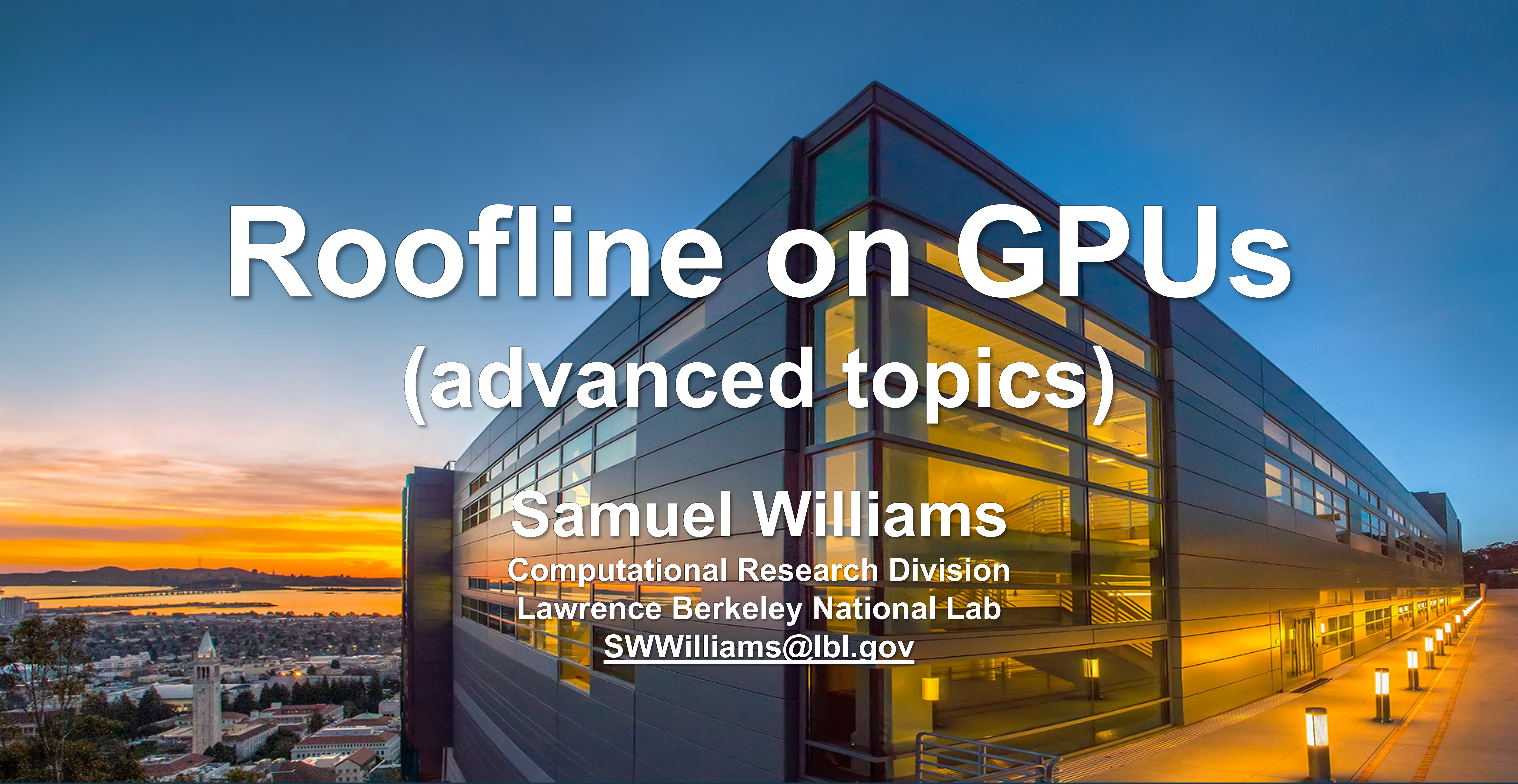
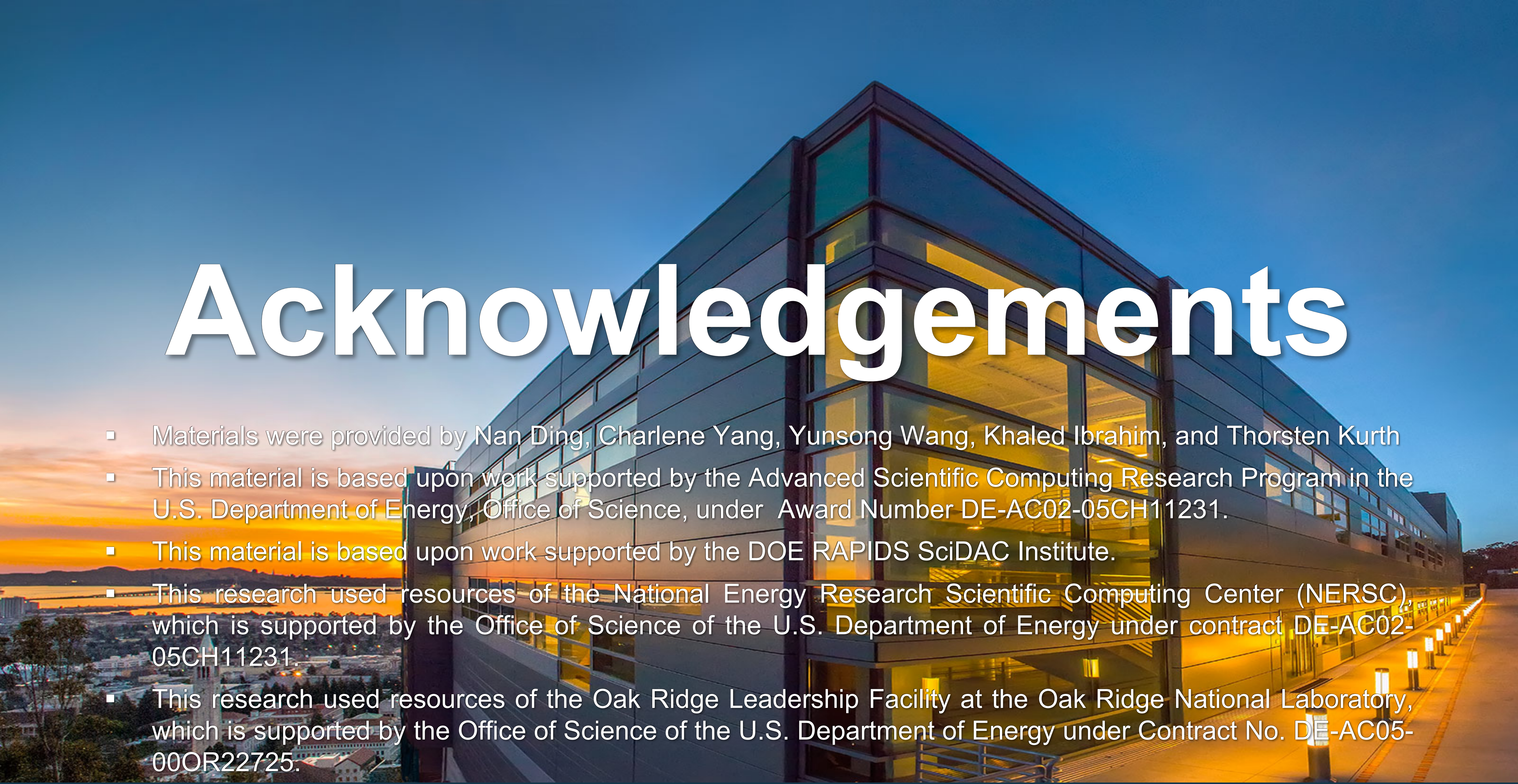


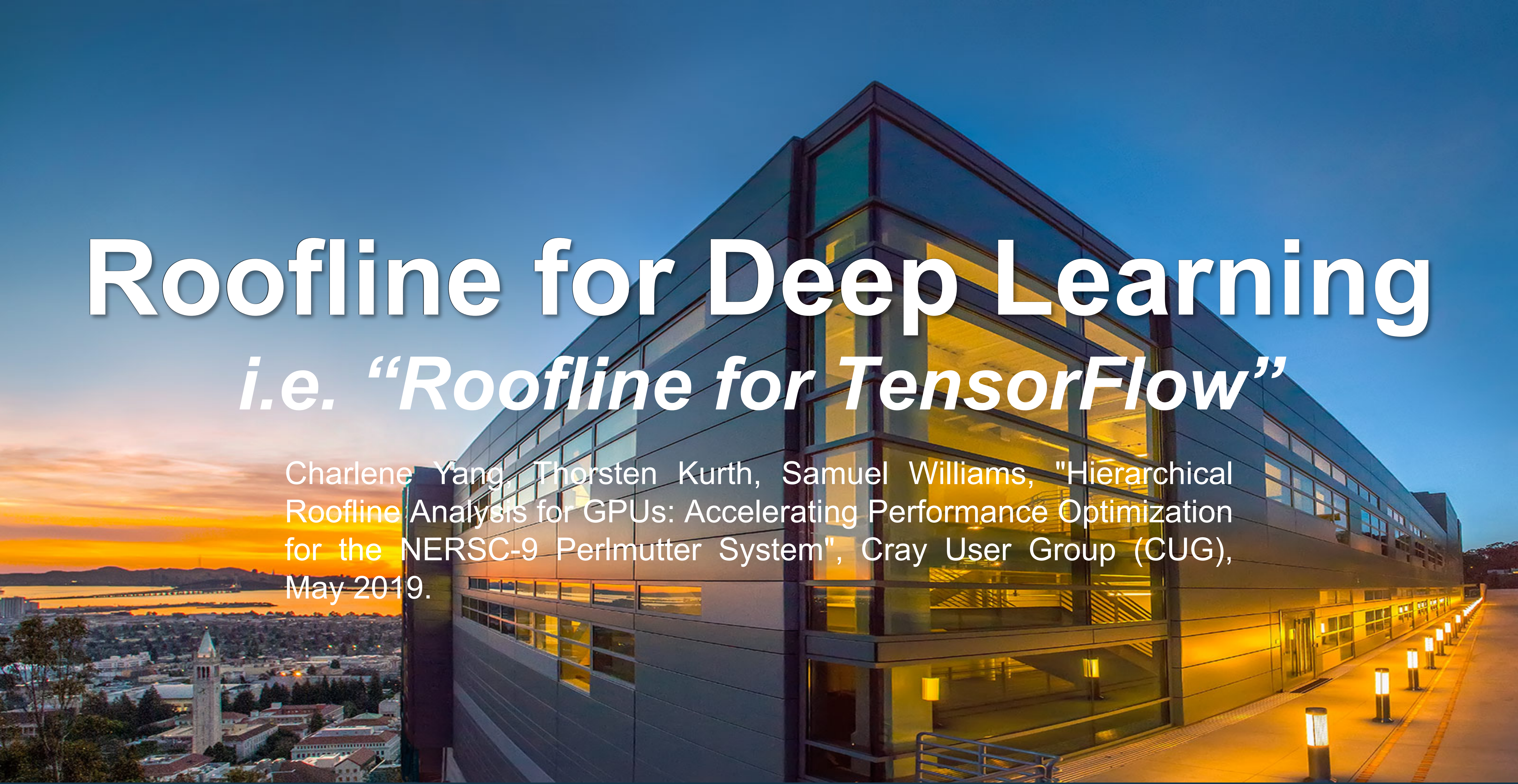
Roofline on GPUs
(advanced topics)

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



Acknowledgements

- Materials were provided by Nan Ding, Charlene Yang, Yunsong Wang, Khaled Ibrahim, and Thorsten Kurth
- This material is based upon work supported by the Advanced Scientific Computing Research Program in the U.S. Department of Energy, Office of Science, under Award Number DE-AC02-05CH11231.
- This material is based upon work supported by the DOE RAPIDS SciDAC Institute.
- This research used resources of the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-05CH11231.
- This research used resources of the Oak Ridge Leadership Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.



Roofline for Deep Learning

i.e. “Roofline for TensorFlow”

Charlene Yang, Thorsten Kurth, Samuel Williams, "Hierarchical Roofline Analysis for GPUs: Accelerating Performance Optimization for the NERSC-9 Perlmutter System", Cray User Group (CUG), May 2019.

