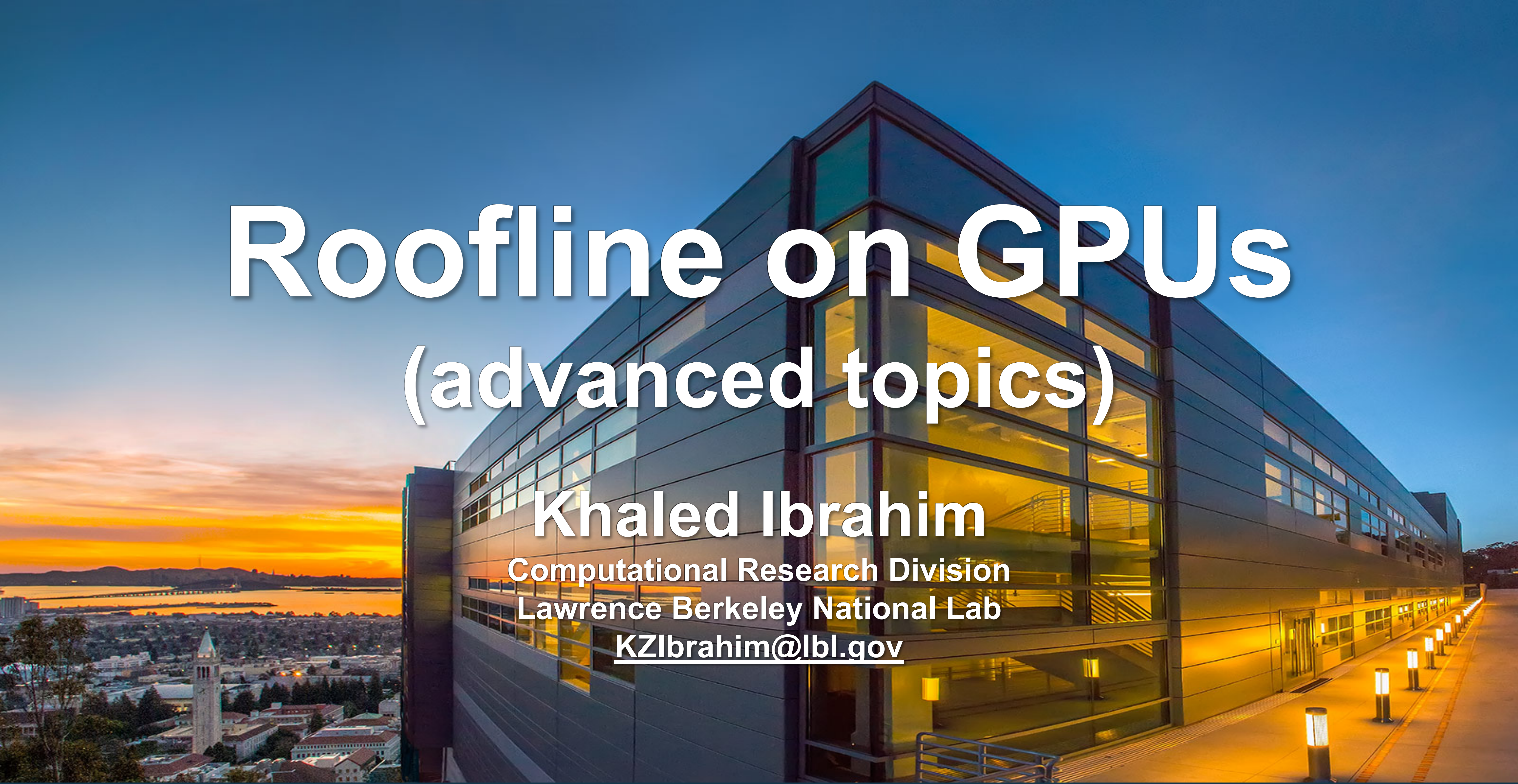


# Roofline on GPUs

## (advanced topics)

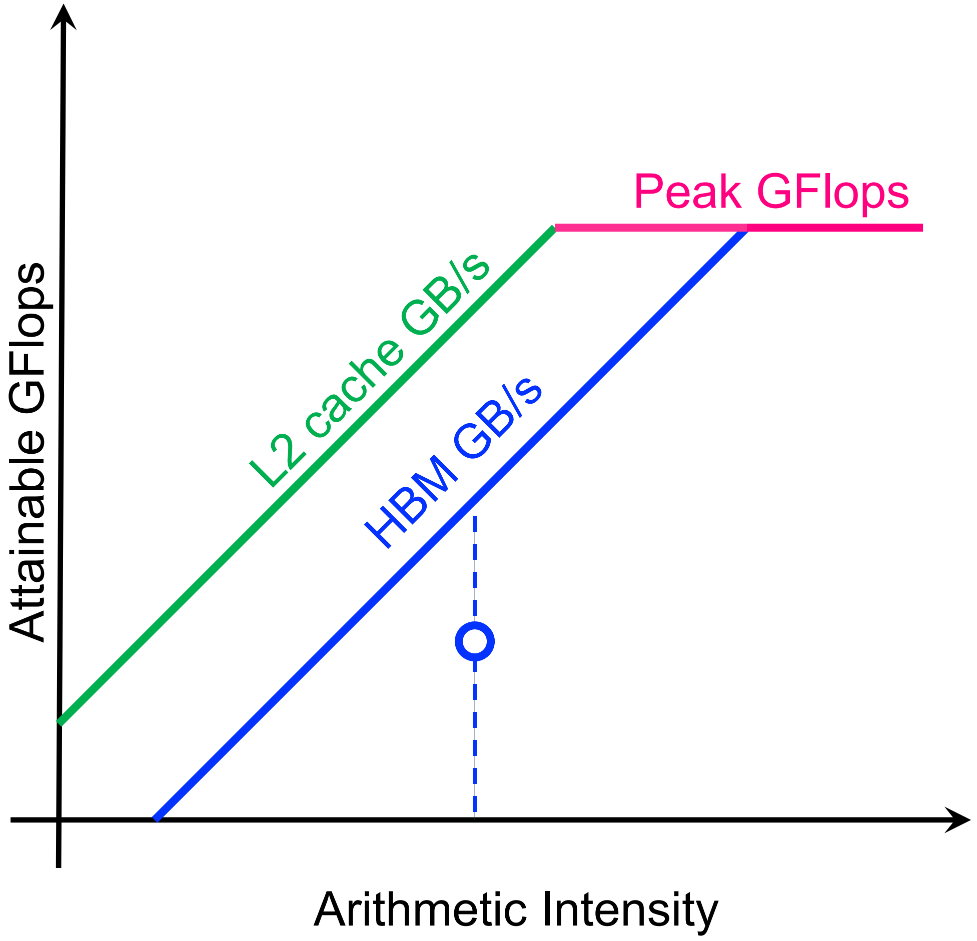
Khaled Ibrahim

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

# Acknowledgements

- Materials were provided by Samuel Williams, 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.

