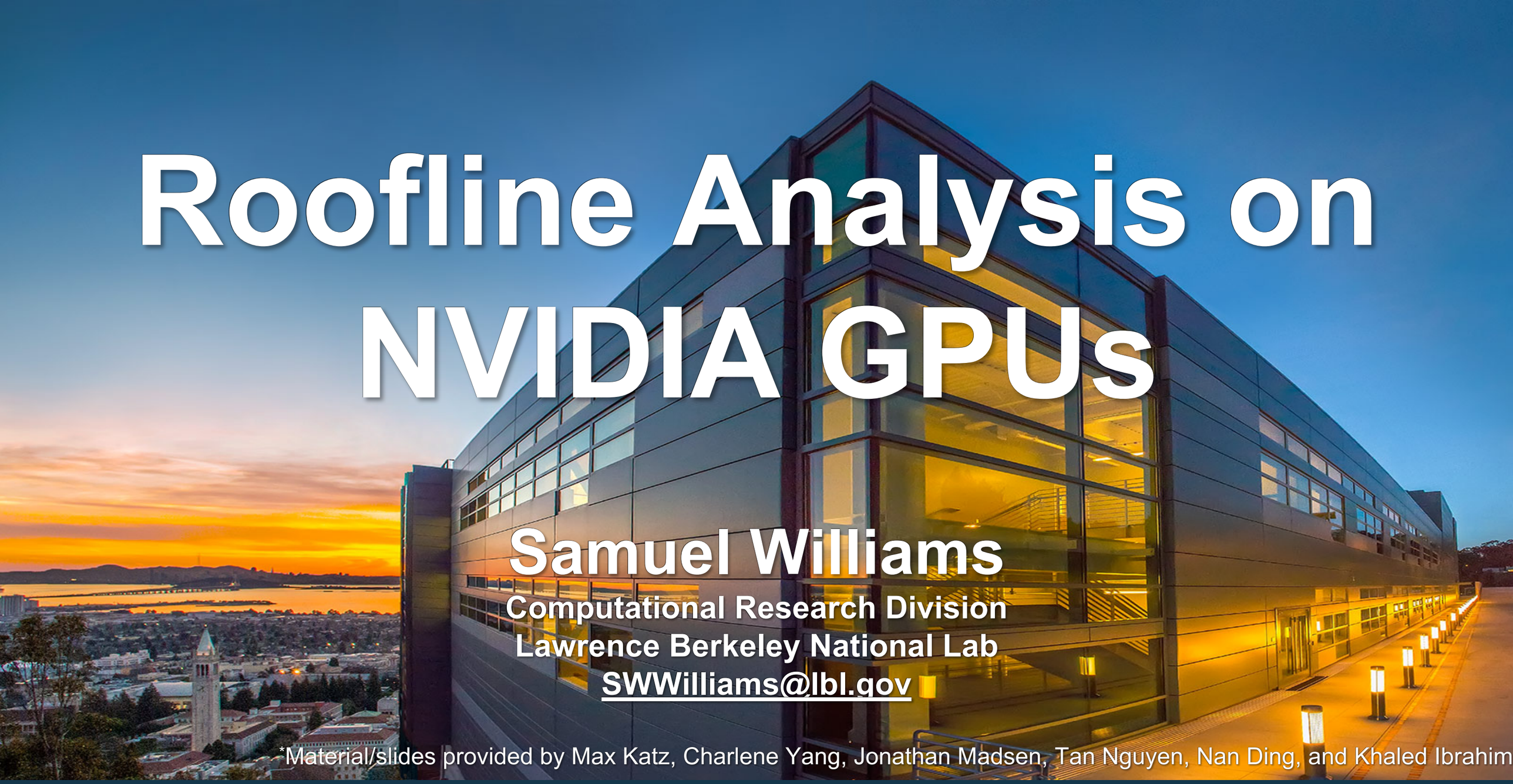


Roofline Analysis on NVIDIA GPUs

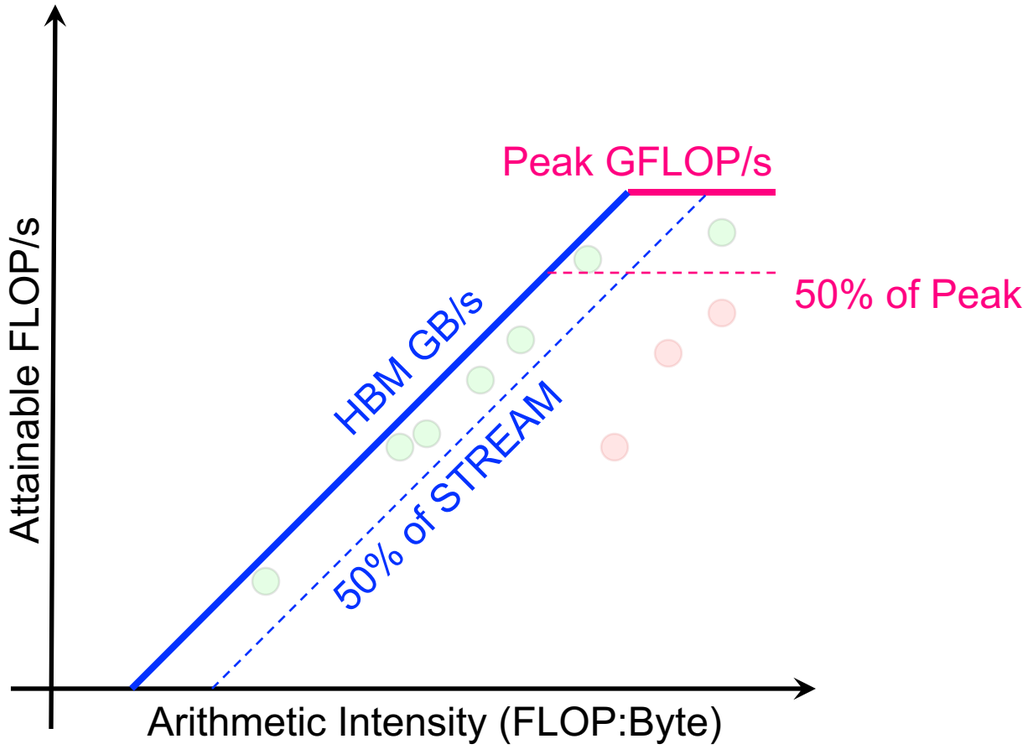
Samuel Williams

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