Performance of DL is as important as simulation

- DOE is beginning to incorporate DL into simulation and analysis
- Training can be extremely computationally expensive
- Performance analysis/optimization of DL can be as important as performance analysis of simulation.

- Can Roofline be used to…
  - Quantify efficiency of Deep Learning frameworks?
  - Motivate optimizations to improve framework performance?
  - Identify facets of architectures should be emphasized to accelerate DL for science?

Charlene Yang, Thorsten Kurth, Samuel Williams, "Hierarchical Roofline Analysis for GPUs: Accelerating Performance Optimization for the NERSC-9 Perlmutter System", Cray User Group (CUG), May 2019.

conv2d from TensorFlow

- Demonstrate Roofline methodology using TensorFlow+cuDNN

- Setup…

```
input_image = tf.random_uniform(shape=input_size, minval=0., maxval=1., dtype=dtype)
output_result = conv2d(input_image, ’NHWC’, kernel_size, stride_size, dtype)
```

- Forward Pass (2D conv)

```
exec_op = output_result
```

- Backward Pass (2D conv + derivative)

```
    opt = tf.train.GradientDescentOptimizer(0.5)
exec_op = opt.compute_gradients(output_result)
```

Charlene Yang, Thorsten Kurth, Samuel Williams, "Hierarchical Roofline Analysis for GPUs: Accelerating Performance Optimization for the NERSC-9 Perlmutter System", Cray User Group (CUG), May 2019.

conv2d from TensorFlow

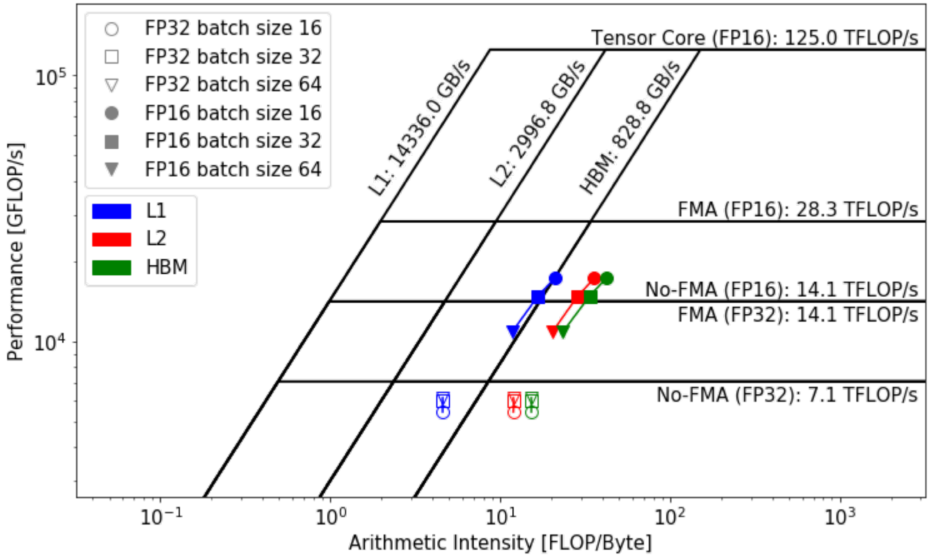
- Each TensorFlow kernel includes multiple sub-kernels
  - Padding, permutations, conversions, compute, etc…
  - Should include all of them when analyzing performance

- TensorFlow also includes an autotuning step
- Must ignore autotuning when profiling/modeling…
  - nvprof --profile-from-start off
  - run 5 warmup iterations (autotuning / not profiled)
  - start profiler (pyc.driver.start_profiler), run 20 iterations, stop profiler
  - Sum up DP, SP, HP metrics. Scale NVProf TC utilization metric

- Vary parameters to understand effects on performance

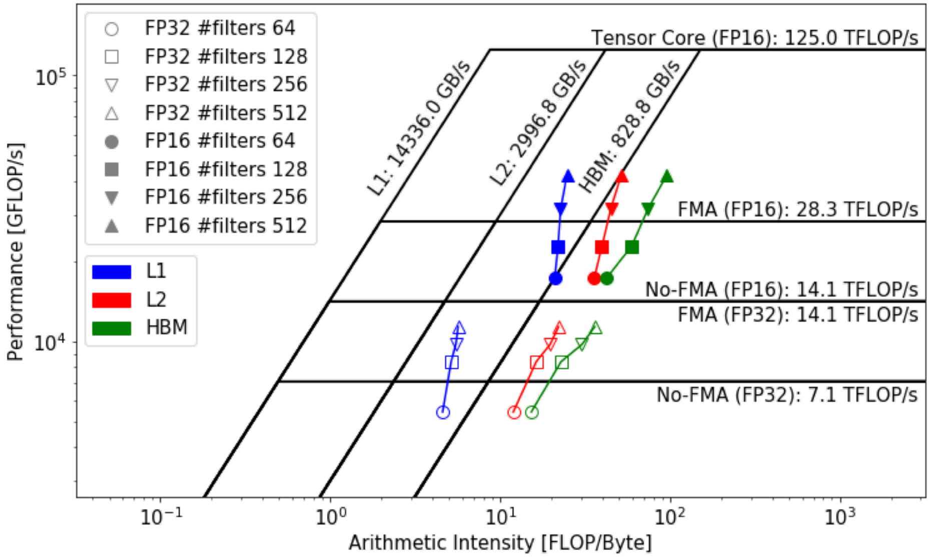
Charlene Yang, Thorsten Kurth, Samuel Williams, "Hierarchical Roofline Analysis for GPUs: Accelerating Performance Optimization for the NERSC-9 Perlmutter System", Cray User Group (CUG), May 2019.

TensorFlow / Forward Pass



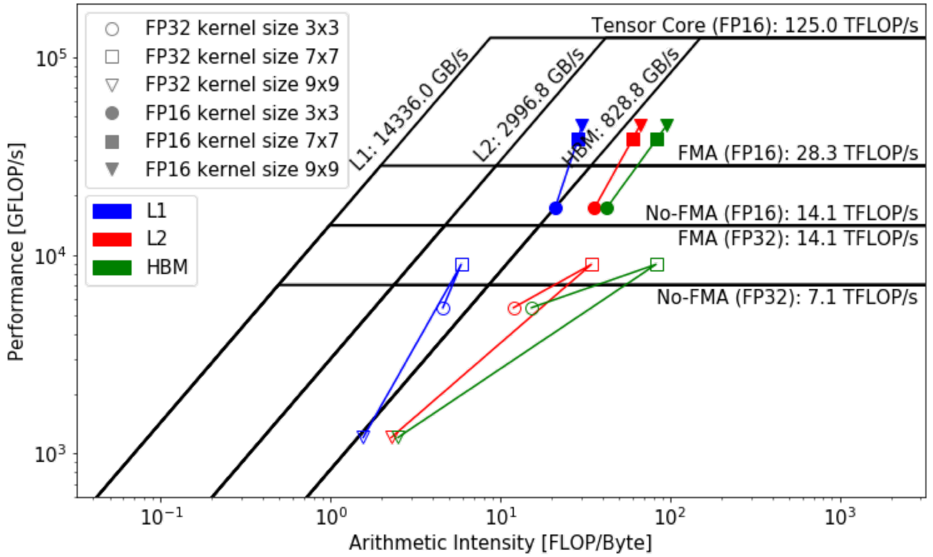
Batch Size

- Constant performance(?)
- **FP16 performance anti-correlated with batch size**
- Performance << TC peak
- Transformation kernels
- Low L2 locality



#Filters

- Intensity ∝ #Filters
- Low L2 data locality
- Some use of TC's (>FP16 FMA)… ***partial TC ceiling***

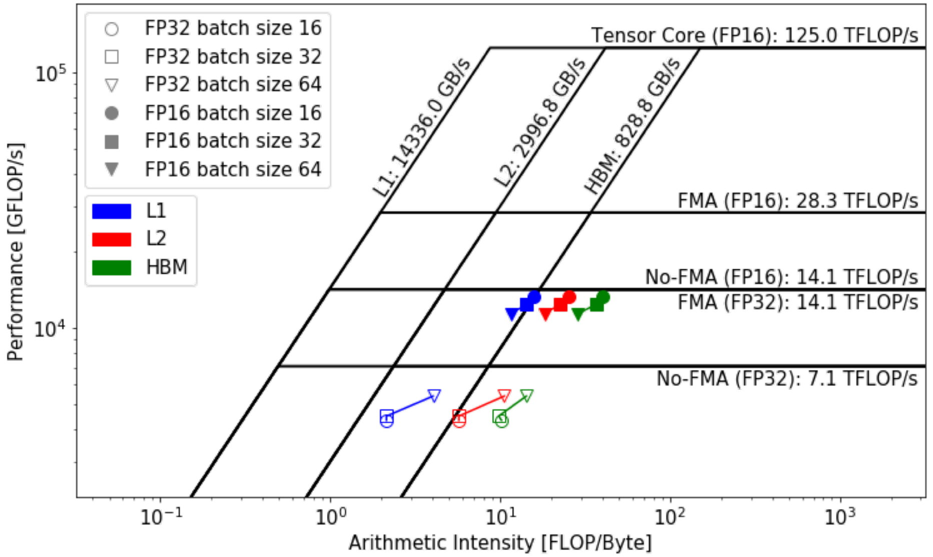


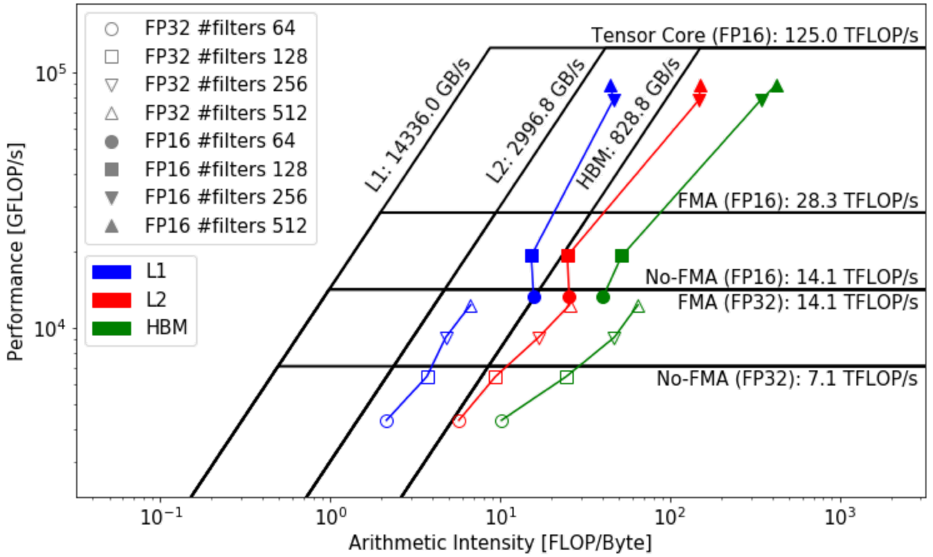
Kernel Size

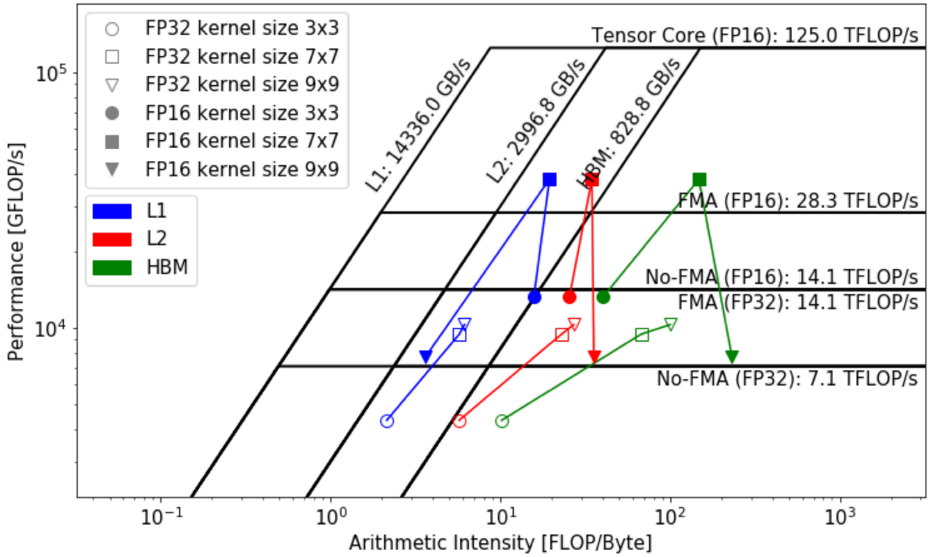
- Intensity ∝ kernel size
- Low L2 data locality
- **Autotuner switched FP32 algorithm to FFT at 9x9**
- Some use of TC's (>FP16 FMA)… ***partial TC ceiling***

Charlene Yang, Thorsten Kurth, Samuel Williams, "Hierarchical Roofline Analysis for GPUs: Accelerating Performance Optimization for the NERSC-9 Perlmutter System", Cray User Group (CUG), May 2019.

