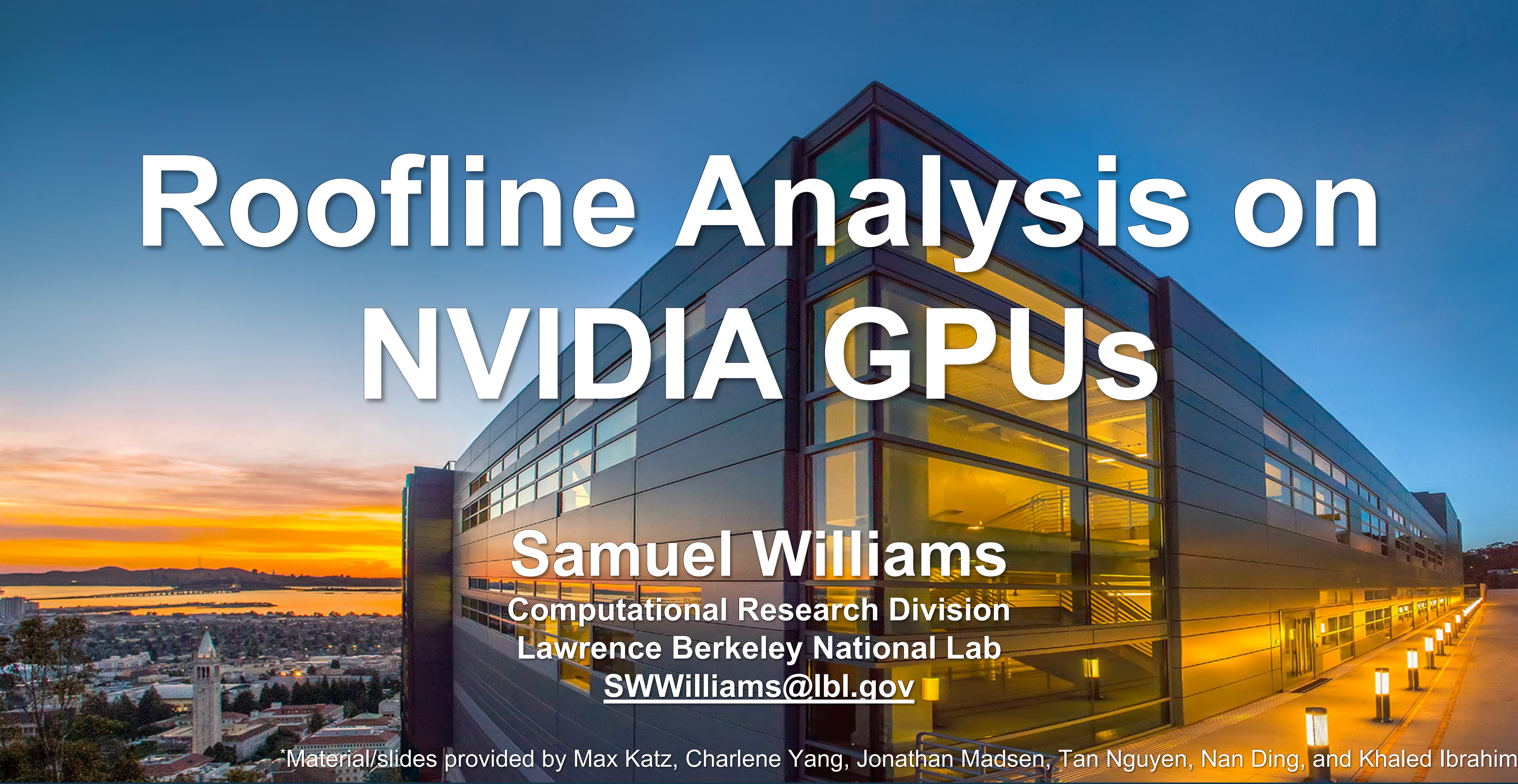


Roofline Analysis on NVIDIA GPUs

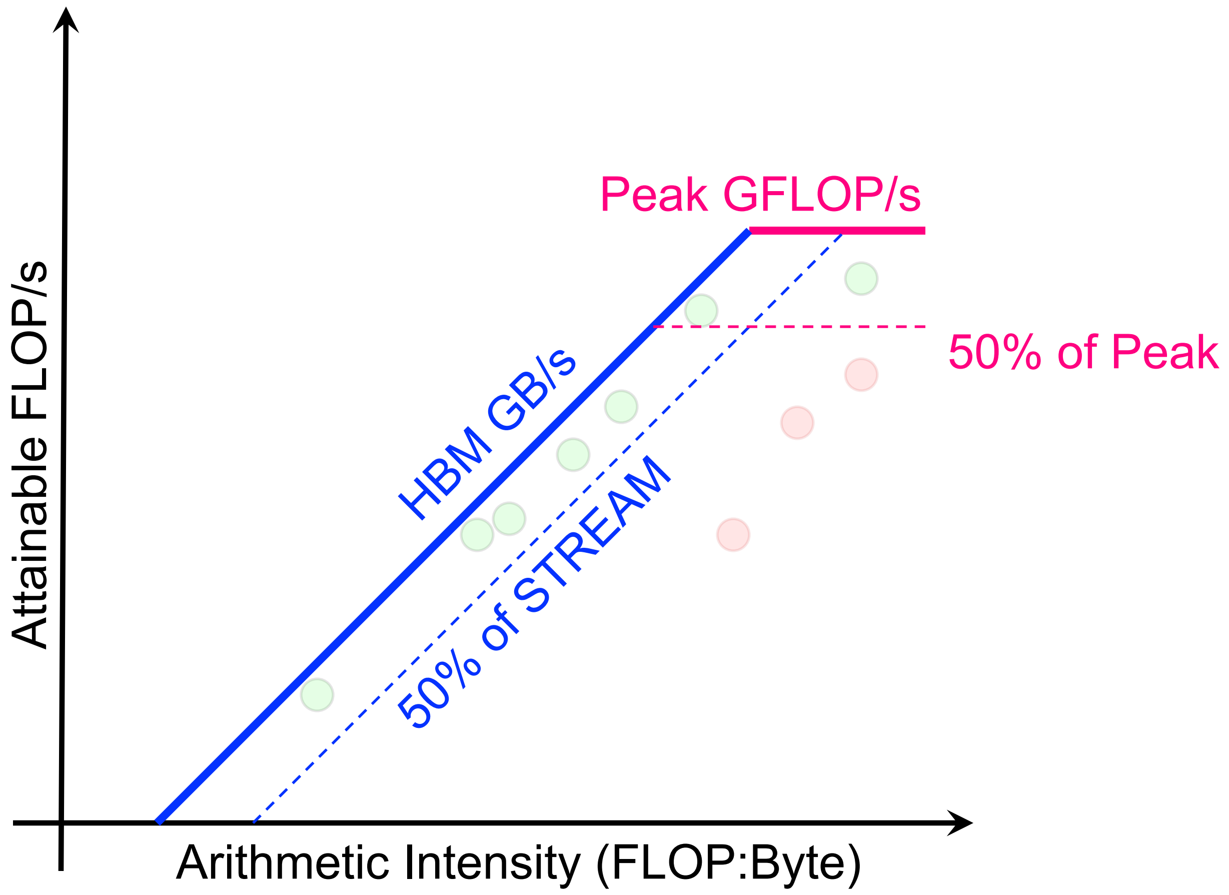
Samuel Williams

Computational Research Division
Lawrence Berkeley National Lab
SWWilliams@lbl.gov

*Material/slides provided by Max Katz, Charlene Yang, Jonathan Madsen, Tan Nguyen, Nan Ding, and Khaled Ibrahim

