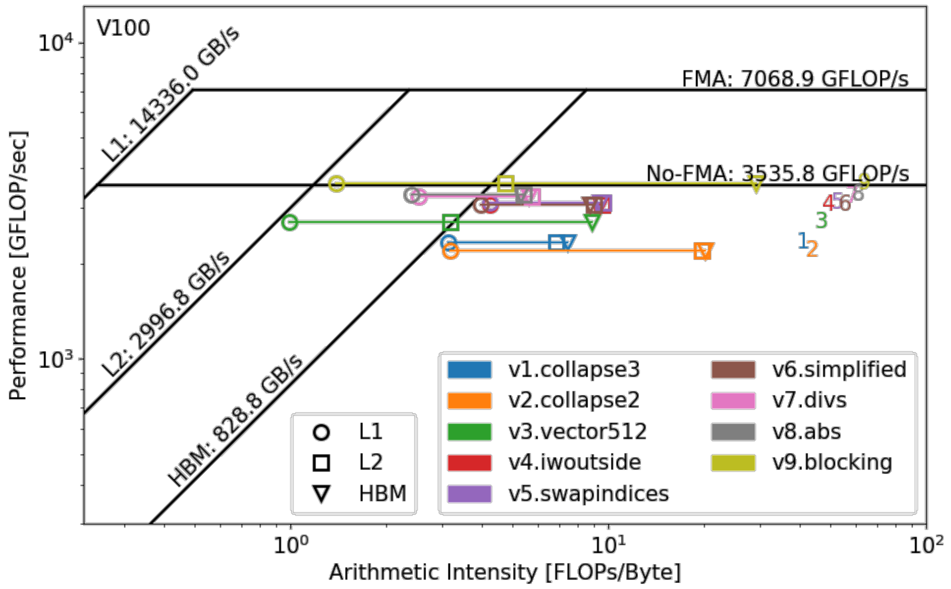


- **Separate HBM, L2, L1 charts**
- Marker size based on percentage of time, kernel count, FLOPs, etc
- AMP O0 vs O1 (**CUDA core vs Tensor Core**)

Example 3. GPP

Show optimization path

1. baseline collapse(3)
2. move n1_loc in and change to collapse(2)
3. vector_length 128 to 512
4. replace div by rcp
5. …



Open questions/issues

- Over-reporting when active SMs < 80 ?
- Integer overflow for _red.sum metrics in Nsight Compute?
- How do we combine ceilings for mixed-precision Rooflines? overlapping pipelines?
  - current section files are based on precisions (dp, sp, hp, tensor)
- Metrics for integer Roofline, power Roofline, and instruction Roofline?
- How to treat 0-AI kernels? instruction Roofline?
- More elegant section files for Tensor Core? 512 repetitions of the same metric.
- ??

NERSC

Thank You