TensorFlow / Backward Pass







Batch Size

- Autotuner chose different (**better**) algorithm for FP32 with batch size = 64 (boost)

#Filters

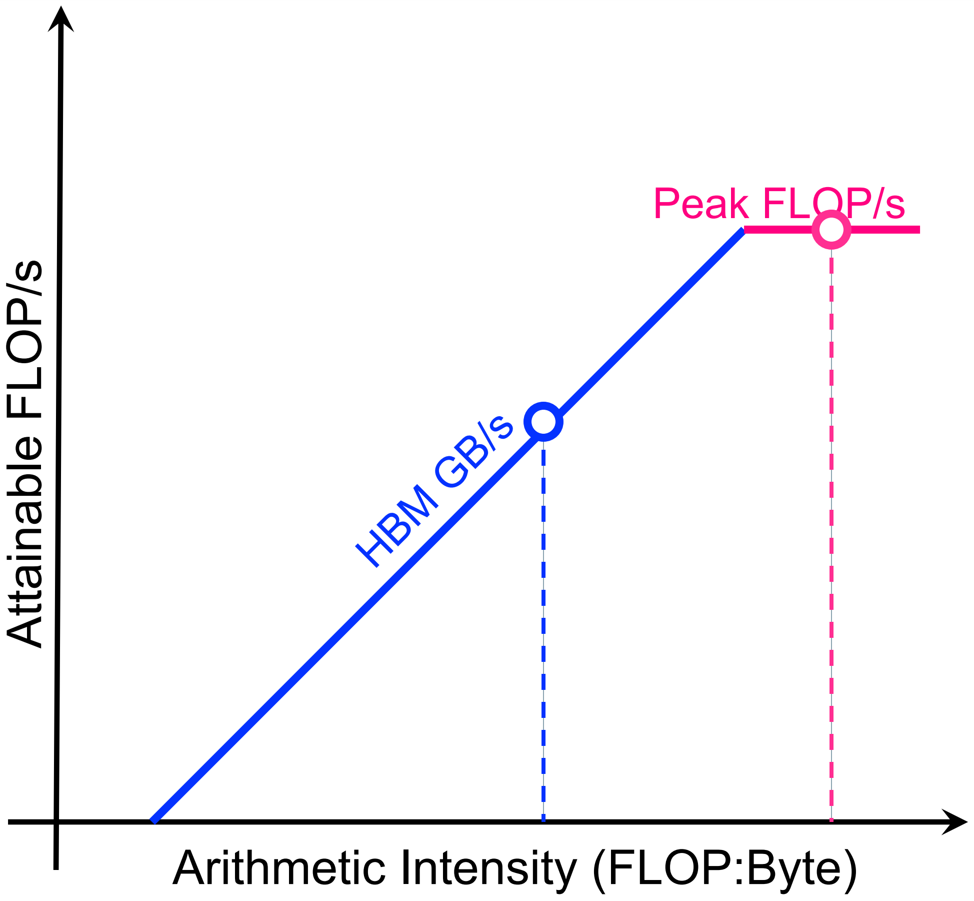
- Close to FP16 TC peak
- Close to FP32 FMA peak

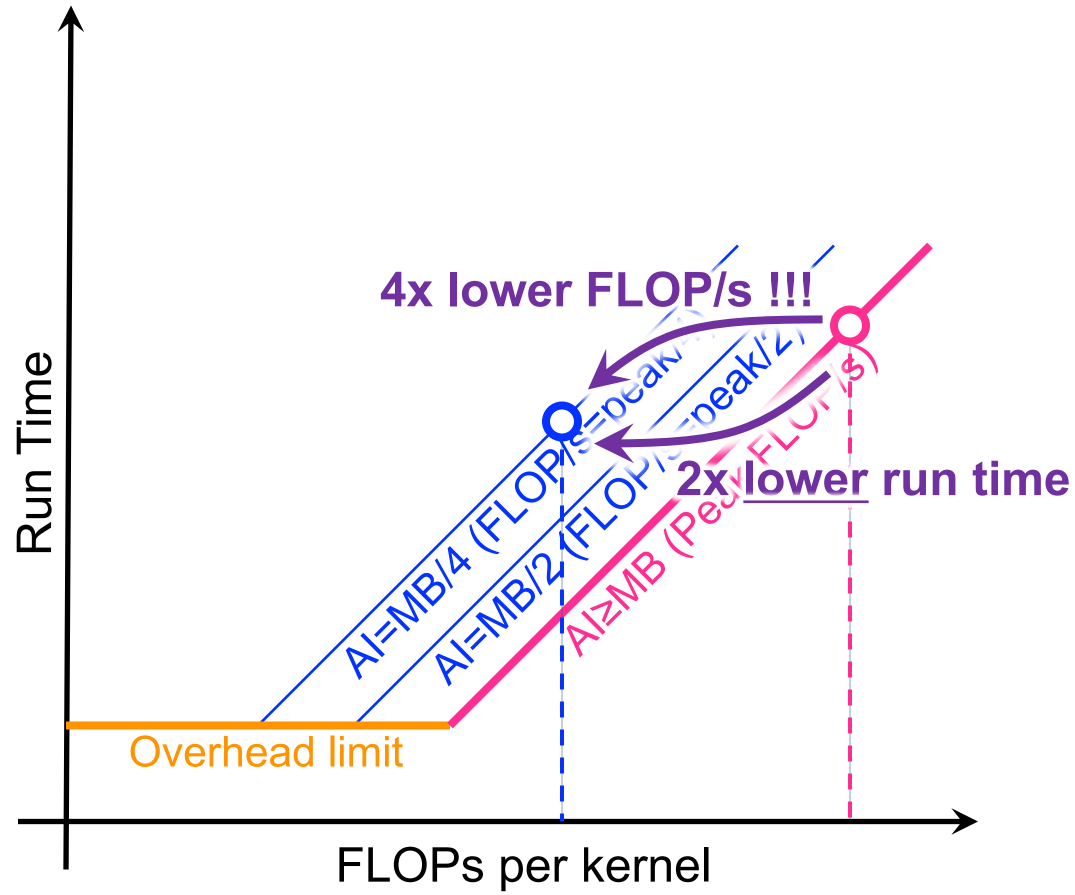
Kernel Size

- Good FP32 performance trend (almost peak)
- **Autotuner chose to run 9x9 FP16 in FP32 !!**

Charlene Yang, Thorsten Kurth, Samuel Williams, "Hierarchical Roofline Analysis for GPUs: Accelerating Performance Optimization for the NERSC-9 Perlmutter System", Cray User Group (CUG), May 2019.

How do you compare different algorithms?





- Imagine 2 kernels solve the same problem using different algorithms
- We're used to thinking one with higher FLOP/s is better

- Original Roofline is about FLOP/s
- Need alternate time-based version to compare optimizations that change the number of FLOPs

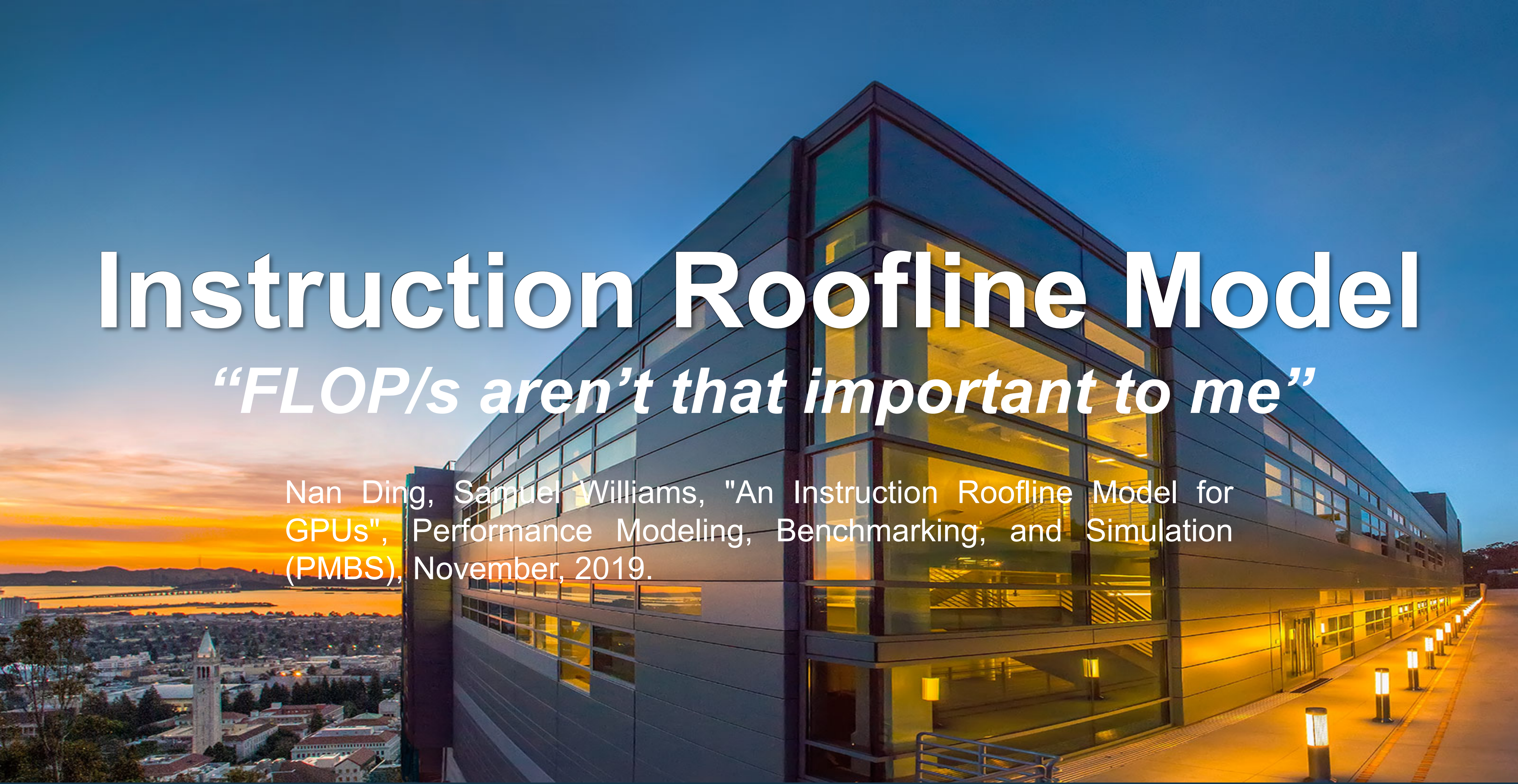
Tensor Flow Takeaway

- Performance rarely anywhere close to FP16 peak
  - Serializing compute and data permutation kernels drives down AI
  - Changing algorithms or precisions can confuse Roofline Analysis

  ➢ **Need alternate time formulation of Roofline**
  i.e. differentiating architectural efficiency from algorithmic efficiency

  ➢ **Need a better way of analyzing mixed precision codes**
  i.e. understanding bottlenecks when mixing FP32, FP16, and tensor core instructions

- Migrating from NVProf to Nsight Compute
  - More accurate WMMA counts
  - >10x faster



Instruction Roofline Model

“FLOP/s aren’t that important to me”

Nan Ding, Samuel Williams, "An Instruction Roofline Model for GPUs", Performance Modeling, Benchmarking, and Simulation (PMBS), November, 2019.