Reminder: Roofline is made of two components

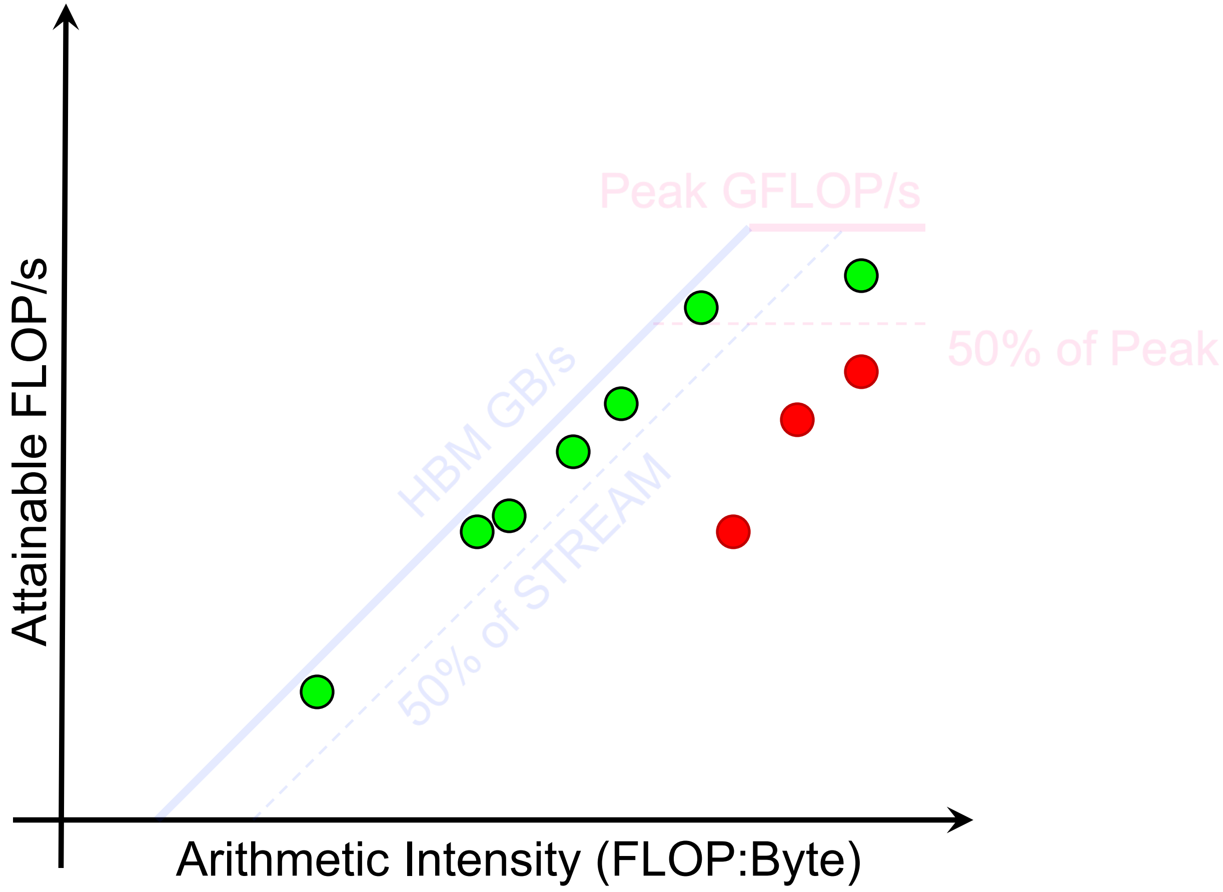
- Machine Model
  - Lines defined by peak GB/s and GF/s (**Benchmarking**)
  - Unique to each architecture
  - Common to all apps on that architecture



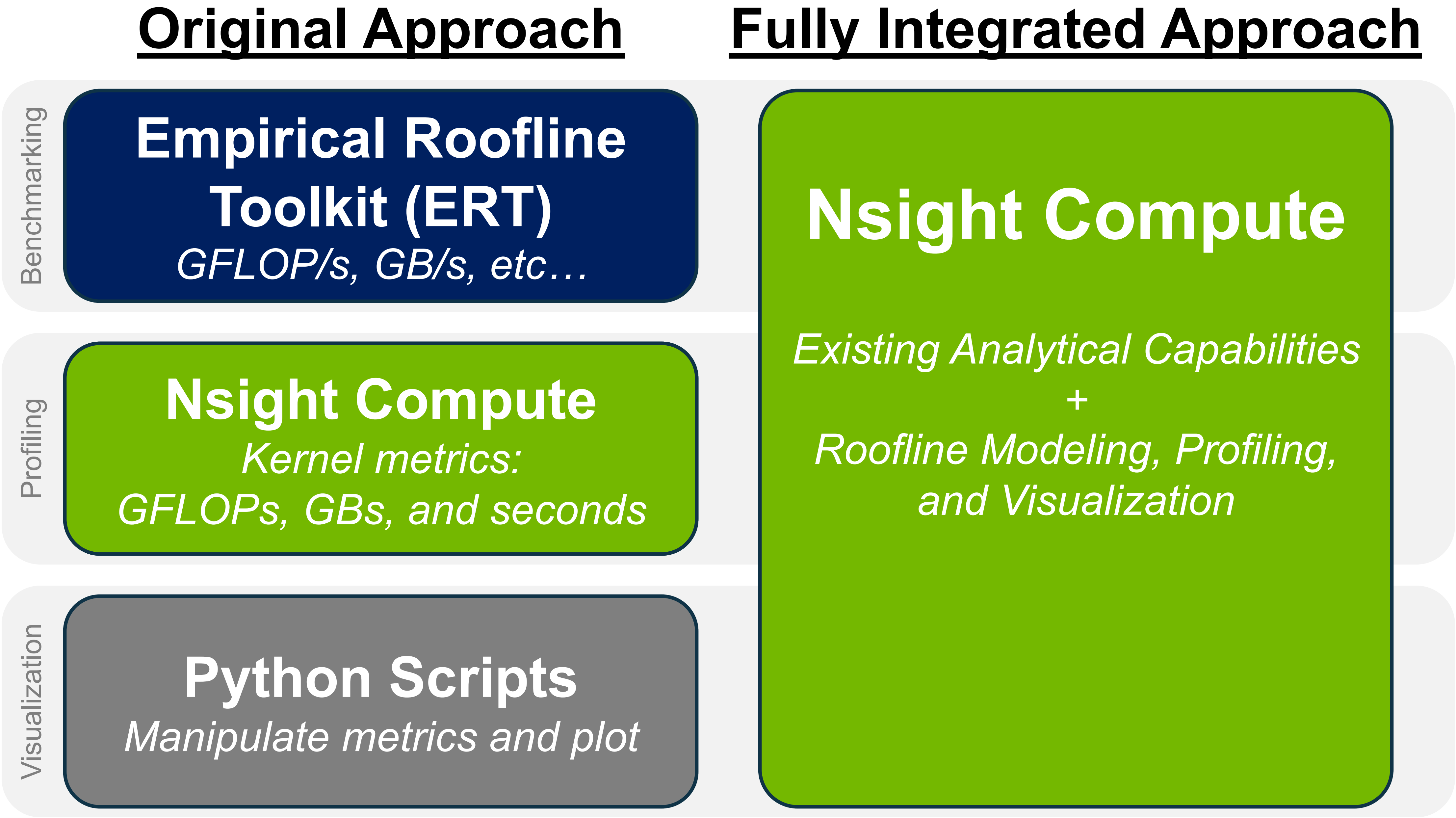
Reminder: Roofline is made of two components

- Machine Model
  - Lines defined by peak GB/s and GF/s (**Benchmarking**)
  - Unique to each architecture
  - Common to all apps on that architecture
- Application Characteristics
  - Dots defined by application GFLOPs, GBs, and run time

    (**Application Instrumentation**)
  - Unique to each application
  - Unique to each architecture



Two Approaches:



Original Approach

Benchmarking

Empirical Roofline Toolkit (ERT)
GFLOP/s, GB/s, etc…

Profiling

Nsight Compute
Kernel metrics:
GFLOPs, GBs, and seconds

Visualization

Python Scripts
Manipulate metrics and plot

Fully Integrated Approach

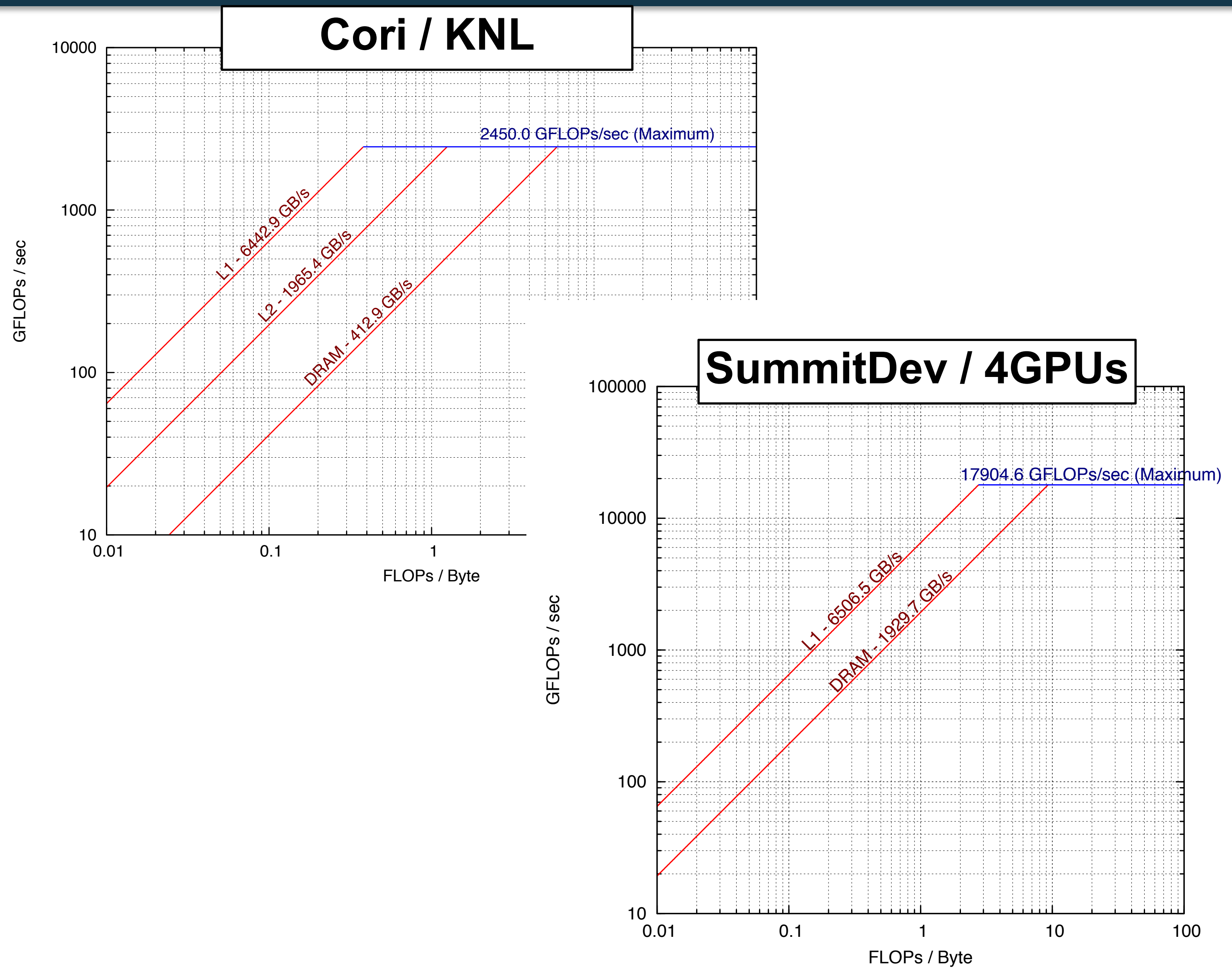
Nsight Compute

Existing Analytical Capabilities
+
Roofline Modeling, Profiling, and Visualization

Empirical Roofline Toolkit (ERT)

Machine Characterization

- **"Theoretical Performance"** numbers can be highly optimistic…
  - Pin BW vs. sustained bandwidth
  - TurboMode / Underclock for AVX
  - compiler failings on high-AI loops.
- LBL developed the Empirical Roofline Toolkit (ERT)…
  - Characterize CPU/GPU systems
  - Peak Flop rates
  - Bandwidths for each level of memory
  - **MPI+OpenMP/CUDA == multiple GPUs**
- Provides a sanity check on programmers, compilers, vendors



https://bitbucket.org/berkeleylab/cs-roofline-toolkit
https://github.com/cyanguwa/nersc-roofline
https://crd.lbl.gov/departments/computer-science/PAR/research/roofline

ERT Configuration Files

Kernel.c
- actual compute
- customizable

Driver.c
- setup
- call kernels
- loop over parameters

config script
- set up ranges of parameters

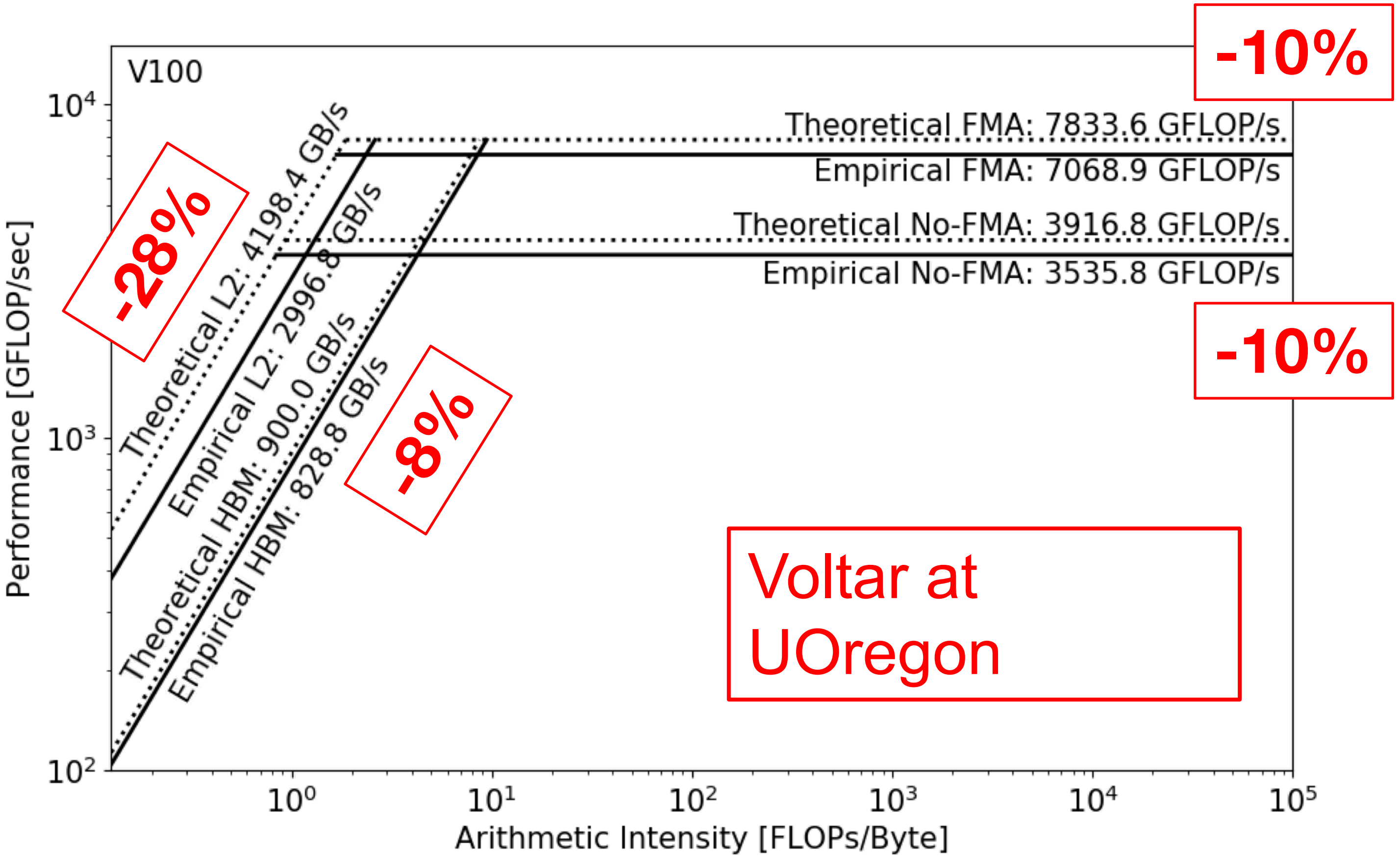
job script
- submit the job and run it

Empirical Roofline Toolkit (ERT).
https://bitbucket.org/berkeleylab/cs-roofline-toolkit/

ERT on NVIDIA GPUs

- Last level of memory is ‘DRAM’ (ERT calls HBM DRAM on V100)
- Enumerates all detected caches as L1, L2, etc…
- Uncacheable memory/WT caches are not detected (ERT misses L1 on V100 and calls the L2 the L1)

- Empirical ceilings are 8-28% lower than the theoretical numbers.