# Motivation - Why am I not hitting the Roofline?



# Roofline Scaling Trajectories

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

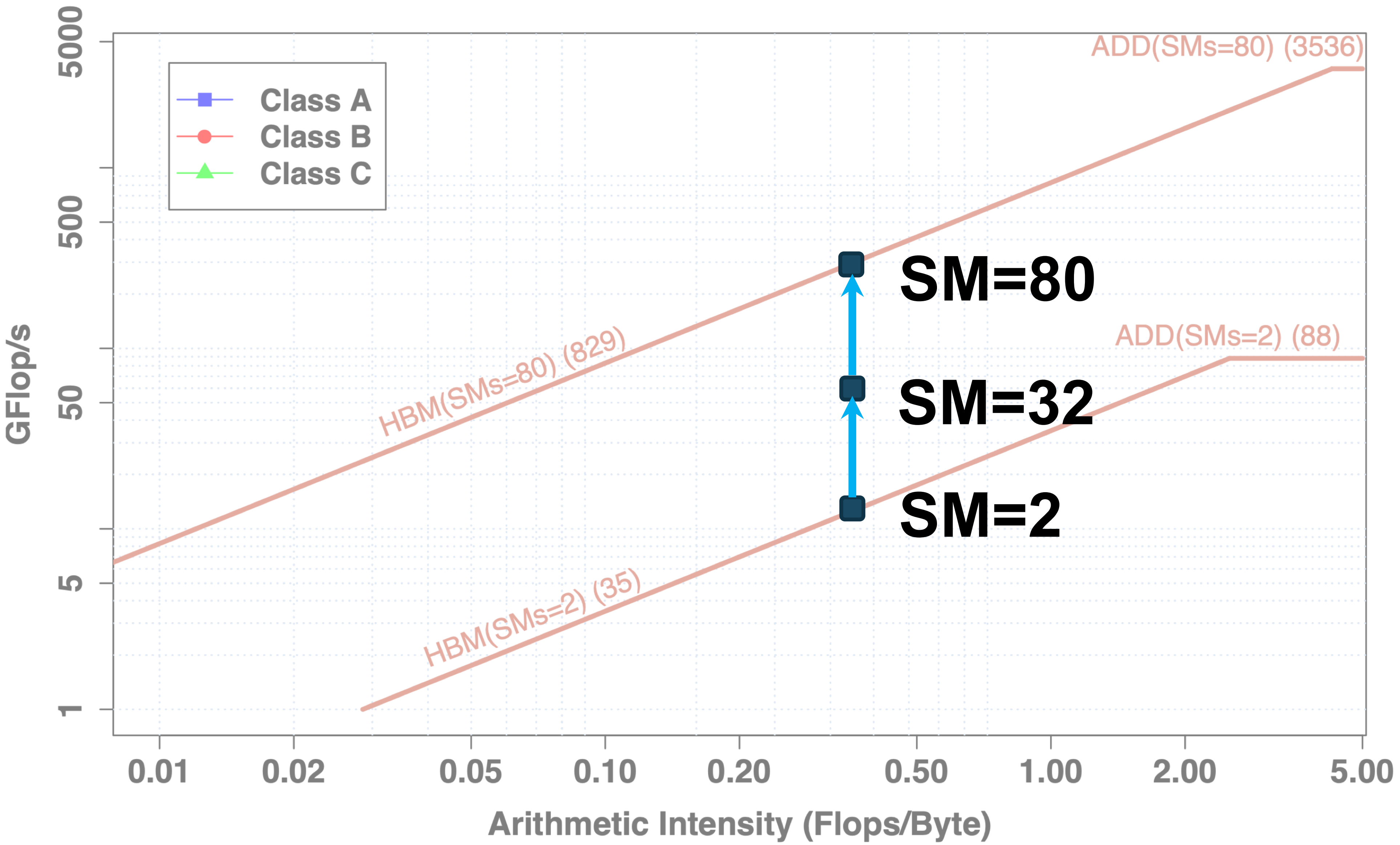
# Roofline Scaling Trajectory

## Visualization Method

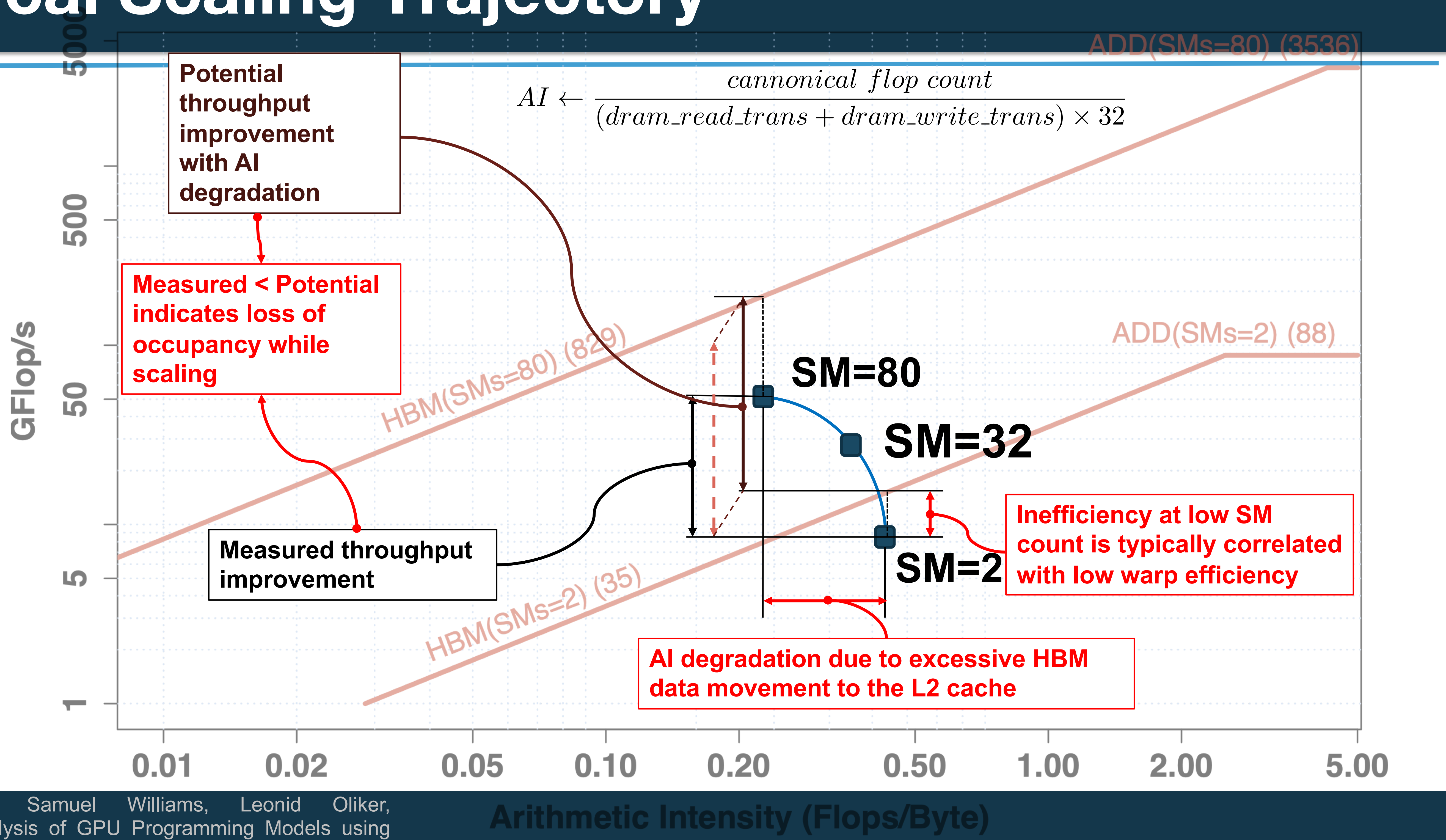
- Define compute and bandwidth ceilings as a function of #SMs
- Plot App scaling as a trendline on the Roofline

## Ideal Scaling

- △y = increase in computational resources or share of BW
- △x=0 (No change in arithmetic intensity)



# Typical Scaling Trajectory



$$AI \leftarrow \frac{cannonical\ flop\ count}{(dram\_read\_trans + dram\_write\_trans) \times 32}$$