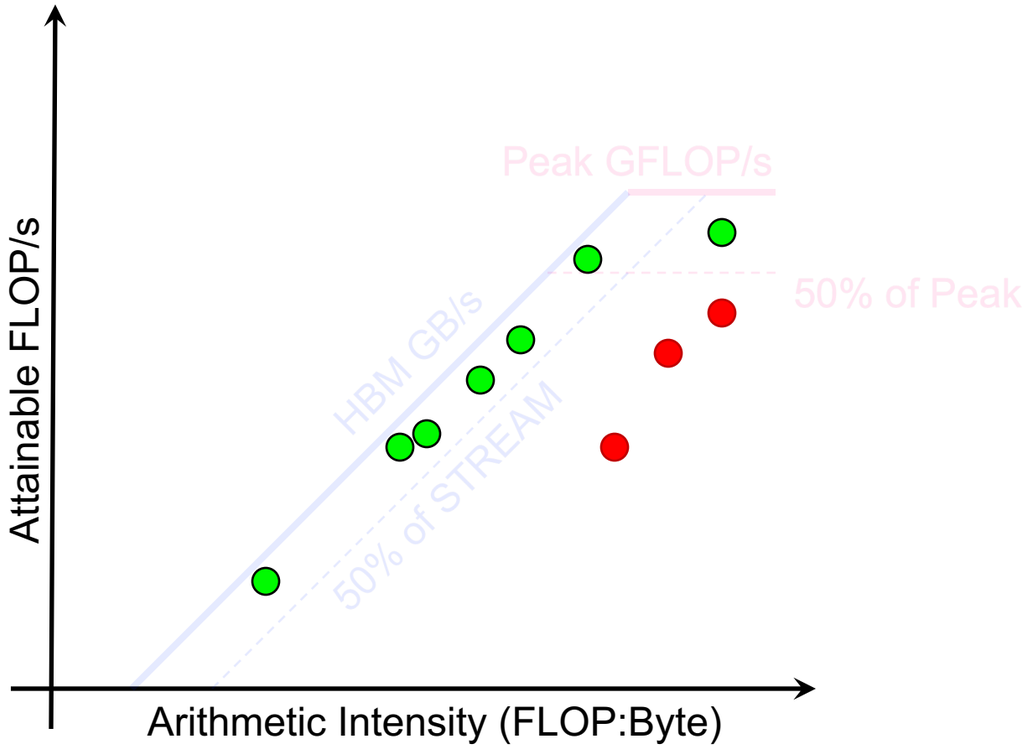
- Machine Model
  - Lines defined by peak GB/s and GF/s (**Benchmarking**)
  - Unique to each architecture
  - Common to all apps on that architecture