Voltar at UOregon

Empirical Roofline Toolkit (ERT). https://bitbucket.org/berkeleylab/cs-roofline-toolkit/

Profiling with ~~NVProf~~ Nsight Compute

Application Characterization with Nsight (1)

Usage:

```
ncu -k [regexp] --metrics [metrics] --csv ./application
```

Kernel Run time:

- Time per invocation of a kernel:

  ```
  sm__cycles_elapsed.avg / sm__cycles_elapsed.avg.per_second
  ```

Application Characterization with Nsight (2)

#FLOPs:

- For {Double, Single, Half} precision, sum the following metrics:

```
  sm__sass_thread_inst_executed_op_{d,f,h}add_pred_on.sum +
  sm__sass_thread_inst_executed_op_{d,f,h}mul_pred_on.sum
2*sm__sass_thread_inst_executed_op_{d,f,h}fma_pred_on.sum
```

- To calculate FLOPs from Tensor Cores (Volta):

```
512*sm__inst_executed_pipe_tensor.sum
```

- (far more accurate than NVProf’s discretized tensor utilization)

Application Characterization with Nsight (2)

#Bytes

- Measure bytes for each level of the memory hierarchy
- Scale transactions where necessary

Level	Metrics
L1 Cache	`l1tex__t_bytes.sum`
Shared Memory (included in L1)	`(l1tex__data_pipe_lsu_wavefronts_mem_shared_op_ld.sum + l1tex__data_pipe_lsu_wavefronts_mem_shared_op_st.sum)*32`
Atomics (included in L1)	`(l1tex__t_set_accesses_pipe_lsu_mem_global_op_atom.sum + l1tex__t_set_accesses_pipe_lsu_mem_global_op_red.sum)*32`
L2 Cache	`lts__t_bytes.sum`
Device Memory	`dram__bytes.sum`
System Memory (PCIe)	`(lts__t_sectors_aperture_sysmem_op_read.sum + lts__t_sectors_aperture_sysmem_op_write.sum)*32`

Visualization

You must combine ERT and Nsight data

- ERT provides compute (horizontal lines) and bandwidth (diagonal lines) ceilings
- NVProf data must be manipulated

$$\underset{\text{(x coordinate)}}{\textbf{AI}} = \frac{\text{NVprof GFLOPs}}{\text{NVprof GBytes}} \qquad \underset{\text{(y coordinate)}}{\textbf{GFLOP/s}} = \frac{\text{NVprof GFLOPs}}{\text{NVprof seconds}}$$

- Plot Using Python script. e.g.

`https://gitlab.com/NERSC/roofline-on-nvidia-gpus/-/tree/master/custom-scripts`

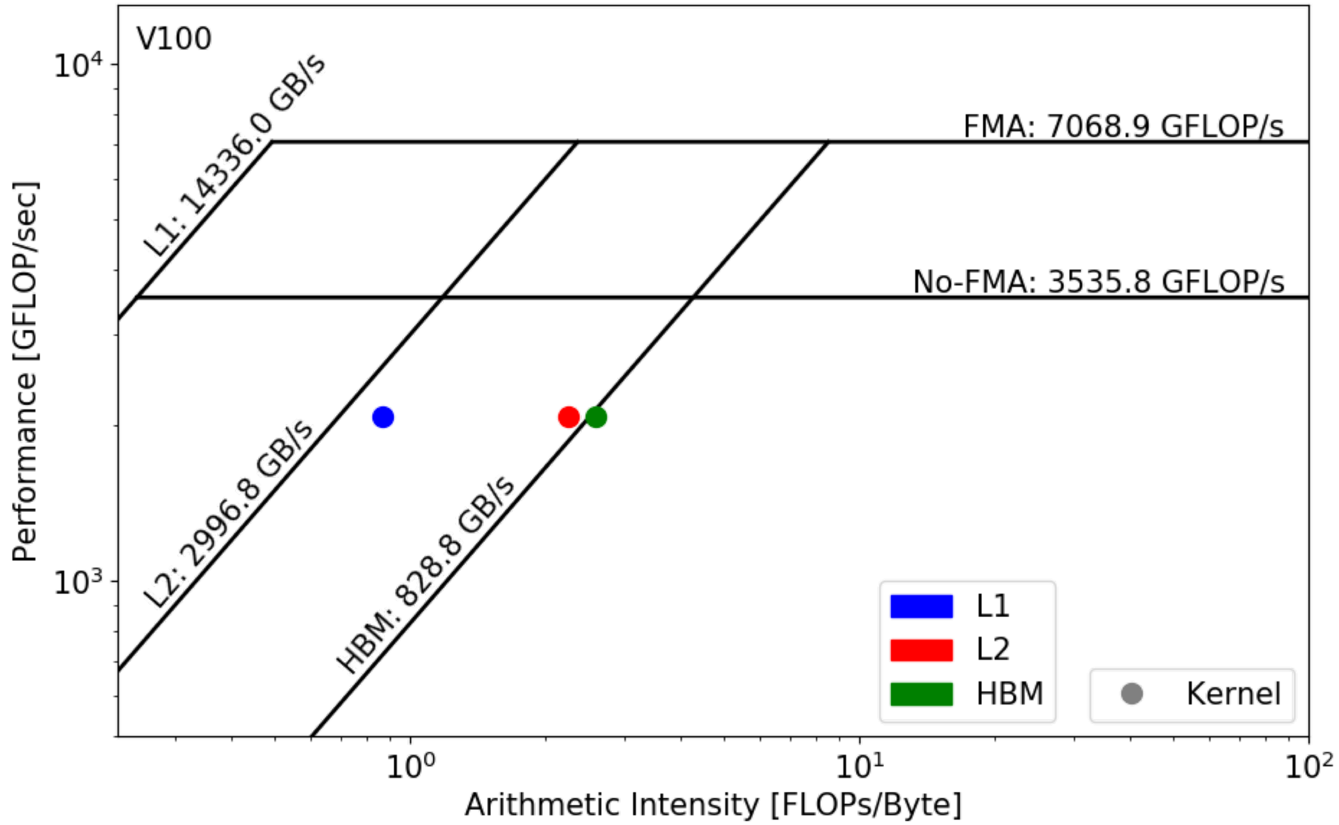
You must combine ERT and Nsight data

```
% cat data.txt
```

```
# all data is space delimited
memroofs 14336.0 2996.8 828.758
mem_roof_names ‘L1’ ‘L2’ ‘HBM’
comproofs 7068.86 3535.79
comp_roof_names ‘FMA’ ‘No-FMA’

# omit the following if only plotting roofs
# AI: arithmetic intensity; GFLOPs: performance
AI 0.87 2.25 2.58
GFLOPs 2085.756683
labels ‘Kernel’
```