How do we go beyond the FLOP Roofline?

- Think about classifying applications by instruction mix…
  - Heavy floating-point (rare in DOE)
  - Mix of integer and floating-point
  - Integer-only (e.g. bioinformatics, graphs, etc…)
  - Mixed precision

- FLOP/s → IntOP/s → FLOP/s+IntOP/s
  - Adopted by Intel Advisor
  - Useful when wanting to understand ‘performance’ rather than bottlenecks
  - **What is an “Integer Op”?** ***LEA but not an SIB byte?***
  - **Instruction Fetch/Decode/Issue bottlenecks?**
  - **Functional Unit Bottlenecks?**

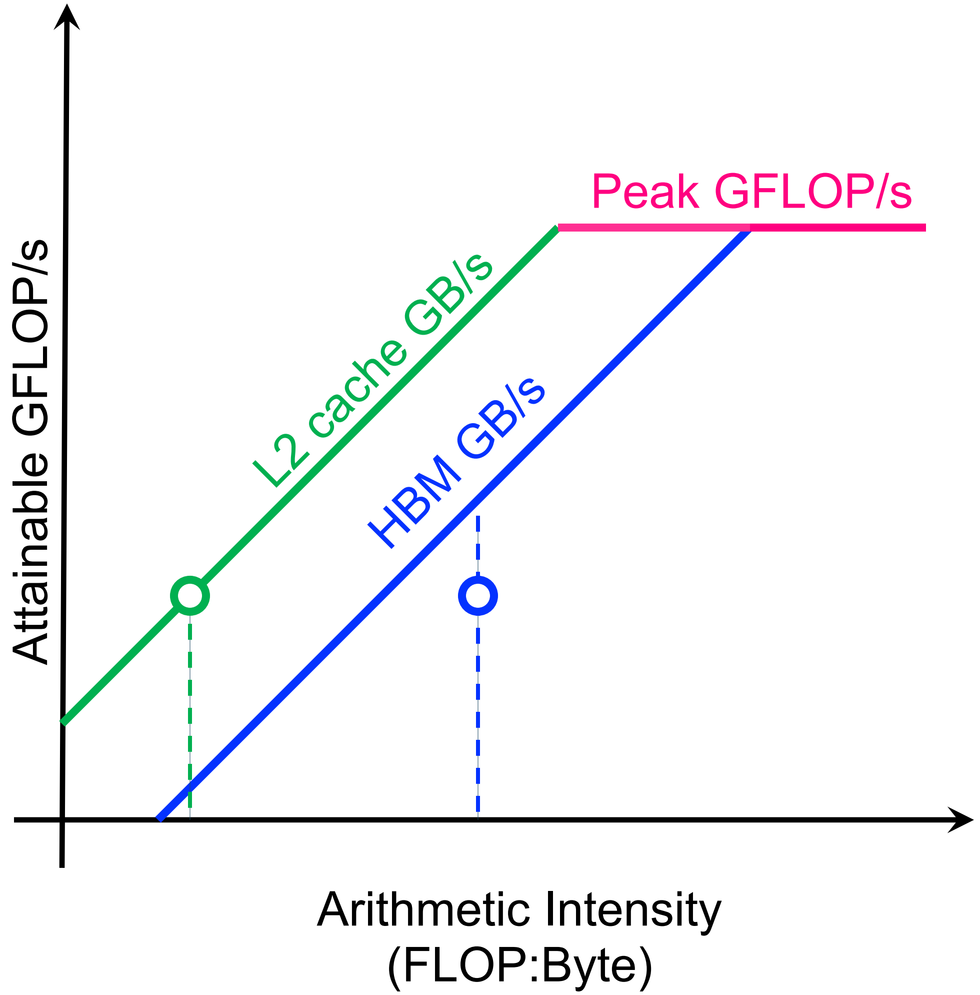
  ➢ **Need to create a true instruction Roofline**

NVIDIA GPU Instruction Roofline

- Instructions/second? Instructions per Byte?
- What is an 'Instruction' on a GPU?
  - Thread-level hides issue limits?
  - Warp-level hides predication effects?
  - **Scale non-predicated threads down by the warp size (divide by 32)**
  - **Show warp instructions per second**
  - Break instructions into subclasses (integer, FP32, FP64, LDST, WMMA)
- Naively, one would think instruction intensity should use 'bytes'
  - Matches well to existing Roofline; works with well-known bandwidths
- GPUs access memory using 'transactions'
  - 32B for global/local/L2/HBM
  - 128B for shared memory
  - ➢ **"Instructions/Transaction" preserves traditional Roofline, but enables a new way of understanding memory access**

Instruction Roofline

$$\text{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ AI_{DRAM} * \text{DRAM GB/s} \end{cases}$$



Instruction Roofline

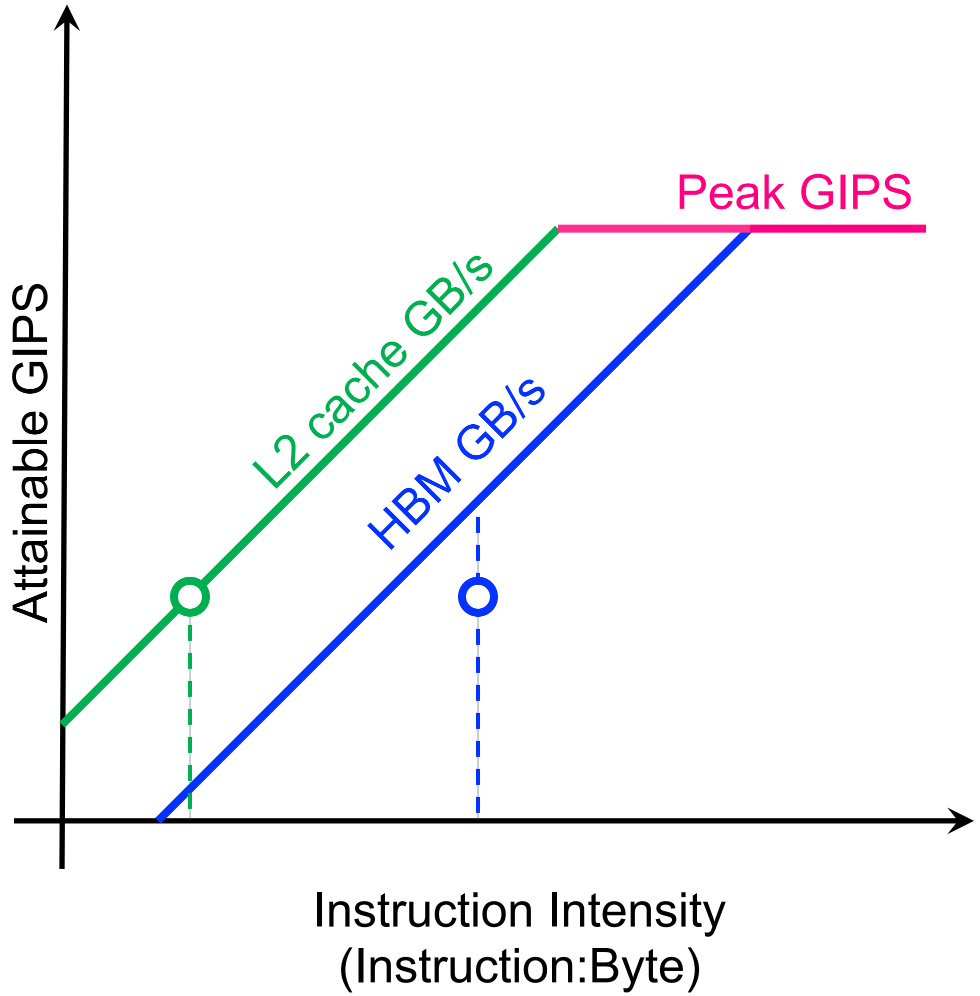
$$\textbf{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ AI_{DRAM} * \text{DRAM GB/s} \end{cases}$$

⬇

$$\textbf{GIPS} = \min \begin{cases} \text{Peak GIPS} \\ II_{DRAM} * \text{DRAM GB/s} \end{cases}$$

↑

Instructions per Byte



Nan Ding, Samuel Williams, "An Instruction Roofline Model for GPUs", PMBS, November, 2019.

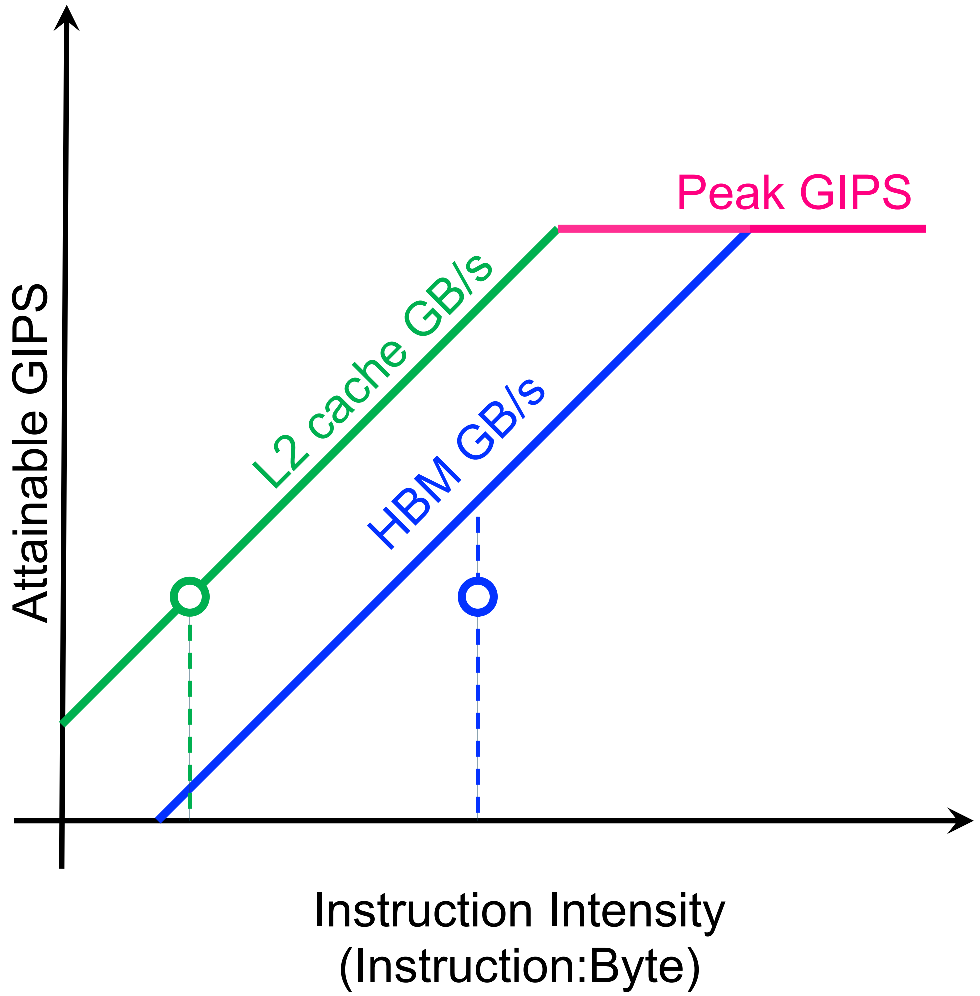
Instruction Roofline on GPUs

$$\text{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ AI_{DRAM} * \text{DRAM GB/s} \end{cases}$$

Warp Instructions

$$\text{GIPS} = \min \begin{cases} \text{Peak GIPS} \\ II_{DRAM} * \text{DRAM GB/s} \end{cases}$$

As the natural quanta for GPU memory access is a "transaction"...