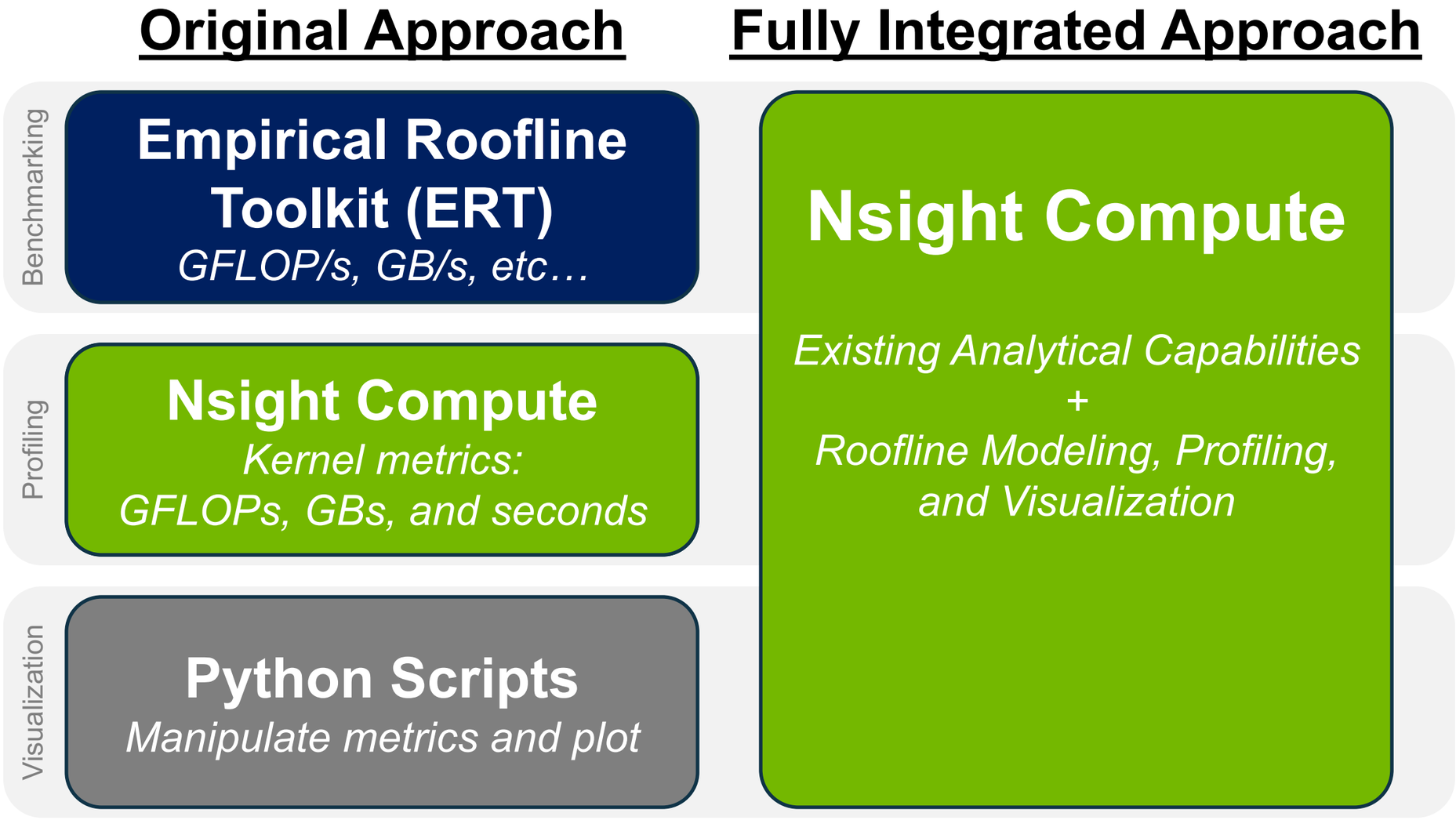
Reminder: Roofline is made of two components