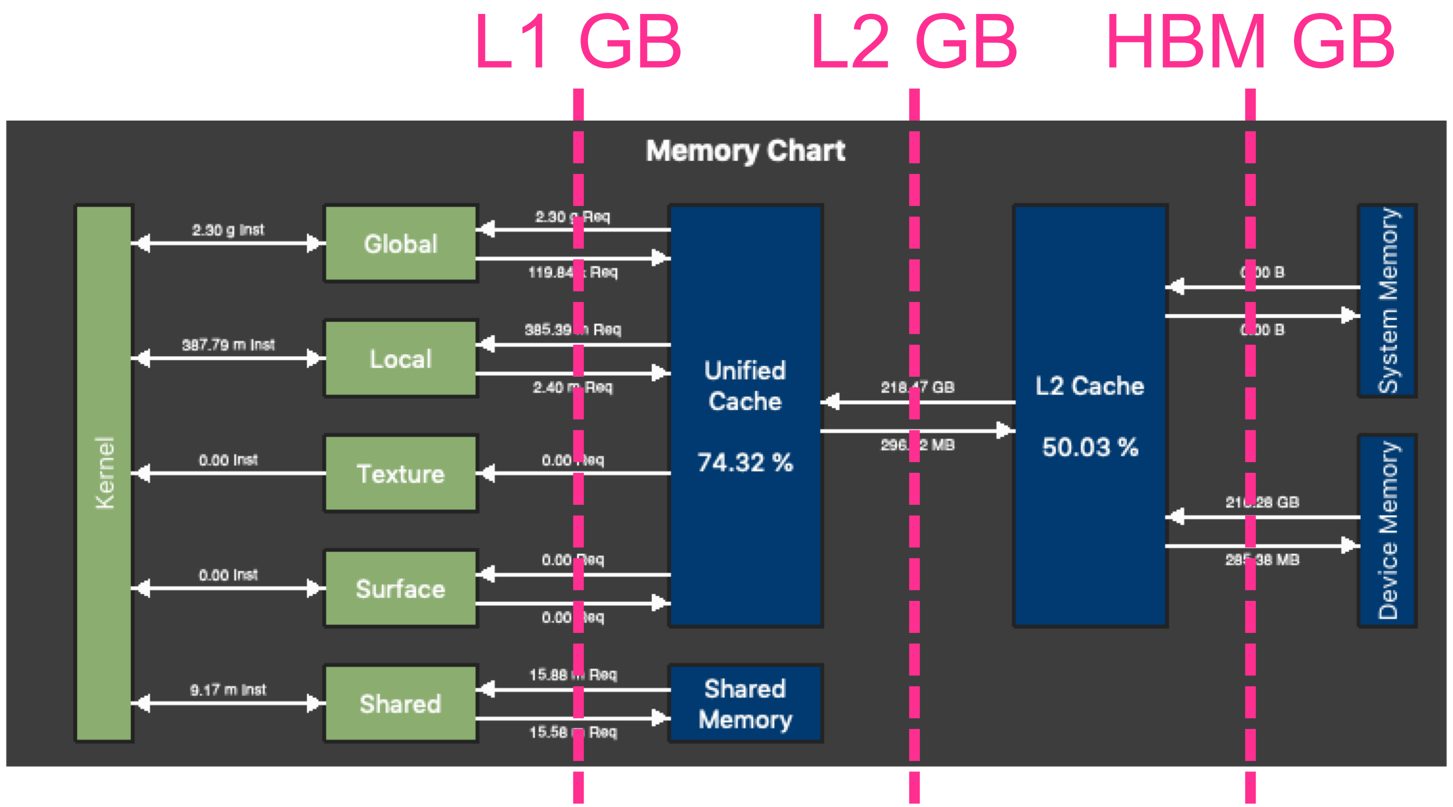
```
% plot_roofline.py data.txt
```



Nsight Compute (Roofline integration)

Nsight has integrated Roofline Analysis

- Nsight’s view of V100’s memory architecture
  - Green boxes are logical regions
  - Blue boxes are physical levels
- Roofline is calculated based on the data movement between physical levels

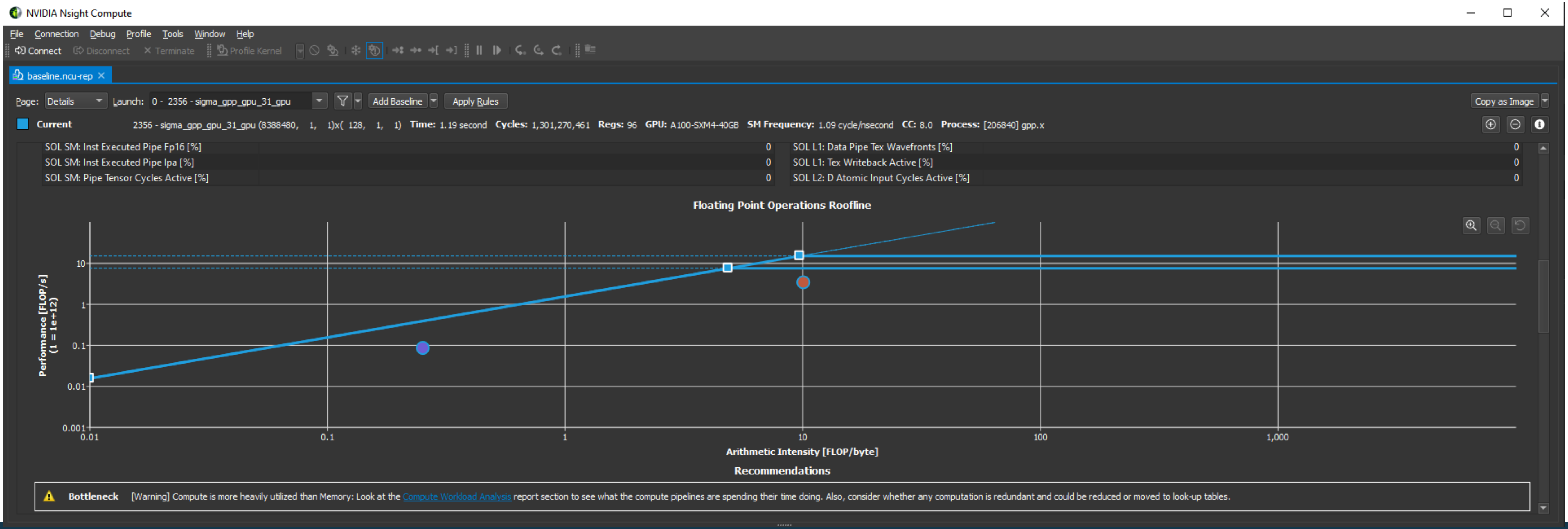


- Roofline in GUI:

```
ncu -k [kernel] --metrics [metrics]./application
```

Nsight has integrated Roofline Analysis

- Automatically plots Roofline (DRAM shown below)
- Allows you to compare multiple versions of a kernel on the same Roofline (tracks progress towards optimality)


NVIDIA Nsight Compute
File Connection Debug Profile Tools Window Help
Connect Disconnect Terminate Profile Kernel
baseline.ncu-rep
Add Baseline
Apply Rules
Copy as Image
Floating Point Operations Roofline
Performance [FLOP/s] (1 = 1e+12)
10
1
0.1
0.01
0.001
0.01
0.1
1
10
100
1,000
Arithmetic Intensity [FLOP/byte]
Recommendations

Page: Details Launch: 0 - 2356 - sigma_gpp_gpu_31_gpu

Current 2356 - sigma_gpp_gpu_31_gpu (8388480, 1, 1)x(128, 1, 1) **Time:** 1.19 second **Cycles:** 1,301,270,461 **Regs:** 96 **GPU:** A100-SXM4-40GB **SM Frequency:** 1.09 cycle/nsecond **CC:** 8.0 **Process:** [206840] gpp.x