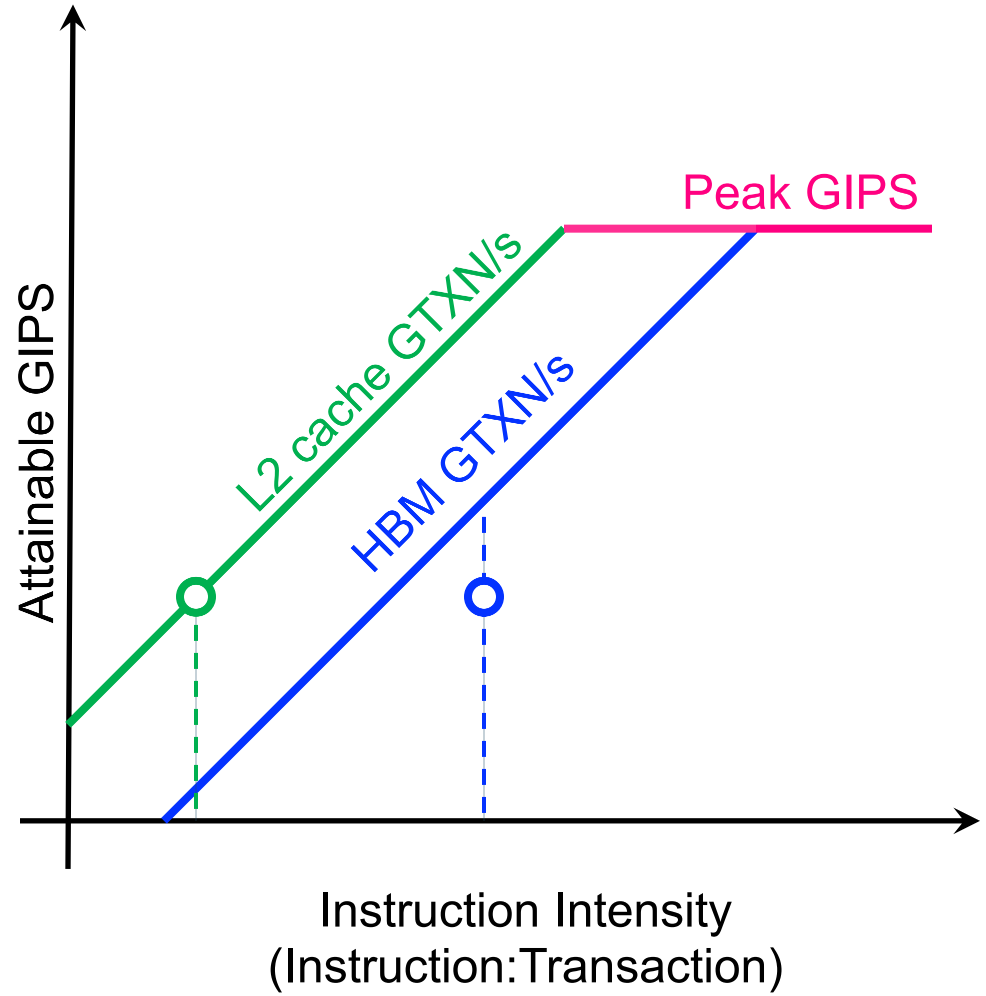
Nan Ding, Samuel Williams, "An Instruction Roofline Model for GPUs", PMBS, November, 2019.

Instruction Roofline on GPUs

$$\text{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ \text{AI}_{\text{DRAM}} * \text{DRAM GB/s} \end{cases}$$

$$\text{GIPS} = \min \begin{cases} \text{Peak GIPS} \\ \text{II}_{\text{DRAM}} * \text{DRAM GB/s} \end{cases}$$

$$\text{GIPS} = \min \begin{cases} \text{Peak GIPS} \\ \text{II}_{\text{DRAM}} * \text{DRAM GTXN/s} \end{cases}$$



II_x (Instruction Intensity at level "x") =

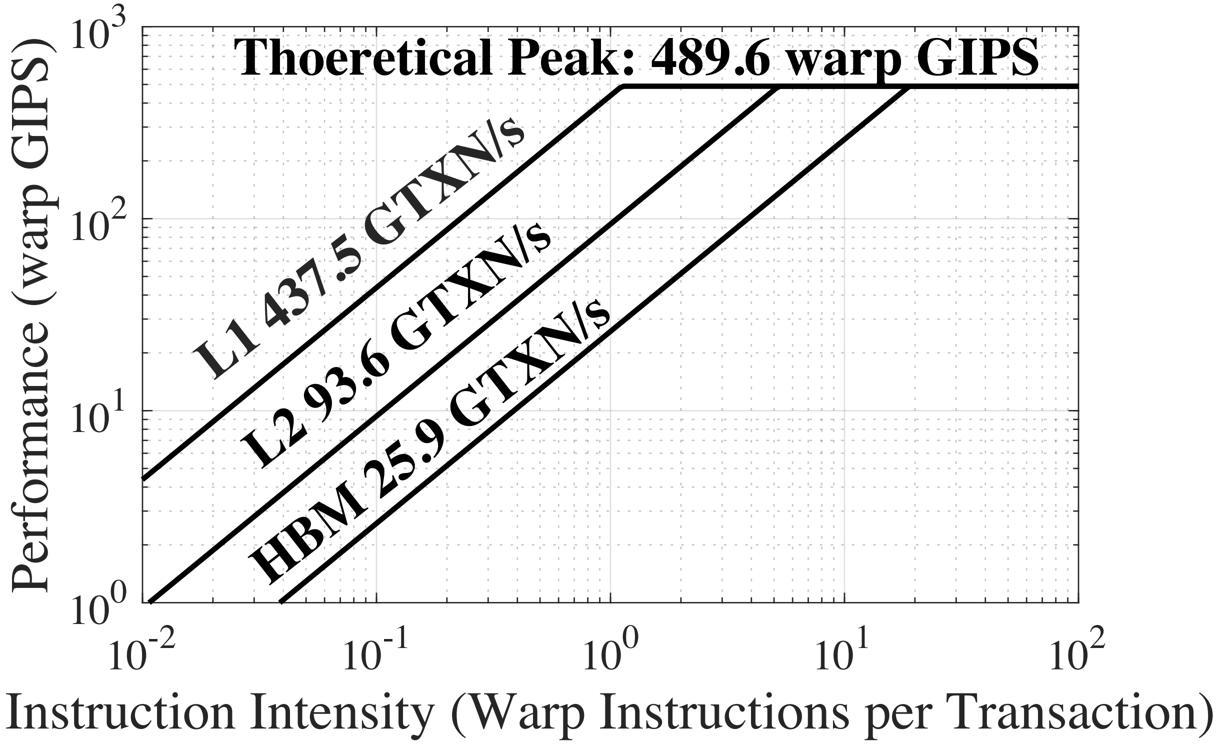
Instructions / Transactions (to/from level "x")

Nan Ding, Samuel Williams, "An Instruction Roofline Model for GPUs", PMBS, November, 2019.

Instruction Roofline on NVIDIA GPUs

- Instruction Intensity (II)
  - (Warp or equivalent) Instructions / Transaction
  - Refine into L1 (global+local+shared), L2, HBM Instruction Intensities
  - Further refine based on instruction type (**LDST instructions / global transaction**)

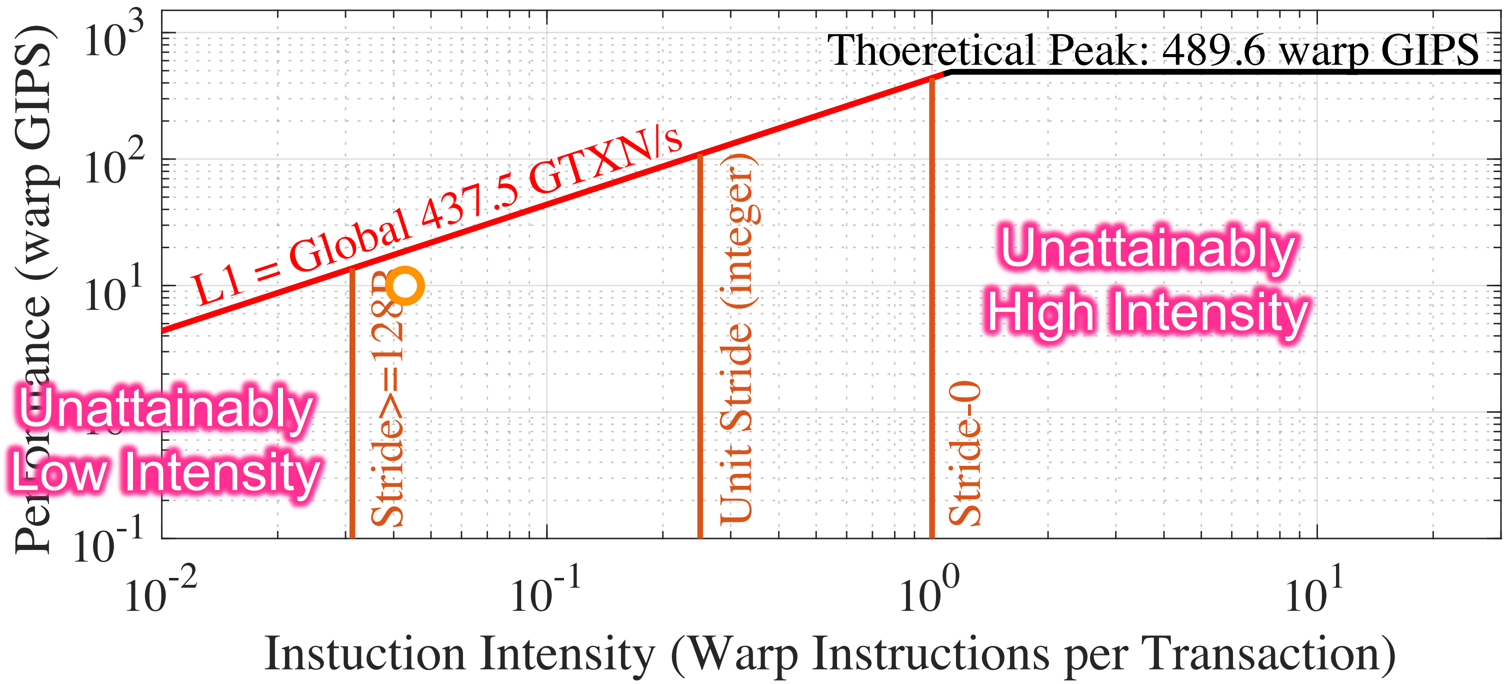
- Peak Performance and Peak Bandwidths
  - Instruction:
    - 80 SMs * 4 warps * 1.53GHz ~ 490 GIPS (warp-level)
  - Use ERT for memory (convert from GB/s)
    - L1: 80 SMs * 4 transactions/cycle * 1.53 GHz ~ 490 GTXN/s
    - L2: 94 GTXN/s (empirical)
    - HBM: 26 GTXN/s (empirical)



Nan Ding, Samuel Williams, "An Instruction Roofline Model for GPUs", PMBS, November, 2019.