- 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



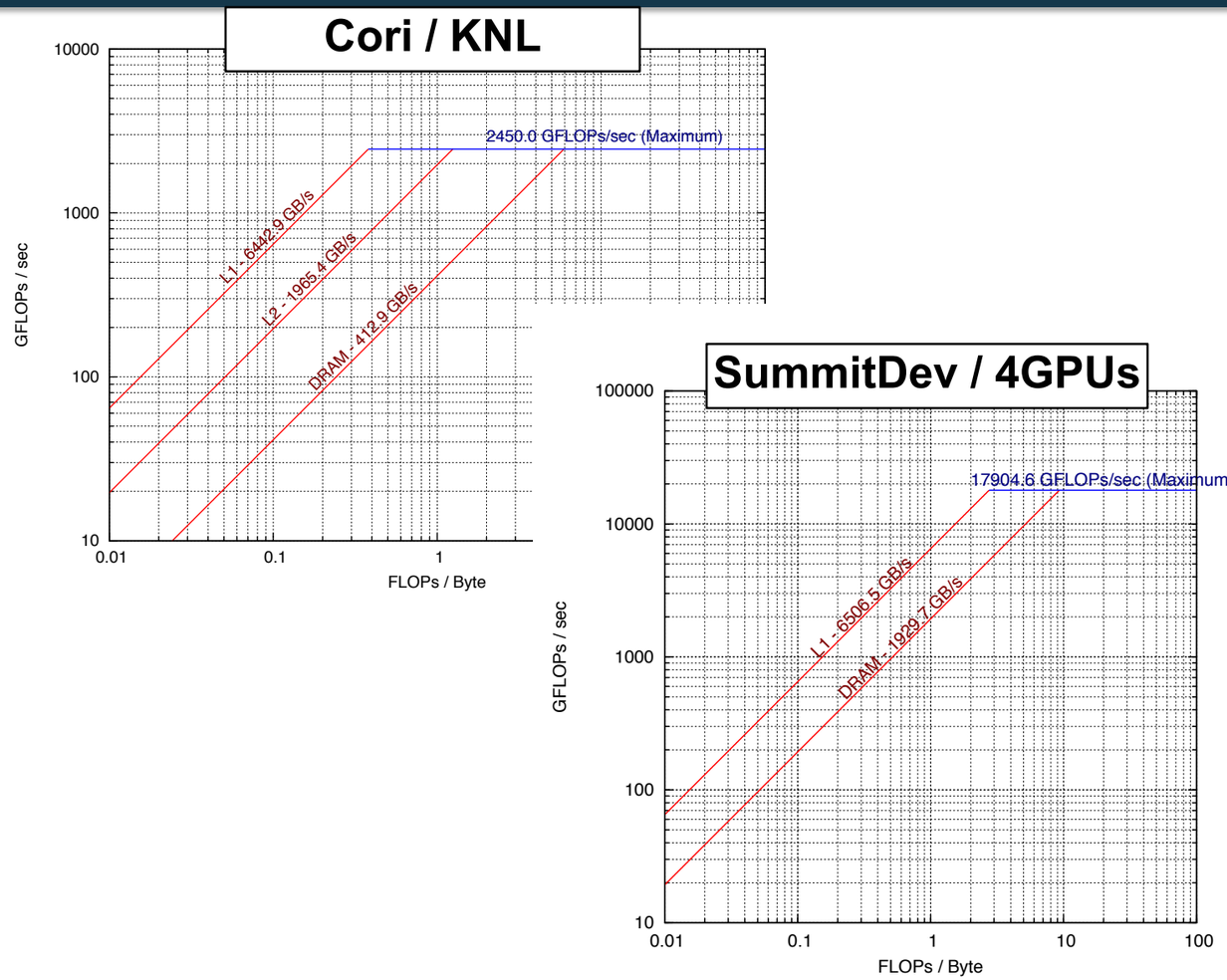
Two Approaches:



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

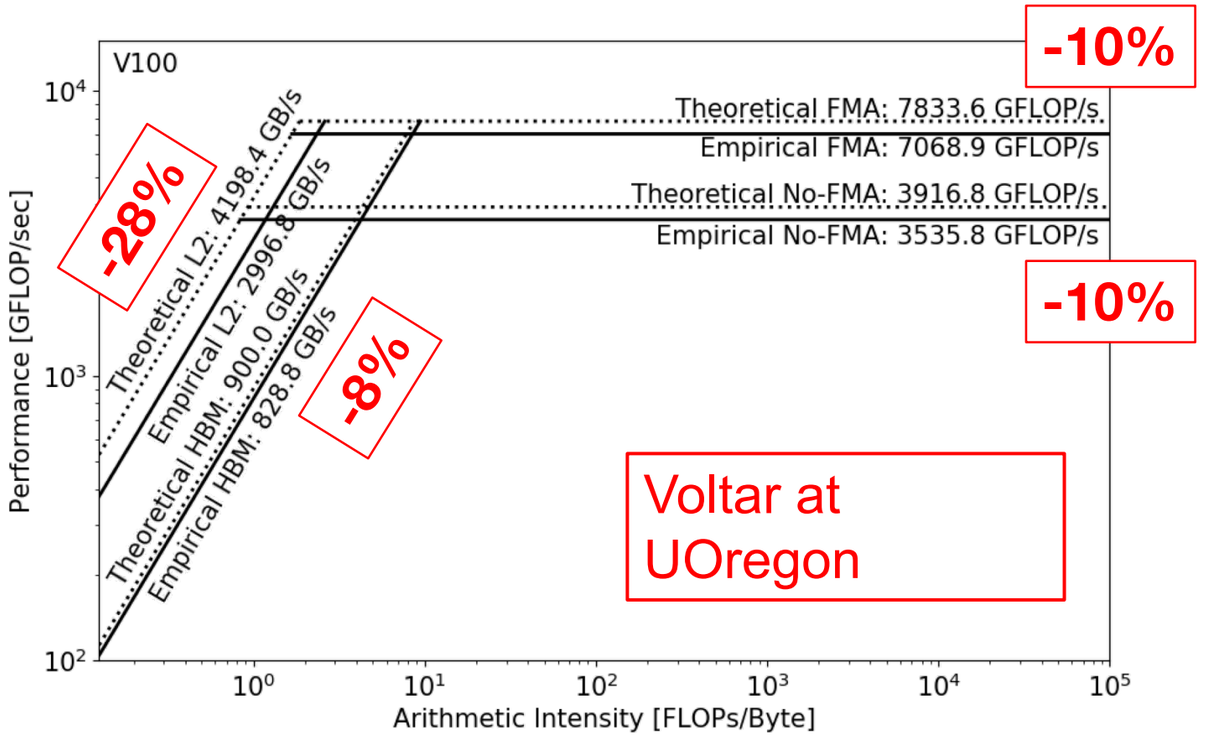
config script
- set up ranges of parameters

job script
- submit the job and run it

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.