Potential throughput improvement with AI degradation

Measured < Potential indicates loss of occupancy while scaling

Measured throughput improvement

Inefficiency at low SM count is typically correlated with low warp efficiency

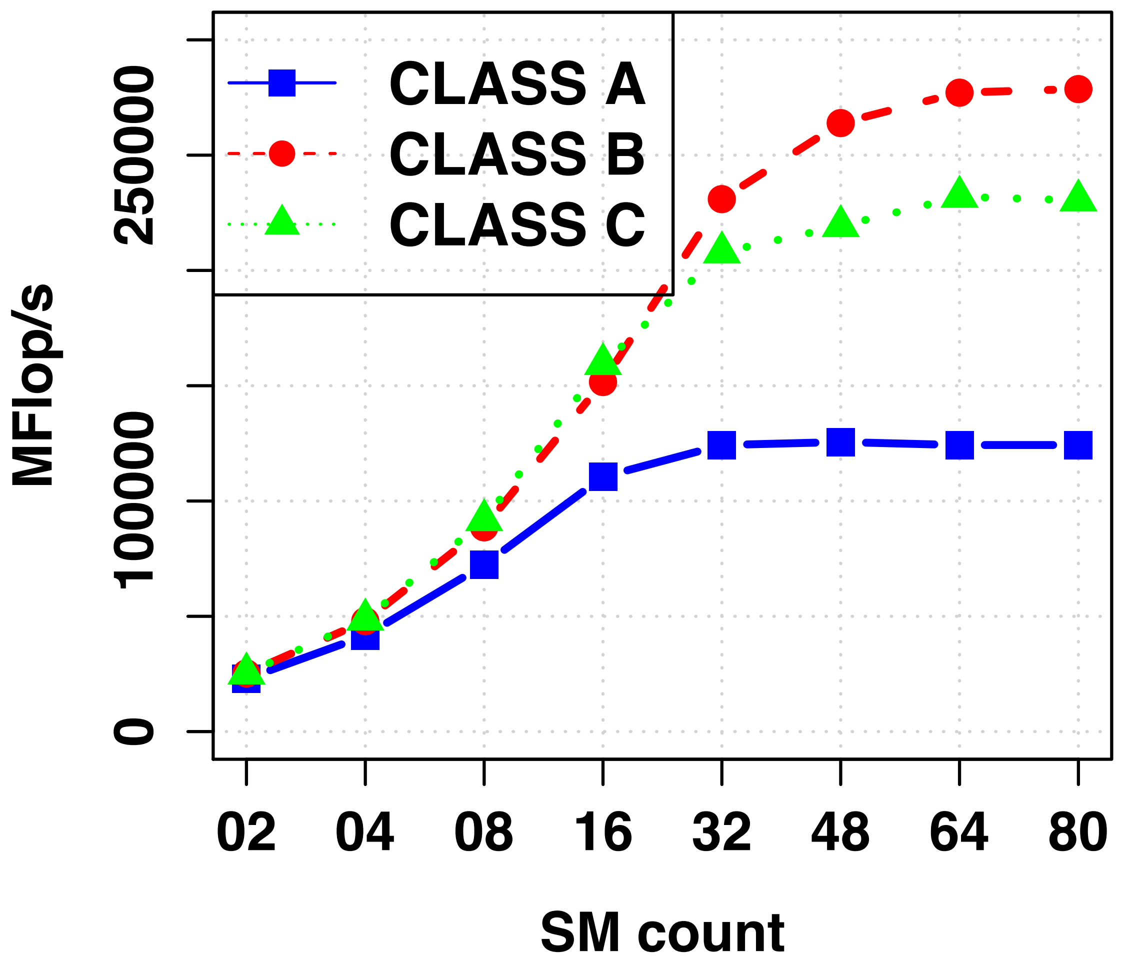
AI degradation due to excessive HBM data movement to the L2 cache