Efficiency of Global Memory Access

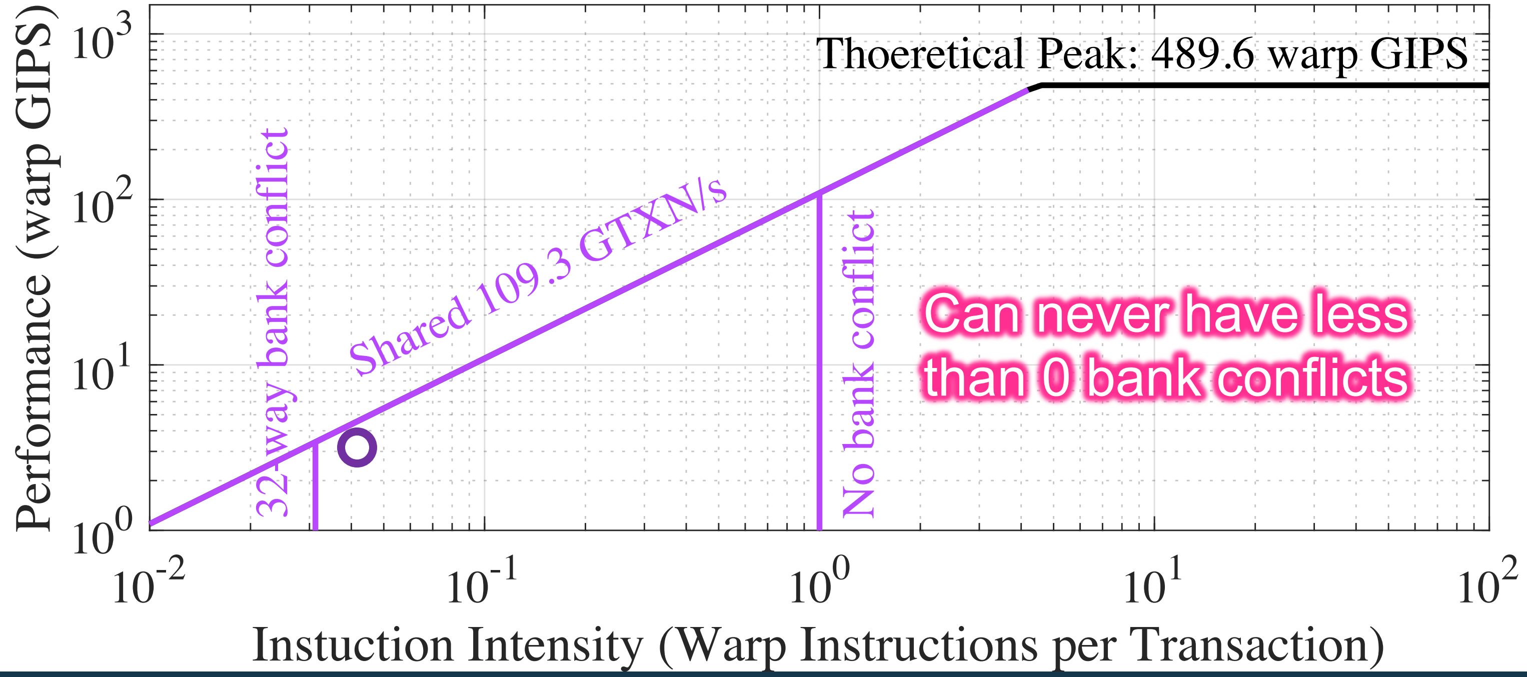
- (Global)LDST Instruction Intensity has a special meaning / use…
  - Global LDST instructions / Global transactions
  - Numerator lower than nominal II
  - Denominator can be lower than nominal L1 II (no local or shared transactions)
- Denotes efficiency of memory access
- 3 “**Walls**” of interest:
  - ≥1 transaction per LDST instruction (all threads access same location)
  - ≤32 transactions per LDST instruction (gather/scatter or stride>=128B)
  - Unit Stride: 1 LDST per 8 transactions (double precision)



Nan Ding, Samuel Williams, "An Instruction Roofline Model for GPUs", PMBS, November, 2019.

Efficiency of Shared Memory Access

- (Shared)LDST Instruction Intensity also has a special meaning / use
  - Shared LDST instructions / Shared transactions
  - II is similarly loosely related to nominal II
- Can be used to infer the number of bank conflicts
- 2 “**Walls**” of interest:
  - Minimum of 1 transaction per shared LDST instruction (***no bank conflicts***)
  - Maximum of 32 transactions per shared LDST instruction (***all threads access different lines in the same bank***)



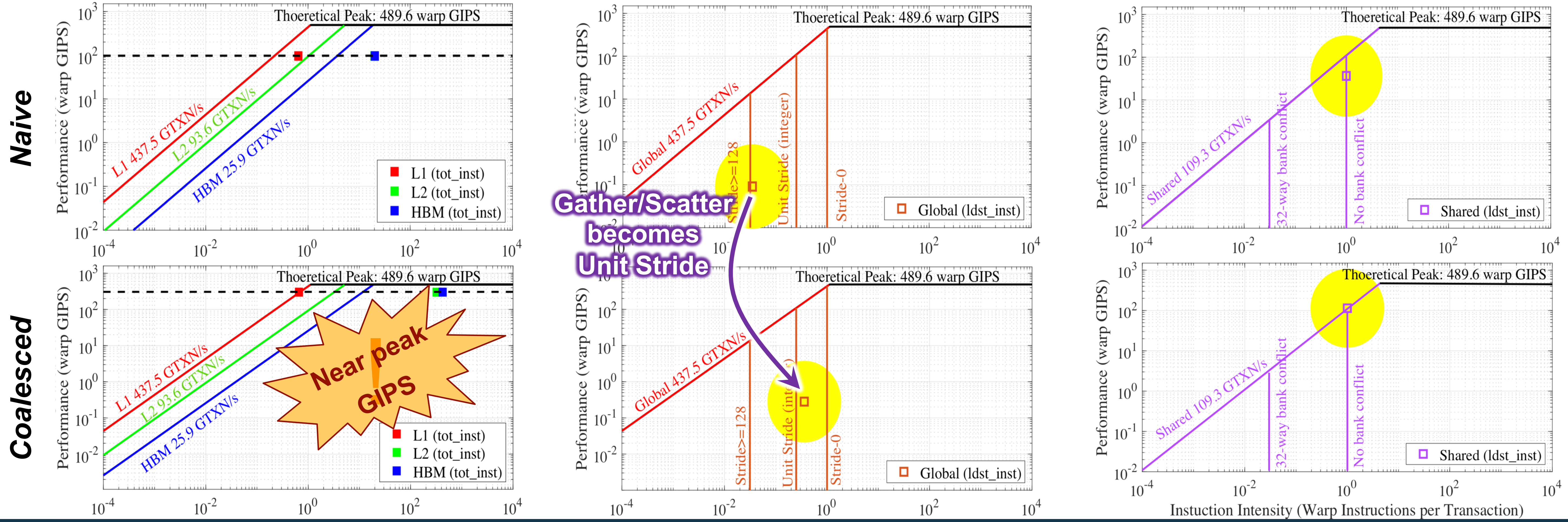
Can never have less than 0 bank conflicts

Nan Ding, Samuel Williams, "An Instruction Roofline Model for GPUs", PMBS, November, 2019.

Instruction Roofline for Smith-Waterman

- Integer-only Alignment code on NVIDIA GPU
- No predication effects, but inefficient global memory access

Instruction Hierarchy & Thread Predication ***Global Memory Efficiency*** ***Shared Memory Efficiency***



Gather/Scatter becomes Unit Stride

Near peak GIPS

Nan Ding, Samuel Williams, "An Instruction Roofline Model for GPUs", PMBS, November, 2019.

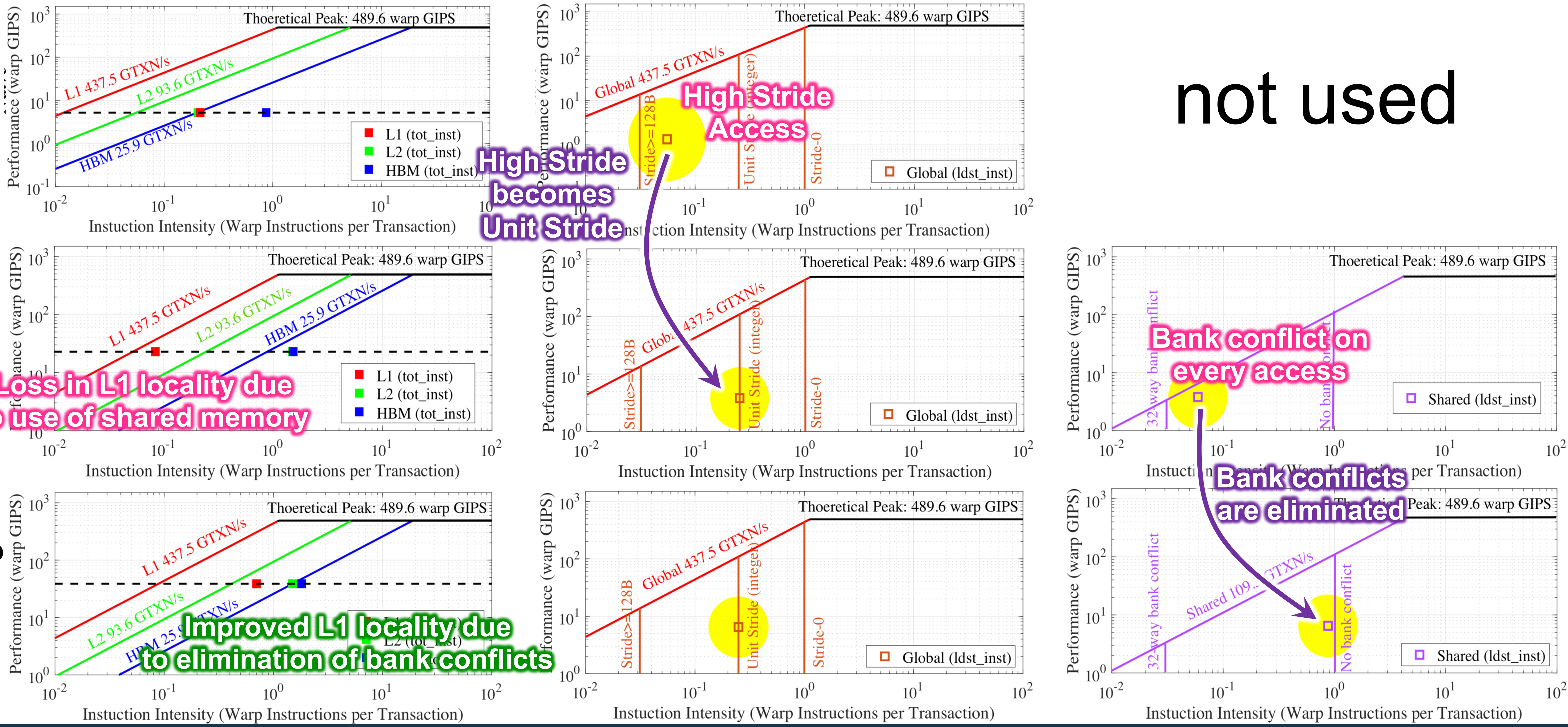
Instruction Roofline for Matrix Transpose

Instruction Hierarchy & Thread Predication ***Global Memory Efficiency*** ***Shared Memory Efficiency***

Naive

Transpose in Shared

Array Padding



not used

High Stride Access

High Stride becomes Unit Stride

Loss in L1 locality due to use of shared memory

Bank conflict on every access

Bank conflicts are eliminated

Improved L1 locality due to elimination of bank conflicts

Nan Ding, Samuel Williams, "An Instruction Roofline Model for GPUs", PMBS, November, 2019.

Other Uses of Instruction Roofline

- Predication
  - Individual threads can mask out execution when in branch-not-taken
  - 16 FLOPs/SM/cycle… 1 FP warp every 2 cycles
    –or –
    1 FP warp every cycle with half threads predicated
  - Use performance metrics to plot both **warp GIPS** and **non-predicated threads** (scaled by 32)

- FMA, Tensor Cores, Mixed Precision, …
  - Rather than counting FLOPs, count GIPS
  - Can differentiate total instruction issue bandwidth from functional unit utilization (FP32, FP64)
  - n.b., some GIPS should be summed (FP16+FP32) while others are have dedicated pipelines (FP64, TC)

Nan Ding, Samuel Williams, "An Instruction Roofline Model for GPUs", PMBS, November, 2019.

Instruction Roofline Takeaway

Traditional Roofline

- **Tells us about performance** *(floating-point)*
- Use of FMA, SIMD, vectors, tensors has no affect on intensity, but may increase performance…
- Presence of integer instructions has no affect on intensity, but may decrease performance
- Reducing precision (64b, 32b, 16b) increases arithmetic intensity

Instruction Roofline

- **Tells us about bottlenecks** *(issue and memory)*
- Use of FMA, SIMD, vectors, tensors decreases intensity and may decrease "performance"
- Presence of integer instructions increases intensity and might increase performance.
- Reducing precision has no affect on intensity

Memory Walls

- **Tells us about efficiency** *(memory access)*
- Intensity based on LDST instructions and transactions
- Predication could affect intensity (could have zero transactions for a LDST instruction, but not all LDST instructions)
- Reducing precision shifts intensity, and the unit-stride wall

Nan Ding, Samuel Williams, "An Instruction Roofline Model for GPUs", PMBS, November, 2019.