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`

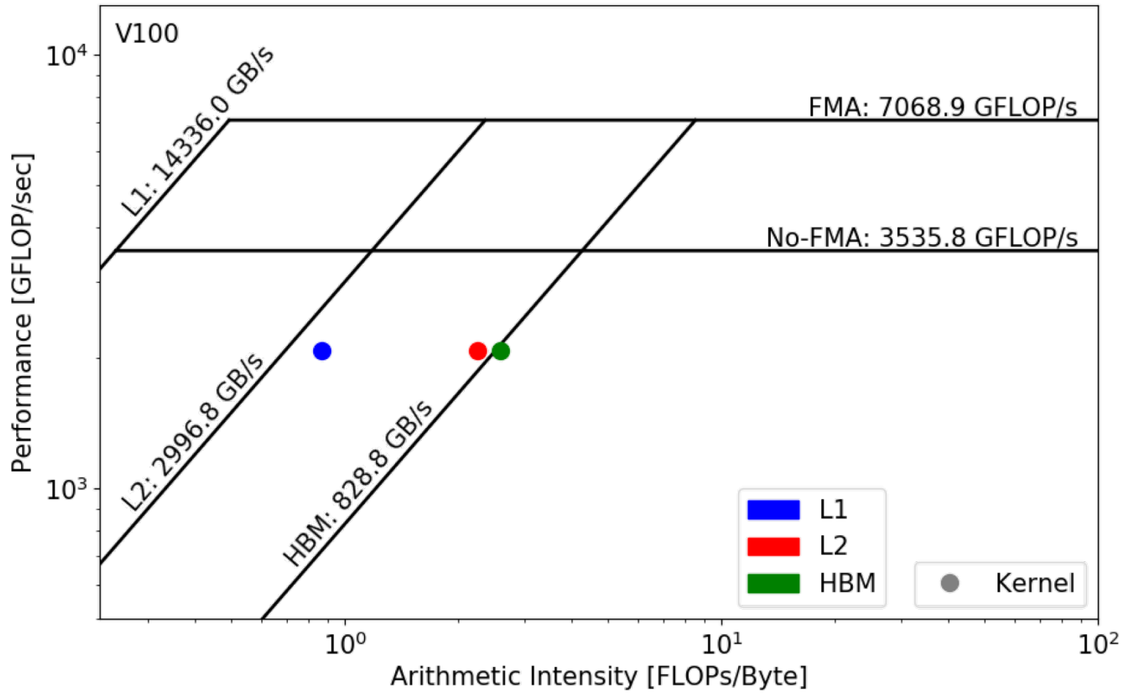
You must combine ERT and Nsight data

```
% 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'
```