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

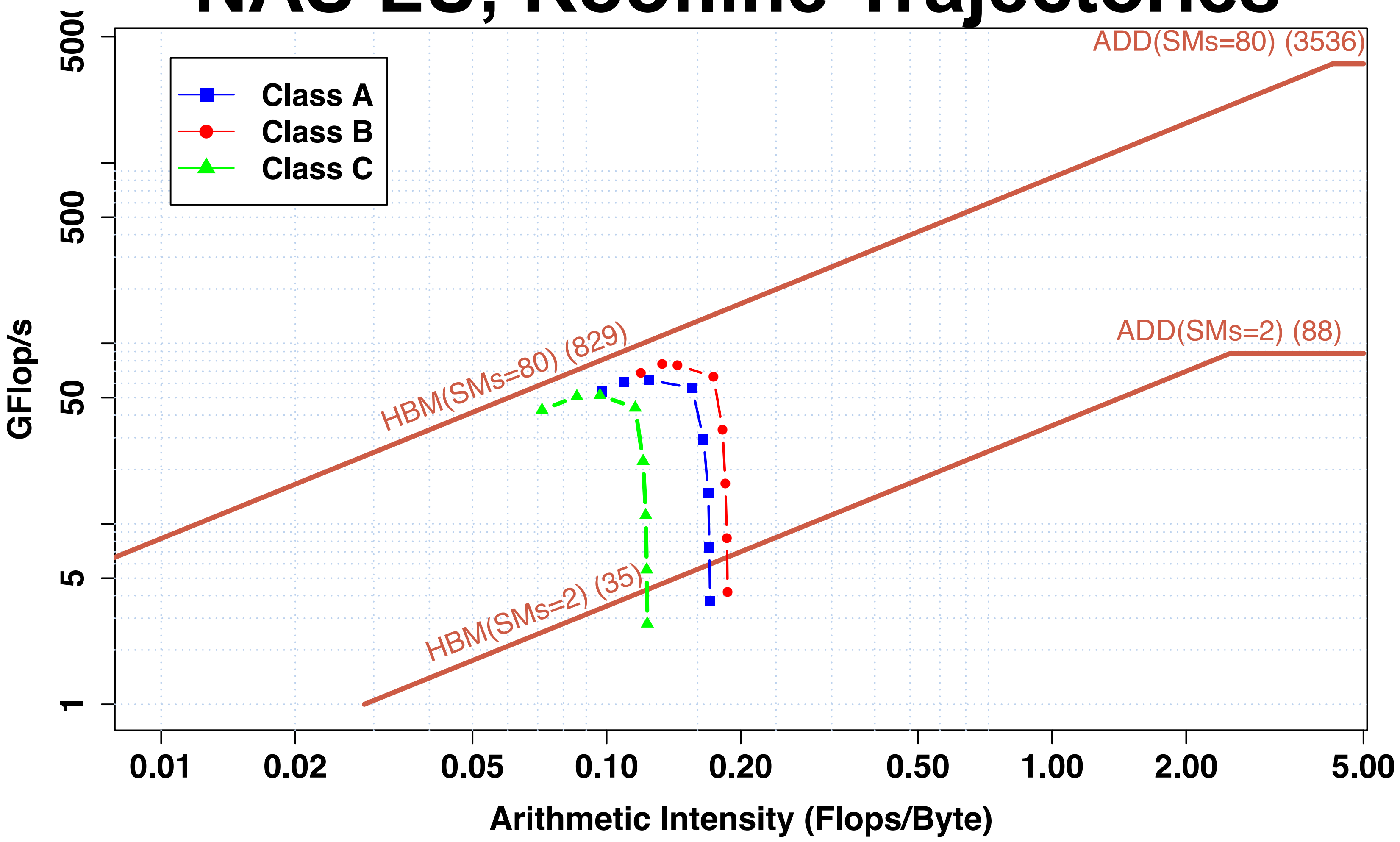
# Understanding the Scaling Trends

- **Scaling plot vs. Roofline scaling trajectory**

## NAS LU, Scaling Plot

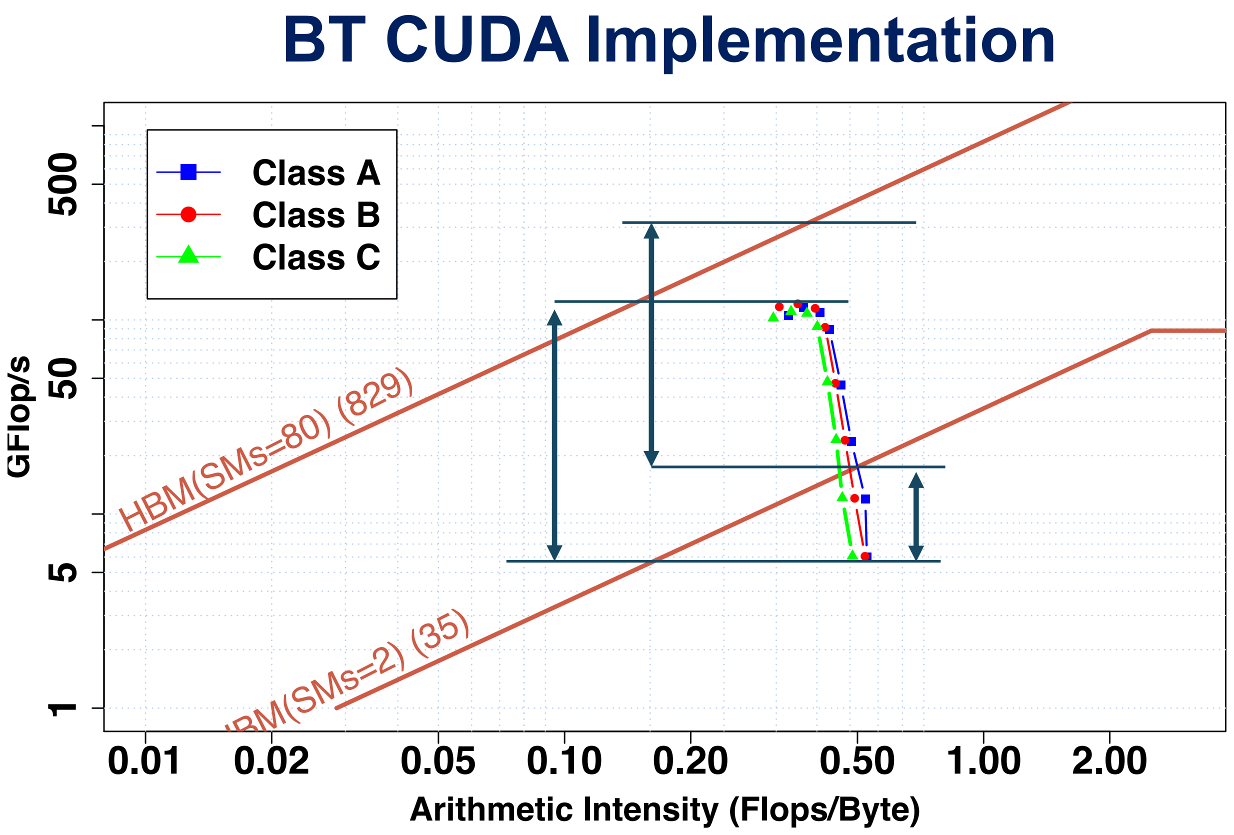


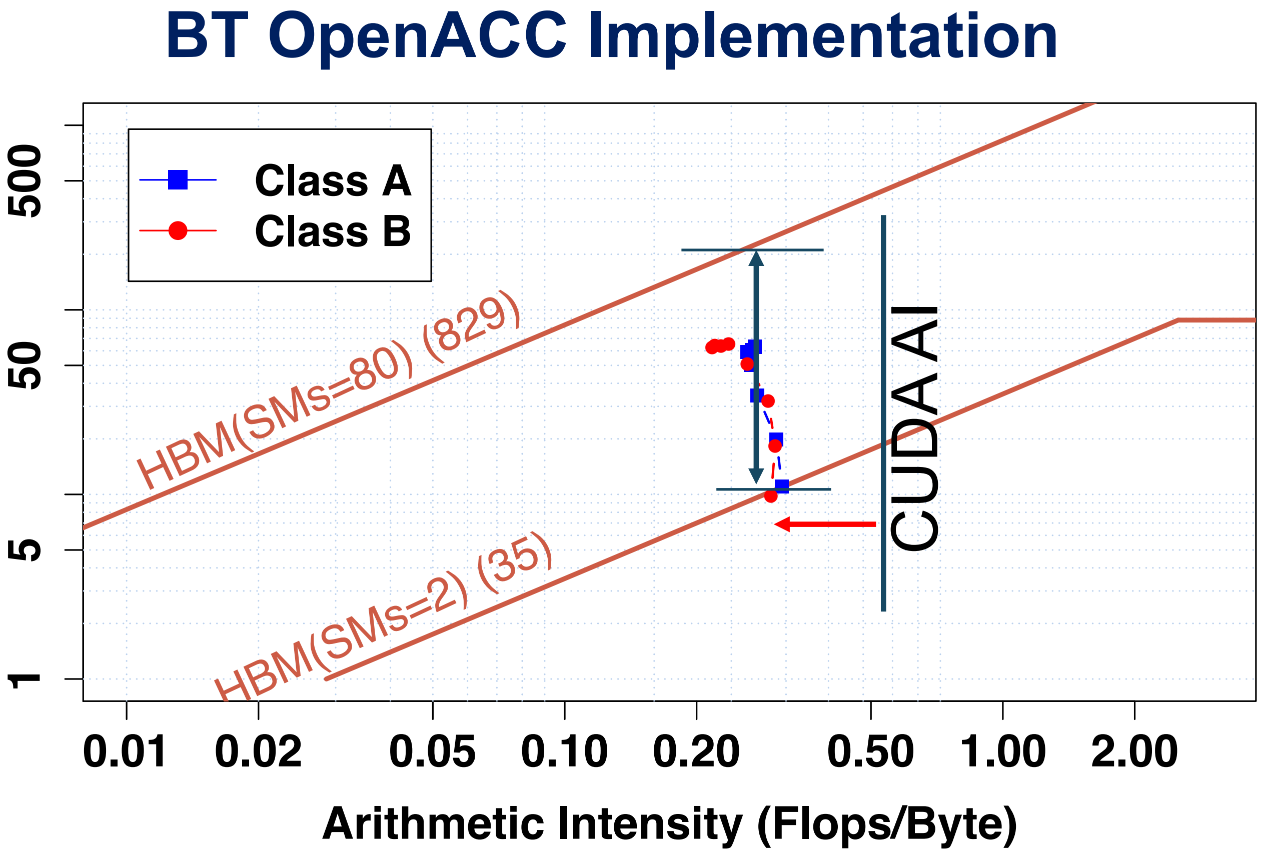
## NAS LU, Roofline Trajectories



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

# Understanding Different Programming Models





- Is the performance issue related to occupancy or warp efficiency?
  - Is the programming model influencing occupancy and warp efficiency?
- 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.