Instruction Roofline on Vector CPUs?

Instruction Roofline on GPUs

- Warp-based → easy to see functional unit contention
- LDST instructions generate multiple transactions → memory walls
- Predication effects inferred through differences in (scaled) thread GIPS and warp GIPS

Instruction Roofline on CPUs

- micro-op (uOP) based → easy to see functional unit contention
- ***Memory walls can only be constructed if VGATHER/VSCATTER generate multiple cache performance counter events***
- ***Masking/Predication effects can only be inferred if there are counts for individual vector lanes***

Roofline Scaling Trajectories

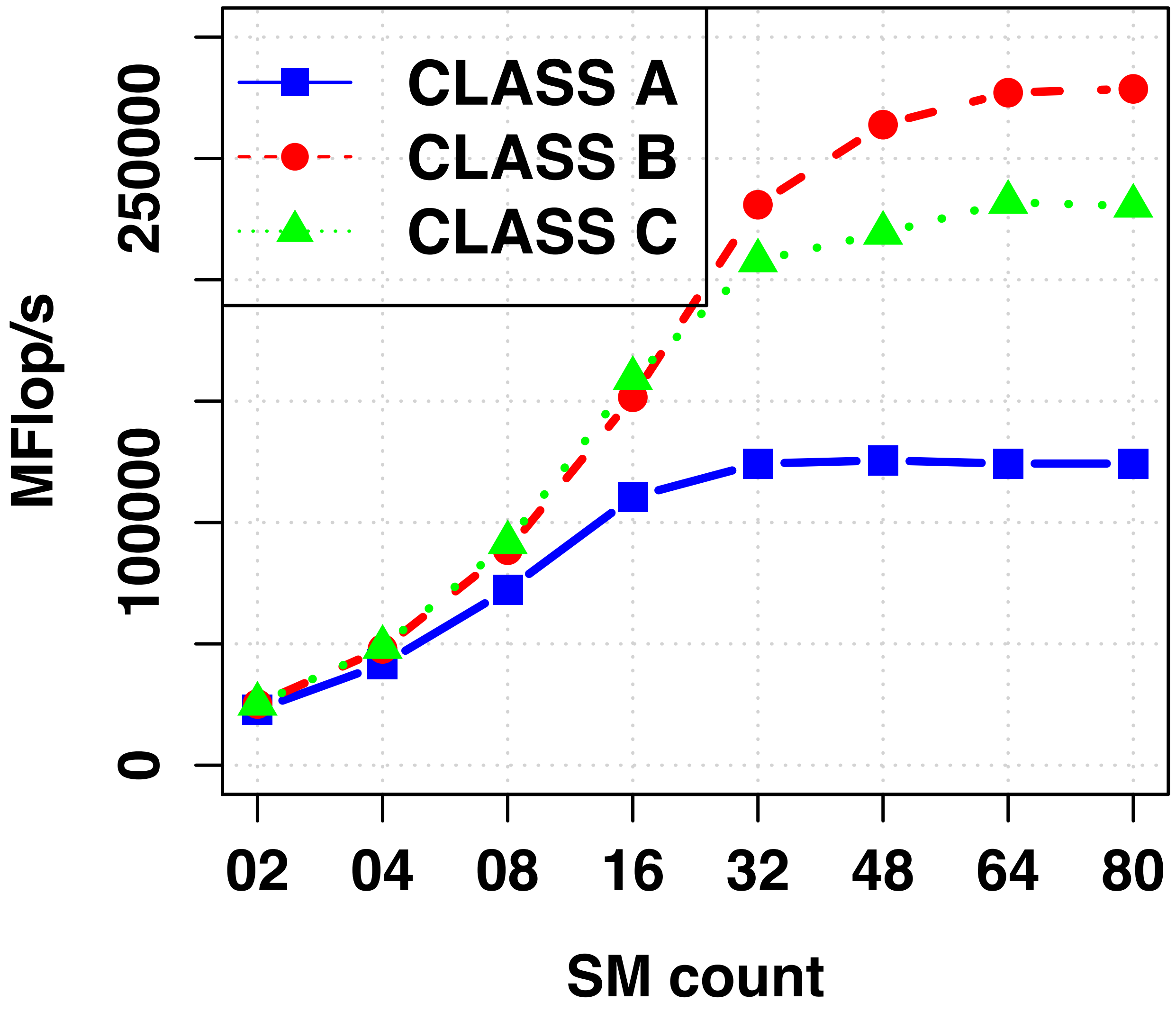
“Is my code ready for Perlmutter, Frontier, …"

Khaled Ibrahim, Samuel Williams, Leonid Oliker, “Performance Analysis of GPU Programming Models using the Roofline Scaling Trajectories”, Bench, November, 2019.

Understanding SM Scalability is hard

- Control SM count on Volta GPUs
- LU NAS Parallel benchmark (in CUDA)

- **Typical Scaling Plot**
  - Provide performance with SM change,
  - No insights into root causes.
  - Why Class B scale better than A,
  - but Class C is not better than B?



Khaled Ibrahim, Samuel Williams, Leonid Oliker, “Performance Analysis of GPU Programming Models using the Roofline Scaling Trajectories”, Bench, November, 2019.

Roofline Scaling Trajectory

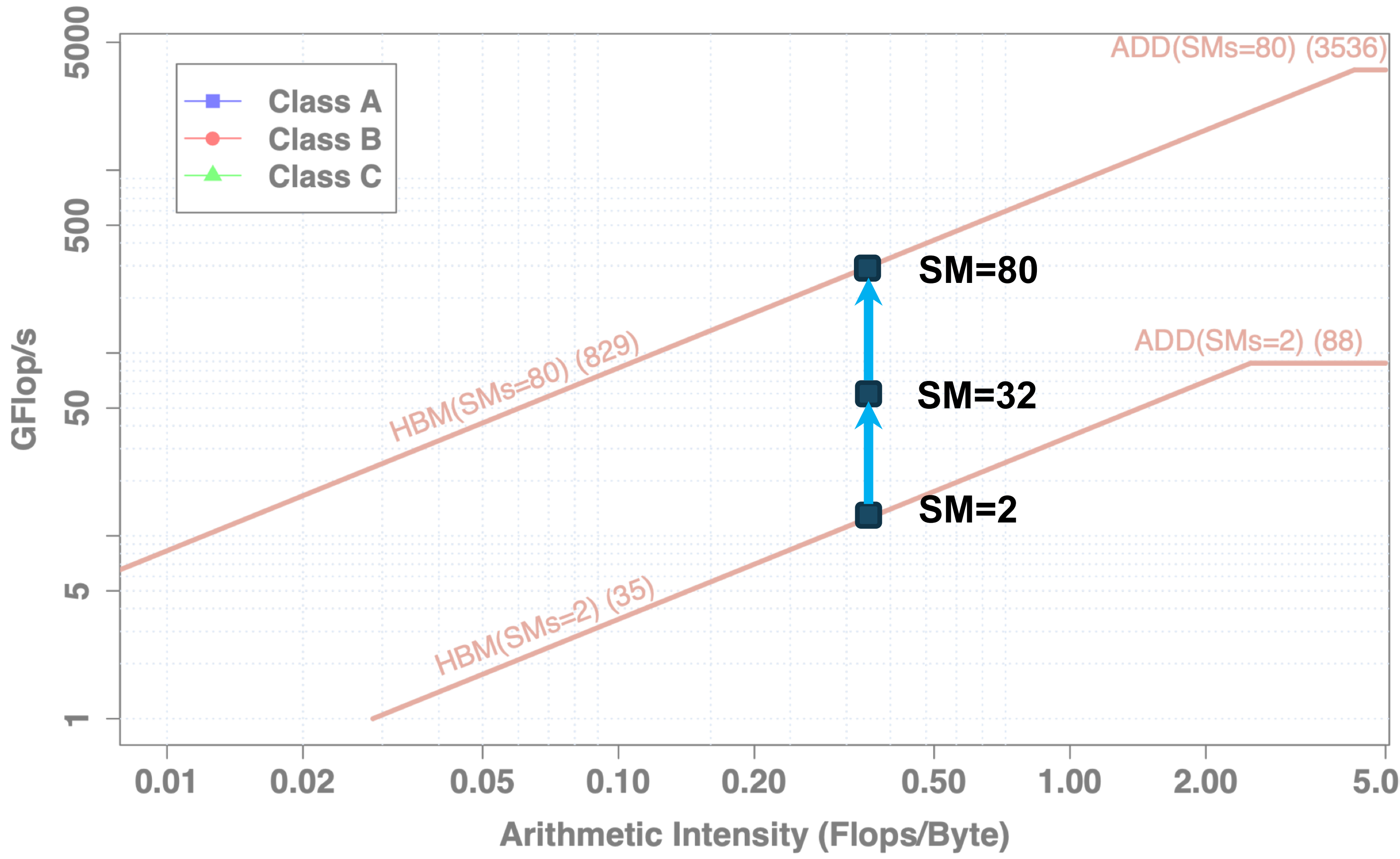
- Replot SM scalability as a trendline on the Roofline

- Define compute and bandwidth ceilings as a function of #SMs

- **Ideal behavior:**

  △*y* = increase in computational resources or share of BW

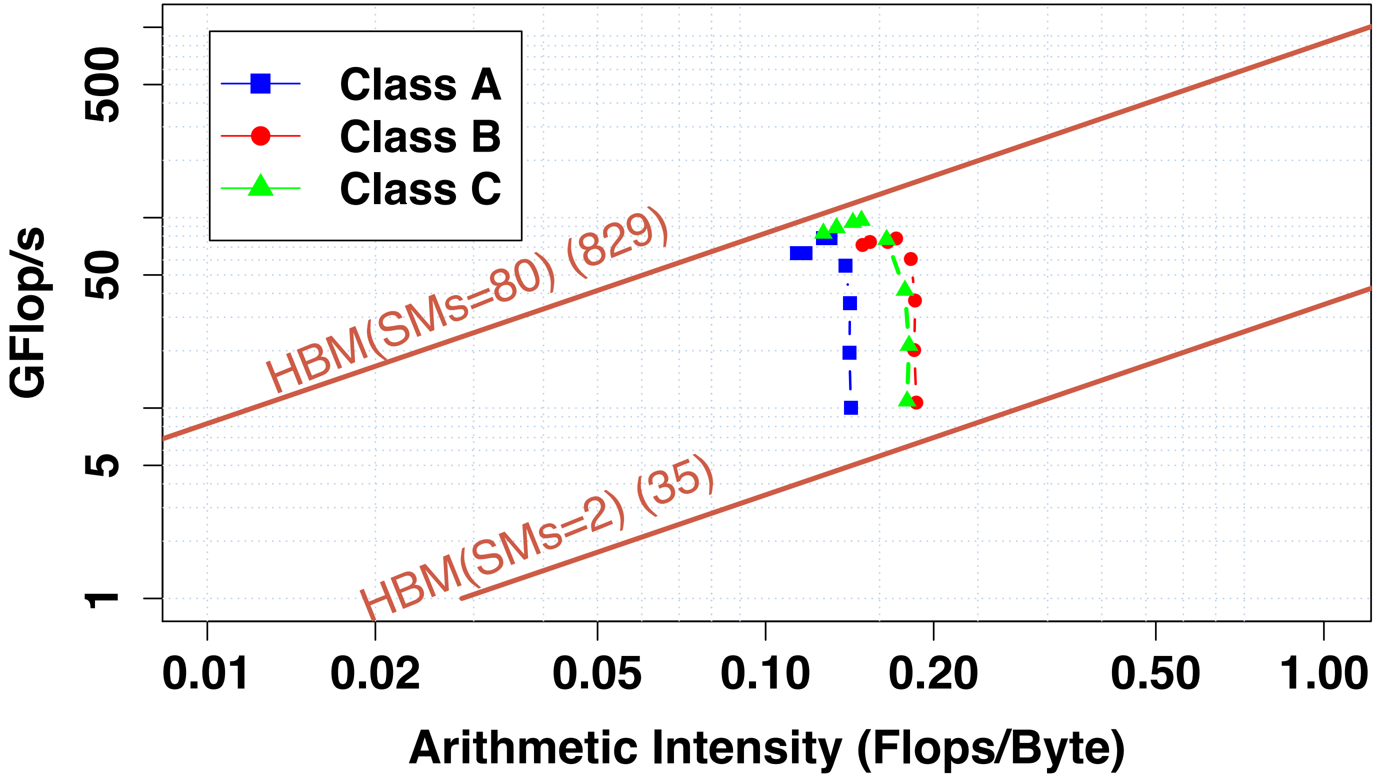
  △x=0 (No change in arithmetic intensity)



Khaled Ibrahim, Samuel Williams, Leonid Oliker, “Performance Analysis of GPU Programming Models using the Roofline Scaling Trajectories”, Bench, November, 2019.