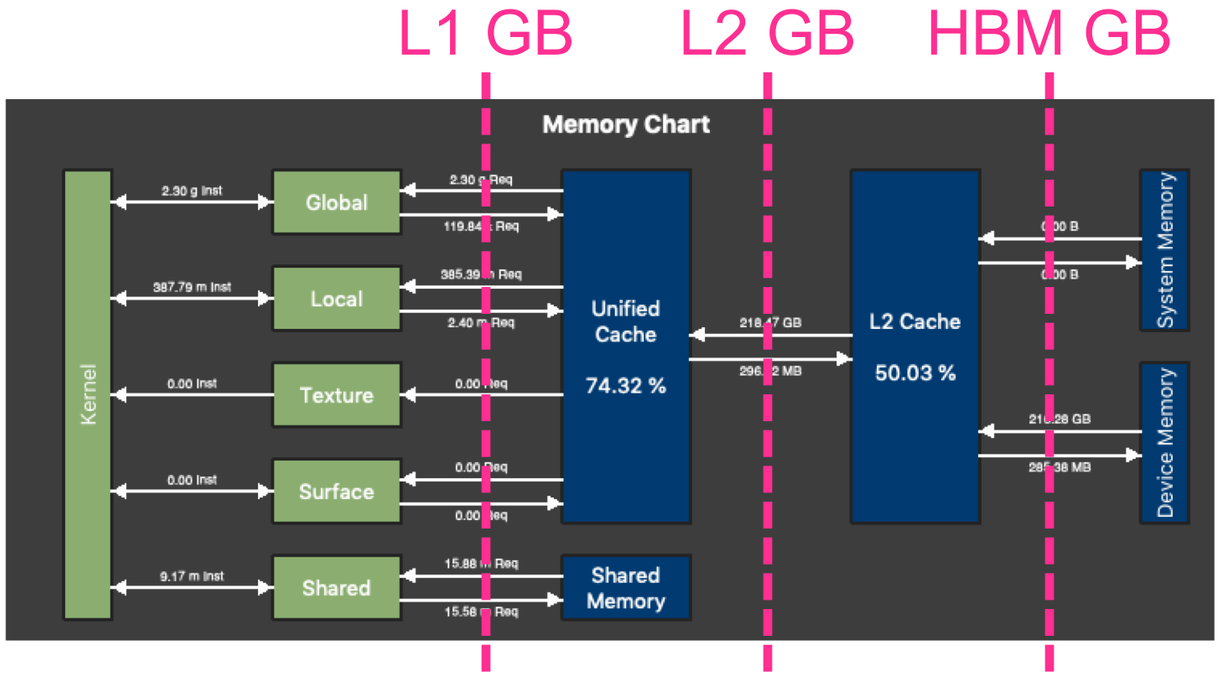
```
% plot_roofline.py data.txt
```




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

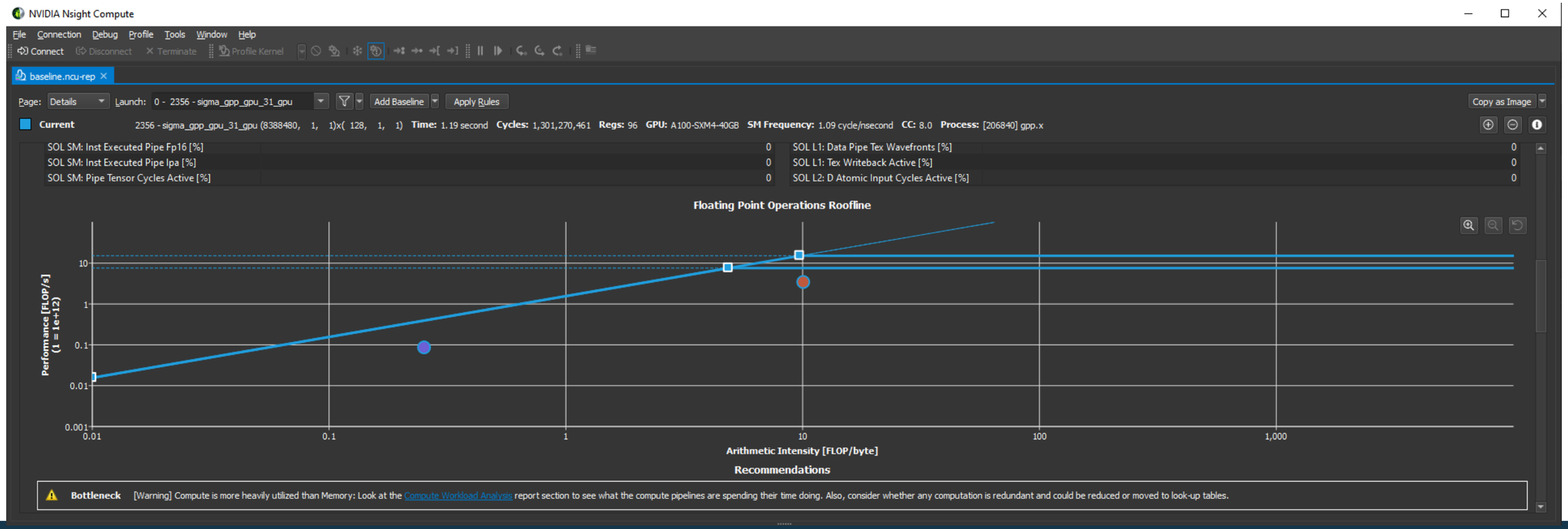


- Roofline in GUI:

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

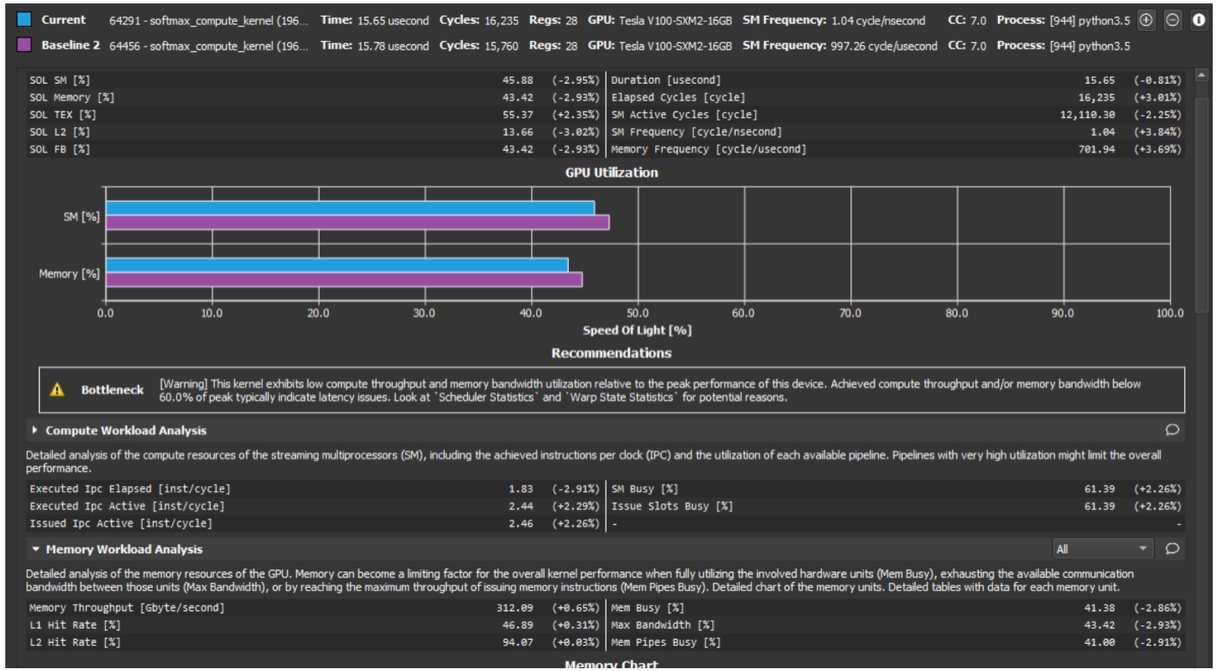
Nsight has integrated Roofline Analysis

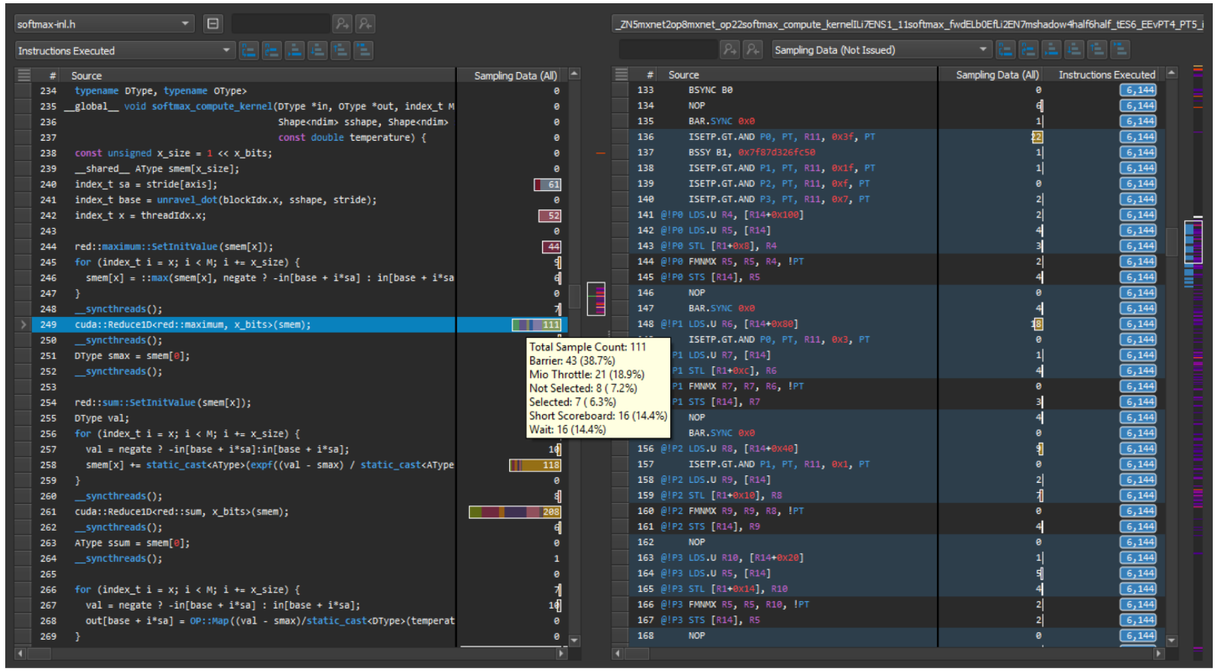
- Automatically plots Roofline (DRAM shown below)
- Allows you to compare multiple versions of a kernel on the same Roofline (tracks progress towards optimality)



Complements Existing SOL and PTX analysis

- speed-of-light analysis comparisons with baseline
- Source/PTX/SASS analysis and correlation





BERKELEY LAB

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?

Benchmarking, cross-architecture, custom plots, etc…

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

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`