# Roofline Trajectories Takeaway

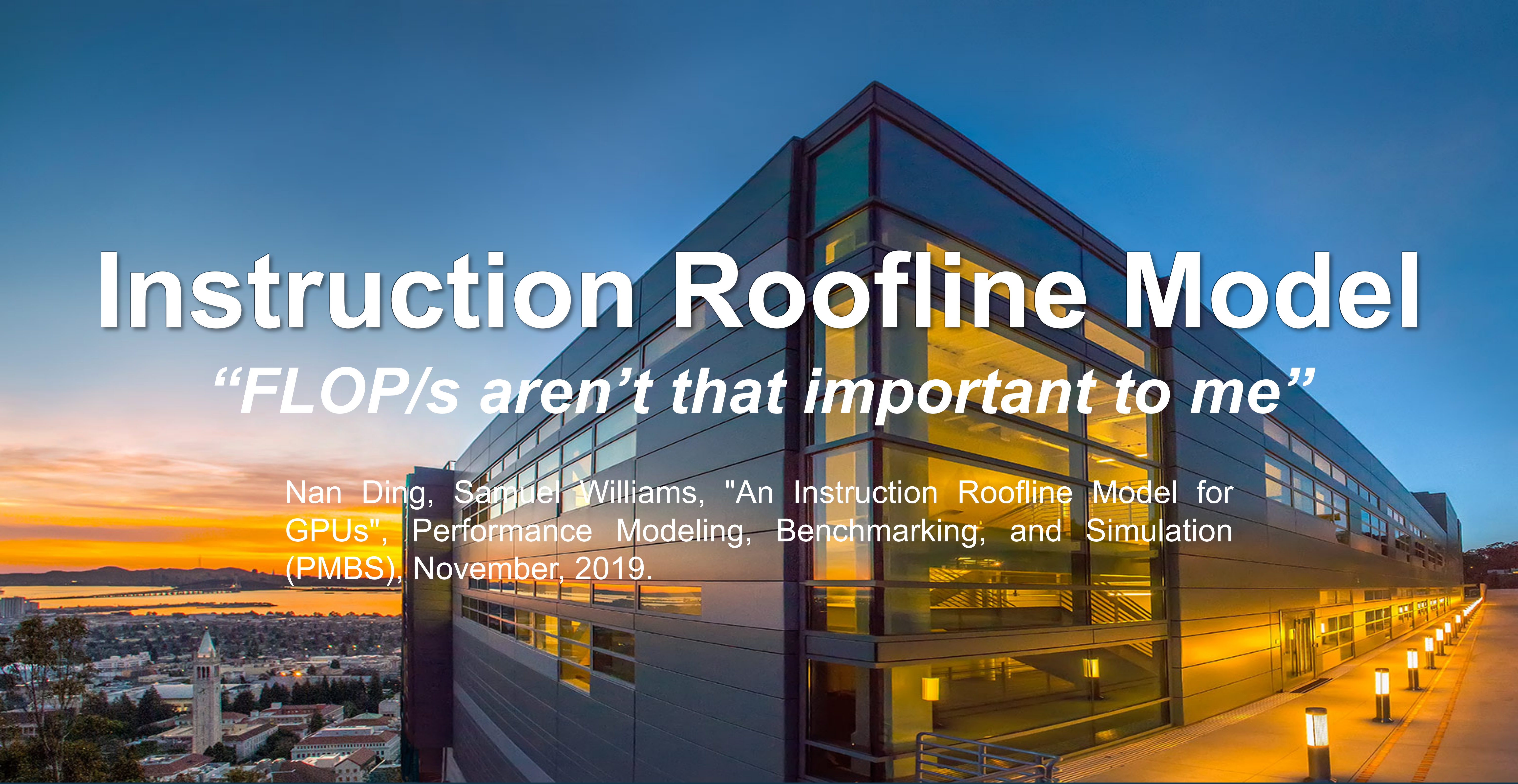
## Traditional Roofline

- **Tells us about performance** *(floating-point)*
- Performance under full utilization of computational resources

## Roofline Scaling Roofline

- **Tells us about scaling bottenecks**
- Incremental scaling of resources (possibly changing what resources are stressed)
- Quantitative analysis of different implementations, programming models, or compilers
- ***Understand potential performance change while migrating code to Perlmutter, Frontier, and Aurora***

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



# 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 to Analyze non-FLOP Application?

- 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”?**
  - **Instruction Fetch/Decode/Issue bottlenecks?**
  - **Functional Unit Bottlenecks?**