Example #1 – NAS MG in OpenACC

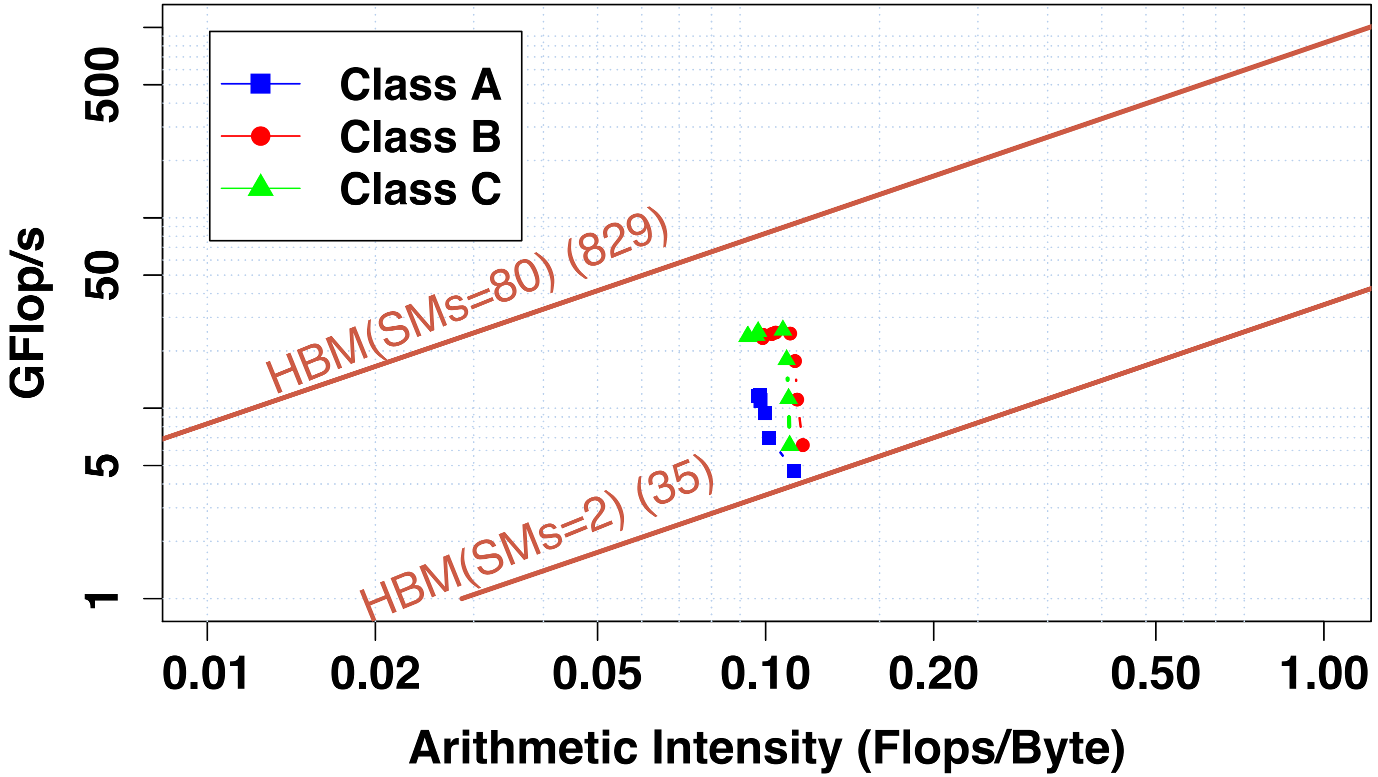
- Performance scales nearly linearly until high concurrency
- Fall over in performance…
    - HBM limited
    - Exhausts cache capacity
    - More SMs generate more capacity misses
    - AI degrades
    - Performance degrades



Khaled Ibrahim, Samuel Williams, Leonid Oliker, "Performance Analysis of GPU Programming Models using the Roofline Scaling Trajectories", Bench, November, 2019.

Example #2 – NAS FT in OpenACC

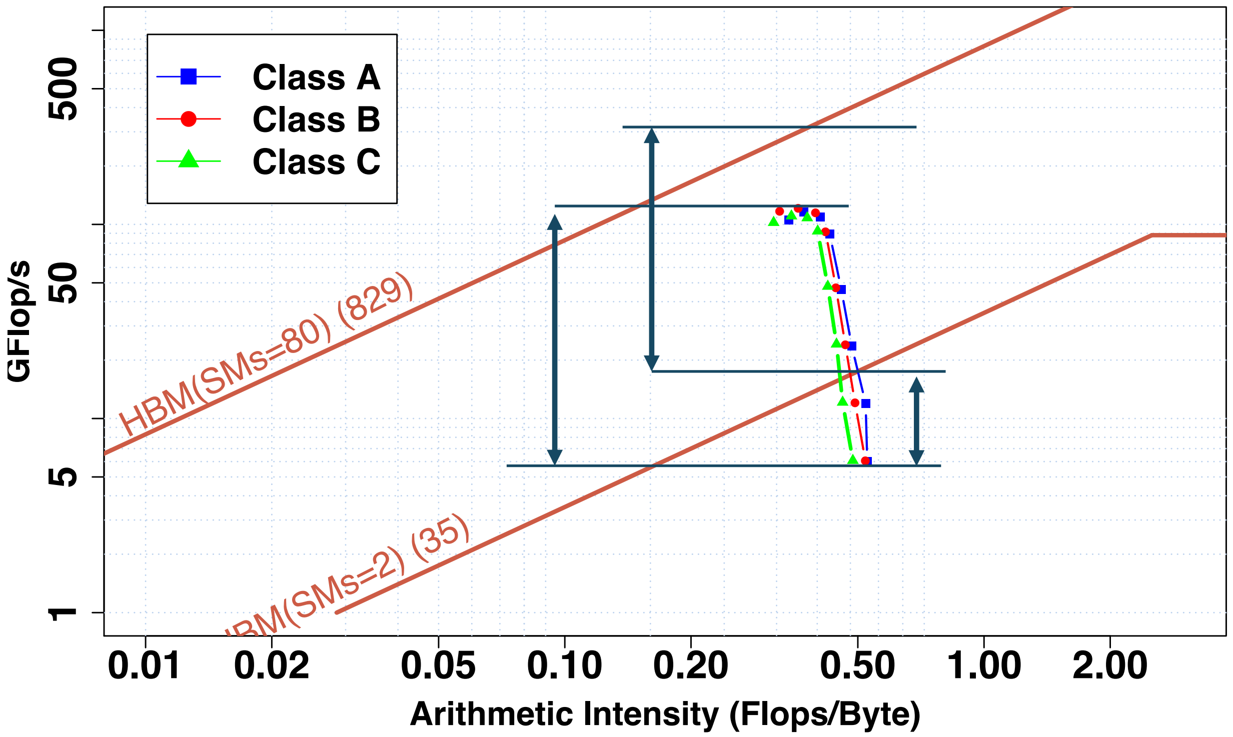
- Performance scales poorly
- Saturation in performance far below HBM ceiling
- Limited degradation in AI (few capacity misses)



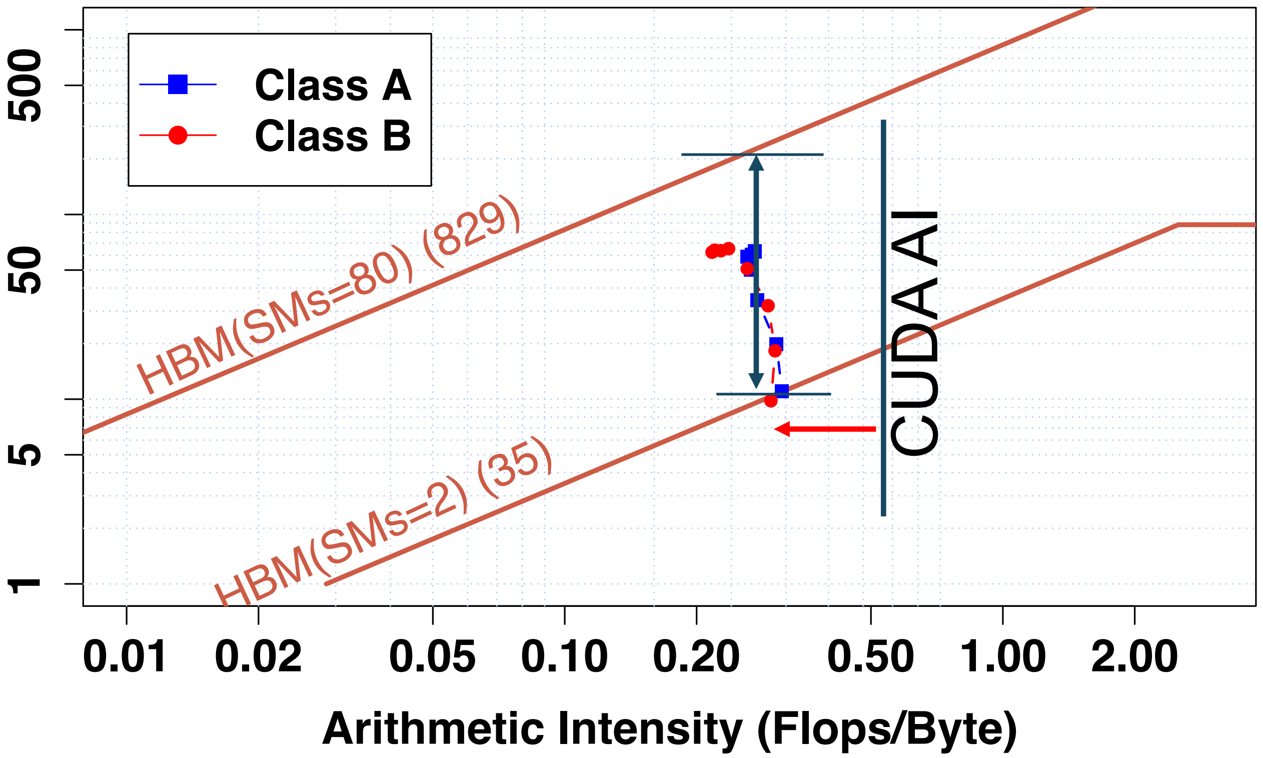
Khaled Ibrahim, Samuel Williams, Leonid Oliker, “Performance Analysis of GPU Programming Models using the Roofline Scaling Trajectories”, Bench, November, 2019.

Understanding Different Programming Models

BT CUDA Implementation



BT OpenACC Implementation



- CUDA delivered better AI and better scalability
- But CUDA efficiency was initially much lower
- Different compilers/PM have different challenges

Khaled Ibrahim, Samuel Williams, Leonid Oliker, “Performance Analysis of GPU Programming Models using the Roofline Scaling Trajectories”, Bench, November, 2019.

Summary

- Recasts thread scalability into the Roofline model

- Quantitative analysis of different implementations, programming models, or compilers

- Infer cache behavior and efficiency

- ***Codes that demonstrate a good Roofline Scaling Trajectory will likely scale to Perlmutter, Frontier, and Aurora***

Khaled Ibrahim, Samuel Williams, Leonid Oliker, “Performance Analysis of GPU Programming Models using the Roofline Scaling Trajectories”, Bench, November, 2019.

Questions?