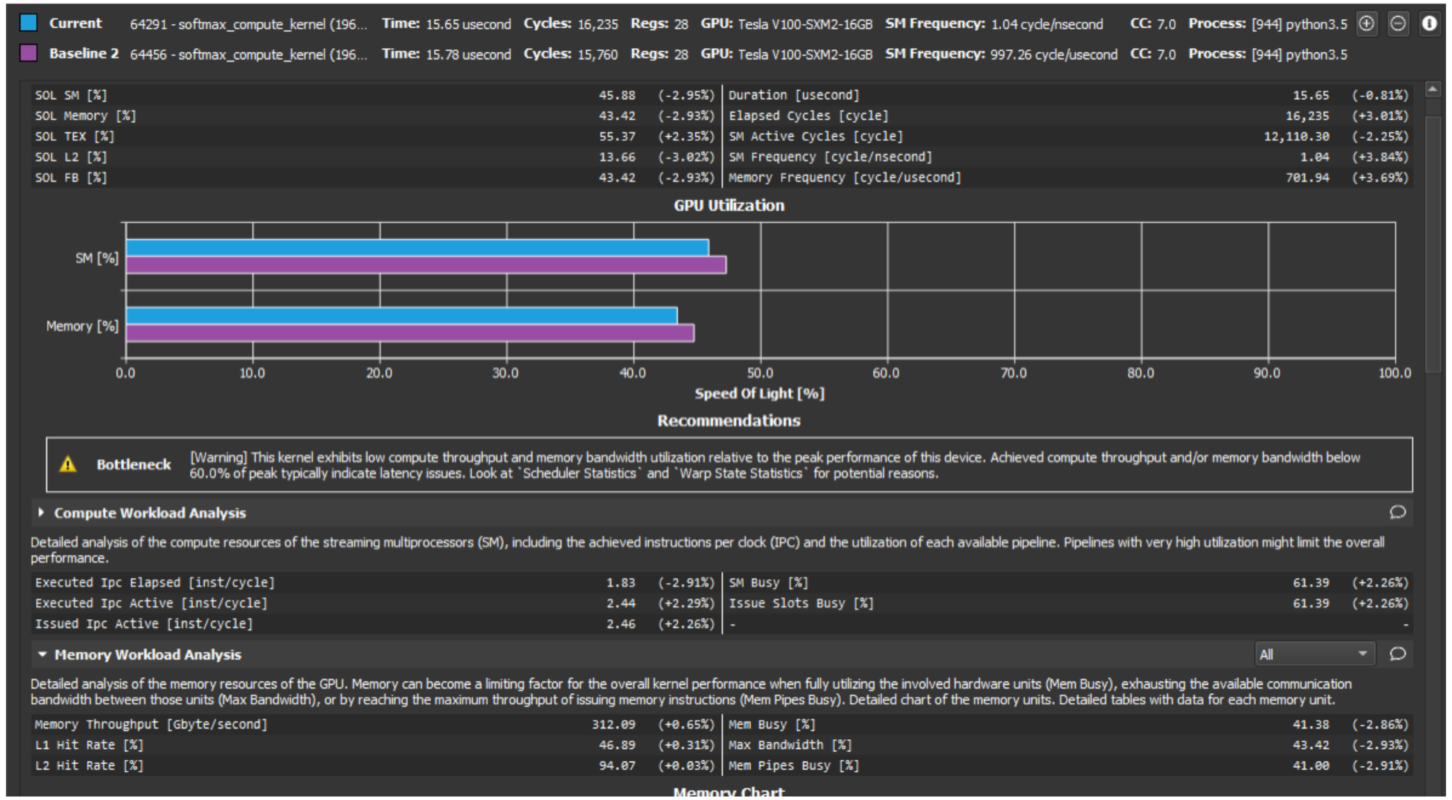
SOL SM: Inst Executed Pipe Fp16 [%]	0	SOL L1: Data Pipe Tex Wavefronts [%]	0
SOL SM: Inst Executed Pipe Ipa [%]	0	SOL L1: Tex Writeback Active [%]	0
SOL SM: Pipe Tensor Cycles Active [%]	0	SOL L2: D Atomic Input Cycles Active [%]	0

⚠ **Bottleneck** [Warning] Compute is more heavily utilized than Memory: Look at the Compute Workload Analysis report section to see what the compute pipelines are spending their time doing. Also, consider whether any computation is redundant and could be reduced or moved to look-up tables.

Complements Existing SOL and PTX analysis

- speed-of-light analysis comparisons with baseline

- Source/PTX/SASS analysis and correlation



```
Current     64291 - softmax_compute_kernel (196...  Time: 15.65 usecond  Cycles: 16,235  Regs: 28  GPU: Tesla V100-SXM2-16GB  SM Frequency: 1.04 cycle/nsecond      CC: 7.0  Process: [944] python3.5
Baseline 2  64456 - softmax_compute_kernel (196...  Time: 15.78 usecond  Cycles: 15,760  Regs: 28  GPU: Tesla V100-SXM2-16GB  SM Frequency: 997.26 cycle/usecond  CC: 7.0  Process: [944] python3.5
```

```
SOL SM [%]              45.88   (-2.95%) | Duration [usecond]                    15.65  (-0.81%)
SOL Memory [%]          43.42   (-2.93%) | Elapsed Cycles [cycle]               16,235  (+3.01%)
SOL TEX [%]             55.37   (+2.35%) | SM Active Cycles [cycle]          12,110.30  (-2.25%)
SOL L2 [%]              13.66   (-3.02%) | SM Frequency [cycle/nsecond]           1.04  (+3.84%)
SOL FB [%]              43.42   (-2.93%) | Memory Frequency [cycle/usecond]     701.94  (+3.69%)
```

Recommendations

⚠ Bottleneck [Warning] This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of this device. Achieved compute throughput and/or memory bandwidth below 60.0% of peak typically indicate latency issues. Look at `Scheduler Statistics` and `Warp State Statistics` for potential reasons.

▸ Compute Workload Analysis

Detailed analysis of the compute resources of the streaming multiprocessors (SM), including the achieved instructions per clock (IPC) and the utilization of each available pipeline. Pipelines with very high utilization might limit the overall performance.

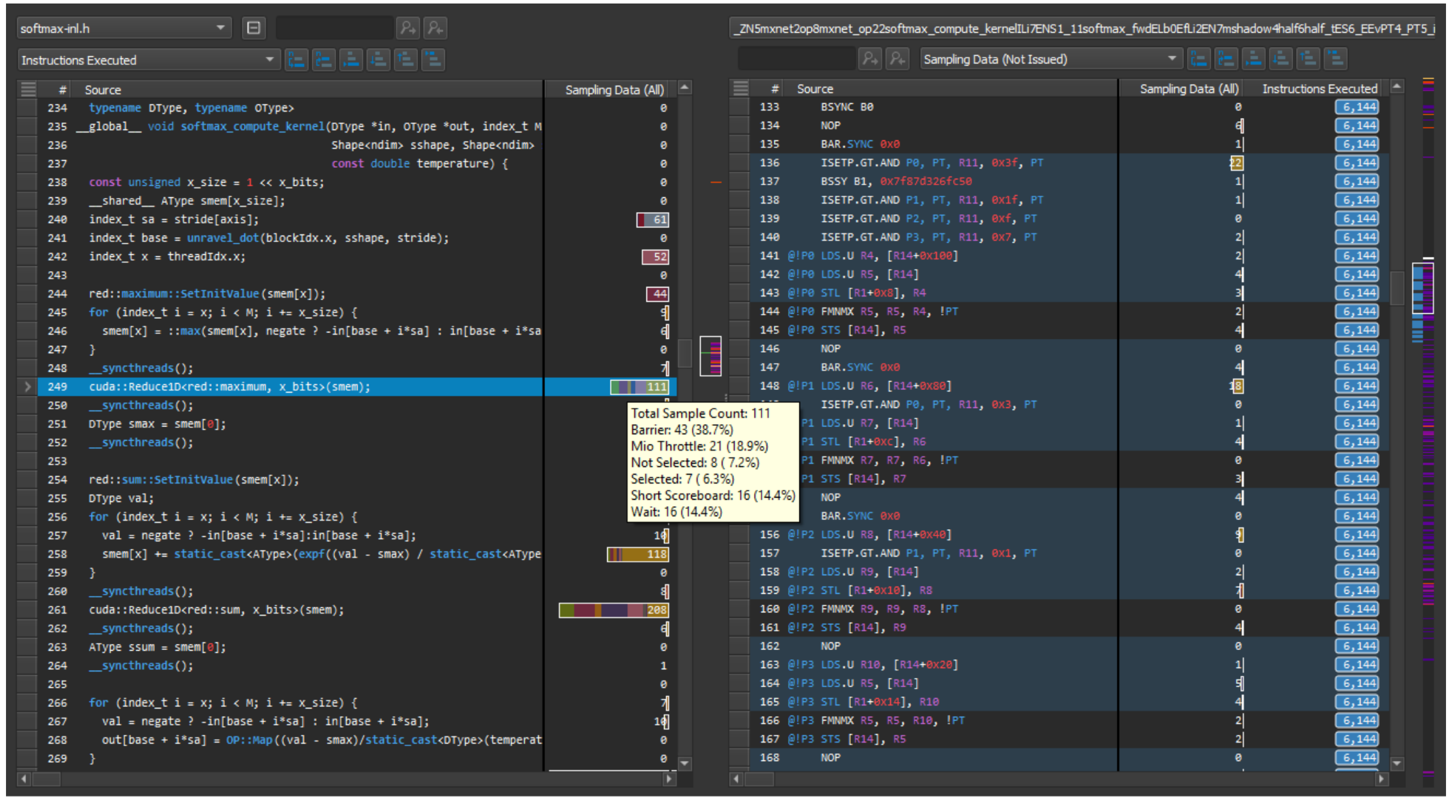
```
Executed Ipc Elapsed [inst/cycle]     1.83   (-2.91%) | SM Busy [%]            61.39  (+2.26%)
Executed Ipc Active [inst/cycle]      2.44   (+2.29%) | Issue Slots Busy [%]   61.39  (+2.26%)
Issued Ipc Active [inst/cycle]        2.46   (+2.26%) | -                          -
```

▾ Memory Workload Analysis All

Detailed analysis of the memory resources of the GPU. Memory can become a limiting factor for the overall kernel performance when fully utilizing the involved hardware units (Mem Busy), exhausting the available communication bandwidth between those units (Max Bandwidth), or by reaching the maximum throughput of issuing memory instructions (Mem Pipes Busy). Detailed chart of the memory units. Detailed tables with data for each memory unit.

```
Memory Throughput [Gbyte/second]    312.09   (+0.65%) | Mem Busy [%]         41.38  (-2.86%)
L1 Hit Rate [%]                      46.89   (+0.31%) | Max Bandwidth [%]    43.42  (-2.93%)
L2 Hit Rate [%]                      94.07   (+0.03%) | Mem Pipes Busy [%]   41.00  (-2.91%)
```

Memory Chart



softmax-inl.h

Instructions Executed

```
 #   Source                                                                        Sampling Data (All)
234   typename DType, typename OType>                                                    0
235 __global__ void softmax_compute_kernel(DType *in, OType *out, index_t M             0
236                                         Shape<ndim> sshape, Shape<ndim>              0
237                                         const double temperature) {                  0
238   const unsigned x_size = 1 << x_bits;                                               0
239   __shared__ AType smem[x_size];                                                     0
240   index_t sa = stride[axis];                                                        61
241   index_t base = unravel_dot(blockIdx.x, sshape, stride);                            0
242   index_t x = threadIdx.x;                                                          52
243                                                                                      0
244   red::maximum::SetInitValue(smem[x]);                                              44
245   for (index_t i = x; i < M; i += x_size) {                                          9
246     smem[x] = ::max(smem[x], negate ? -in[base + i*sa] : in[base + i*sa             6
247   }                                                                                  0
248   __syncthreads();                                                                   7
249   cuda::Reduce1D<red::maximum, x_bits>(smem);                                      111
250   __syncthreads();
251   DType smax = smem[0];
252   __syncthreads();
253
254   red::sum::SetInitValue(smem[x]);
255   DType val;
256   for (index_t i = x; i < M; i += x_size) {
257     val = negate ? -in[base + i*sa]:in[base + i*sa];                                10
258     smem[x] += static_cast<AType>(expf((val - smax) / static_cast<AType            118
259   }                                                                                  0
260   __syncthreads();                                                                   8
261   cuda::Reduce1D<red::sum, x_bits>(smem);                                          208
262   __syncthreads();                                                                   6
263   AType ssum = smem[0];                                                              0
264   __syncthreads();                                                                   1
265                                                                                      0
266   for (index_t i = x; i < M; i += x_size) {                                          7
267     val = negate ? -in[base + i*sa] : in[base + i*sa];                              10
268     out[base + i*sa] = OP::Map((val - smax)/static_cast<DType>(temperat             0
269   }                                                                                  0
```

Total Sample Count: 111
Barrier: 43 (38.7%)
Mio Throttle: 21 (18.9%)
Not Selected: 8 (7.2%)
Selected: 7 (6.3%)
Short Scoreboard: 16 (14.4%)
Wait: 16 (14.4%)

_ZN5mxnet2op8mxnet_op22softmax_compute_kernelILi7ENS1_11softmax_fwdELb0EfLi2EN7mshadow4half6half_tES6_EEvPT4_PT5_i

Sampling Data (Not Issued)

```
 #   Source                                  Sampling Data (All)   Instructions Executed
133       BSYNC B0                                    0                 6,144
134       NOP                                         6                 6,144
135       BAR.SYNC 0x0                                1                 6,144
136       ISETP.GT.AND P0, PT, R11, 0x3f, PT         22                 6,144
137       BSSY B1, 0x7f87d326fc50                     1                 6,144
138       ISETP.GT.AND P1, PT, R11, 0x1f, PT          1                 6,144
139       ISETP.GT.AND P2, PT, R11, 0xf, PT           0                 6,144
140       ISETP.GT.AND P3, PT, R11, 0x7, PT           2                 6,144
141  @!P0 LDS.U R4, [R14+0x100]                       2                 6,144
142  @!P0 LDS.U R5, [R14]                             4                 6,144
143  @!P0 STL [R1+0x8], R4                            3                 6,144
144  @!P0 FMNMX R5, R5, R4, !PT                       2                 6,144
145  @!P0 STS [R14], R5                               4                 6,144
146       NOP                                         0                 6,144
147       BAR.SYNC 0x0                                4                 6,144
148  @!P1 LDS.U R6, [R14+0x80]                       18                 6,144
          ISETP.GT.AND P0, PT, R11, 0x3, PT           0                 6,144
       P1 LDS.U R7, [R14]                             1                 6,144
       P1 STL [R1+0xc], R6                            4                 6,144
       P1 FMNMX R7, R7, R6, !PT                       0                 6,144
       P1 STS [R14], R7                               3                 6,144
          NOP                                         4                 6,144
          BAR.SYNC 0x0                                0                 6,144
156  @!P2 LDS.U R8, [R14+0x40]                        9                 6,144
157       ISETP.GT.AND P1, PT, R11, 0x1, PT           0                 6,144
158  @!P2 LDS.U R9, [R14]                             2                 6,144
159  @!P2 STL [R1+0x10], R8                           7                 6,144
160  @!P2 FMNMX R9, R9, R8, !PT                       0                 6,144
161  @!P2 STS [R14], R9                               4                 6,144
162       NOP                                         0                 6,144
163  @!P3 LDS.U R10, [R14+0x20]                       1                 6,144
164  @!P3 LDS.U R5, [R14]                             5                 6,144
165  @!P3 STL [R1+0x14], R10                          4                 6,144
166  @!P3 FMNMX R5, R5, R10, !PT                      2                 6,144
167  @!P3 STS [R14], R5                               2                 6,144
168       NOP                                         0                 6,144
```

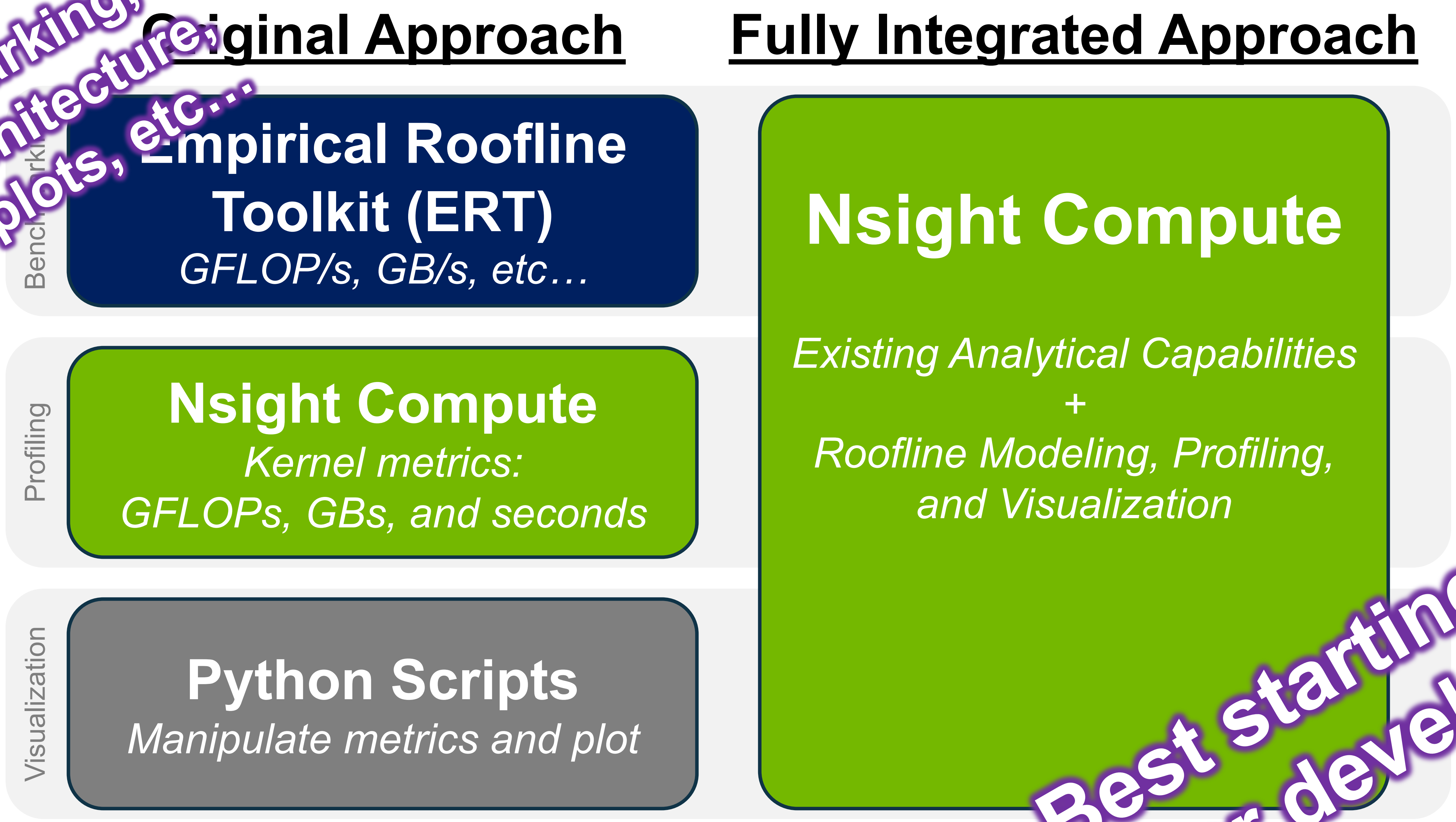
Scripting interface for Custom Rooflines

- Nsight includes a scripting interface where users can define their own custom Rooflines (e.g. hierarchical Roofline)
  https://docs.nvidia.com/nsight-compute/CustomizationGuide/index.html#sections

- NERSC/NVIDIA have provided example scripts:
  https://gitlab.com/NERSC/roofline-on-nvidia-gpus
  e.g.

```
ncu -f -o myprofile --section-folder ../../ncu-section-files
--sectionSpeedOfLight_HierarchicalDoubleRooflineChart ./application
```

Which approach should I use?



Original Approach

Benchmarking

Empirical Roofline Toolkit (ERT)

GFLOP/s, GB/s, etc…

Profiling

Nsight Compute

Kernel metrics:

GFLOPs, GBs, and seconds

Visualization

Python Scripts

Manipulate metrics and plot

Fully Integrated Approach

Nsight Compute

Existing Analytical Capabilities

+

Roofline Modeling, Profiling, and Visualization

Benchmarking, cross-architecture, custom plots, etc…

Best starting point for developers

Questions?

BACKUP

NVProf

(pre-Volta, soon to be deprecated)

Application Characterization with NVProf (1)

Run time:

- 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 Cores:

```
--metrics tensor_precision_fu_utilization
```

Note: integer in the range of 0-10, 0=0, 10=125TFLOP/s; multiply by run time -> #FLOPs

Application Characterization with NVProf (2)

#Bytes

- Measure bytes for each level of the memory hierarchy
- Bytes = (read transactions + write transactions) * transaction size
- Preface with your favor launcher (srun, mpirun, jsrun, etc…)

```
nvprof --kernels [kernel_name] --metrics [metric_name] ./application
```

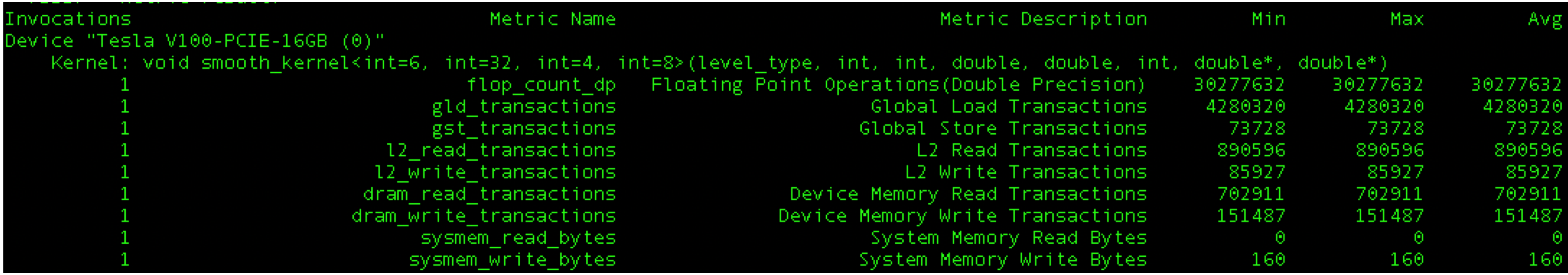
Level	Metrics	Transaction Size
First Level Cache	`gld_transactions, gst_transactions, local_load_transactions, local_store_transactions, atomic_transactions`	32B
Shared Memory	`shared_load_transactions, shared_store_transactions`	128B
Second Level Cache	`l2_read_transactions, l2_write_transactions`	32B
Device Memory	`dram_read_transactions, dram_write_transactions`	32B
System Memory	`system_read_transactions, system_write_transactions`	32B

Application Characterization with NVProf (3)

- You can specify specific context(1), stream(7), and invocation(1) …

  --kernels “1:7:smooth_kernel:1”

- Nominally just get an output table



```
Invocations                               Metric Name                            Metric Description         Min         Max         Avg
Device "Tesla V100-PCIE-16GB (0)"
    Kernel: void smooth_kernel<int=6, int=32, int=4, int=8>(level_type, int, int, double, double, int, double*, double*)
          1                             flop_count_dp   Floating Point Operations(Double Precision)    30277632    30277632    30277632
          1                          gld_transactions                      Global Load Transactions     4280320     4280320     4280320
          1                          gst_transactions                     Global Store Transactions       73728       73728       73728
          1                      l2_read_transactions                          L2 Read Transactions      890596      890596      890596
          1                     l2_write_transactions                         L2 Write Transactions       85927       85927       85927
          1                    dram_read_transactions               Device Memory Read Transactions      702911      702911      702911
          1                   dram_write_transactions              Device Memory Write Transactions      151487      151487      151487
          1                         sysmem_read_bytes                       System Memory Read Bytes           0           0           0
          1                        sysmem_write_bytes                      System Memory Write Bytes         160         160         160
```

- Alternately, you can output to a csv…

  --csv –o nvprof.out