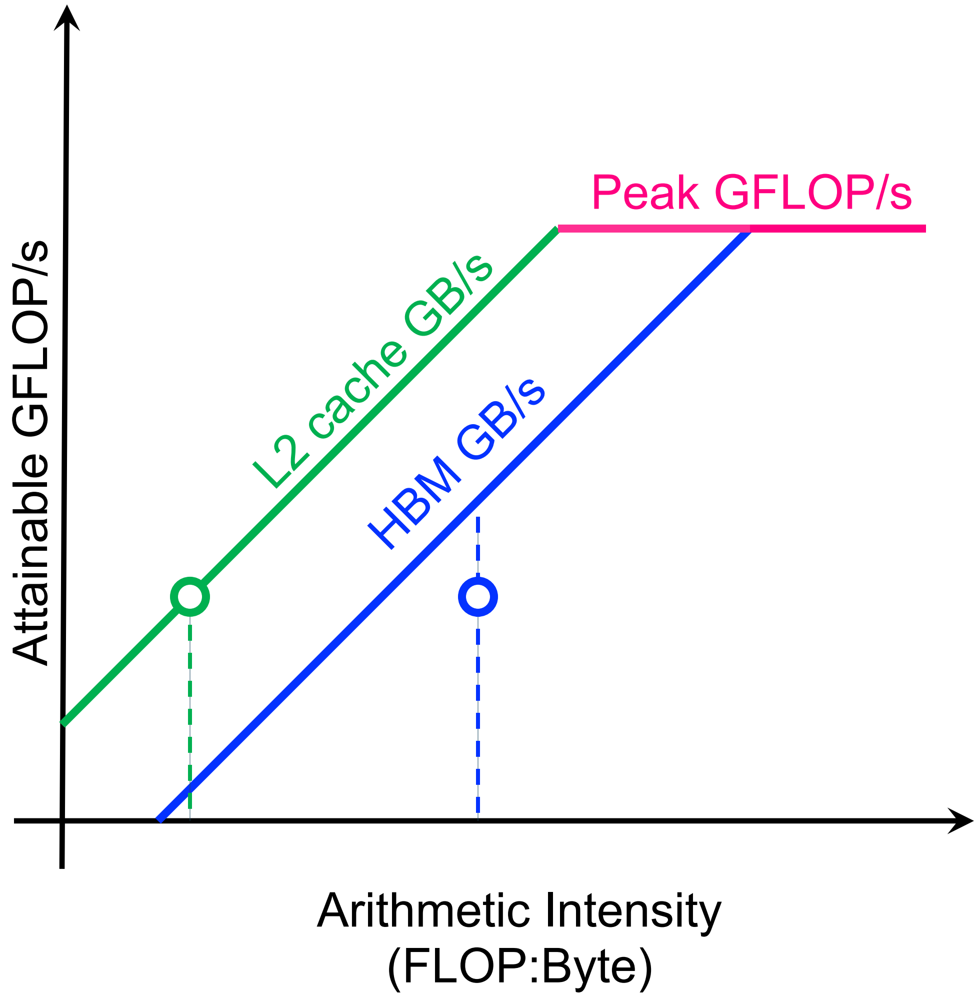
  ➢ **Need to create a true instruction Roofline**

# NVIDIA GPU Instruction Roofline

- 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**
- Conventionally, 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

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



# Instruction Roofline

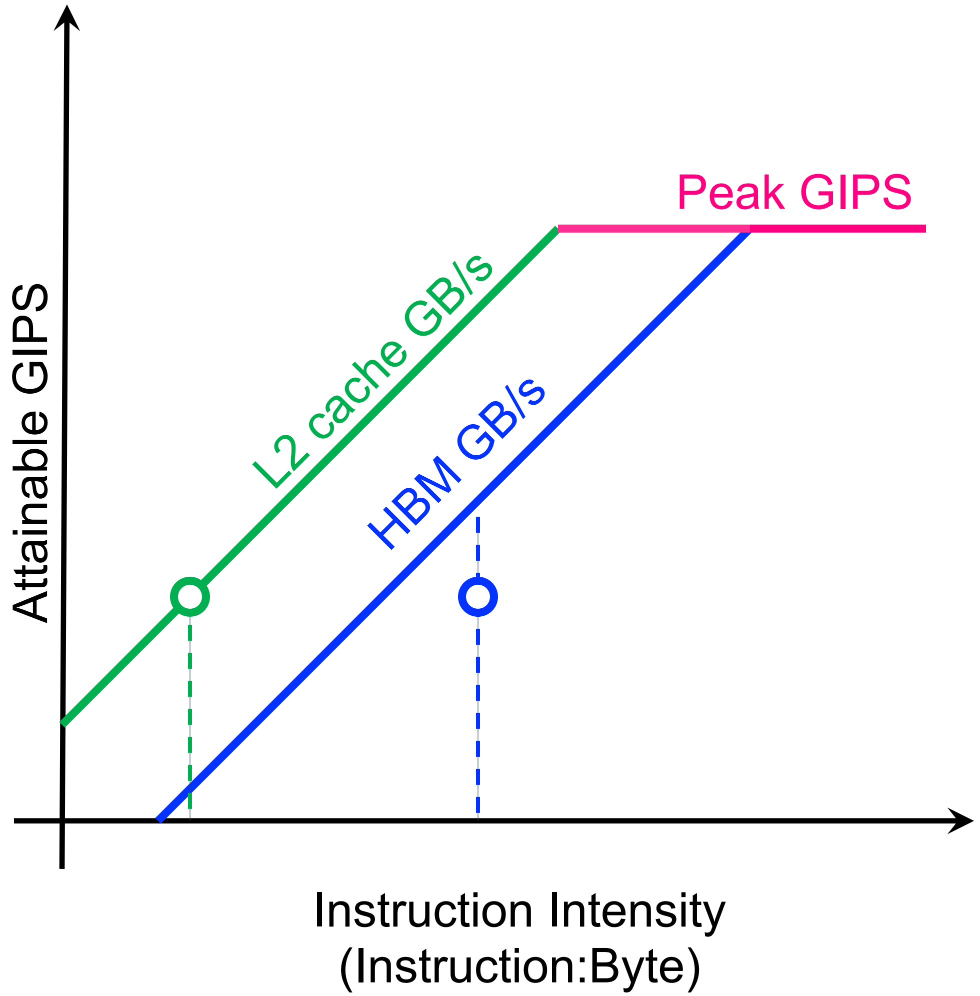
$$\text{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ \text{AI}_{\text{DRAM}} * \text{DRAM GB/s} \end{cases}$$

$$\Downarrow$$

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

↑

***Instructions per Byte***



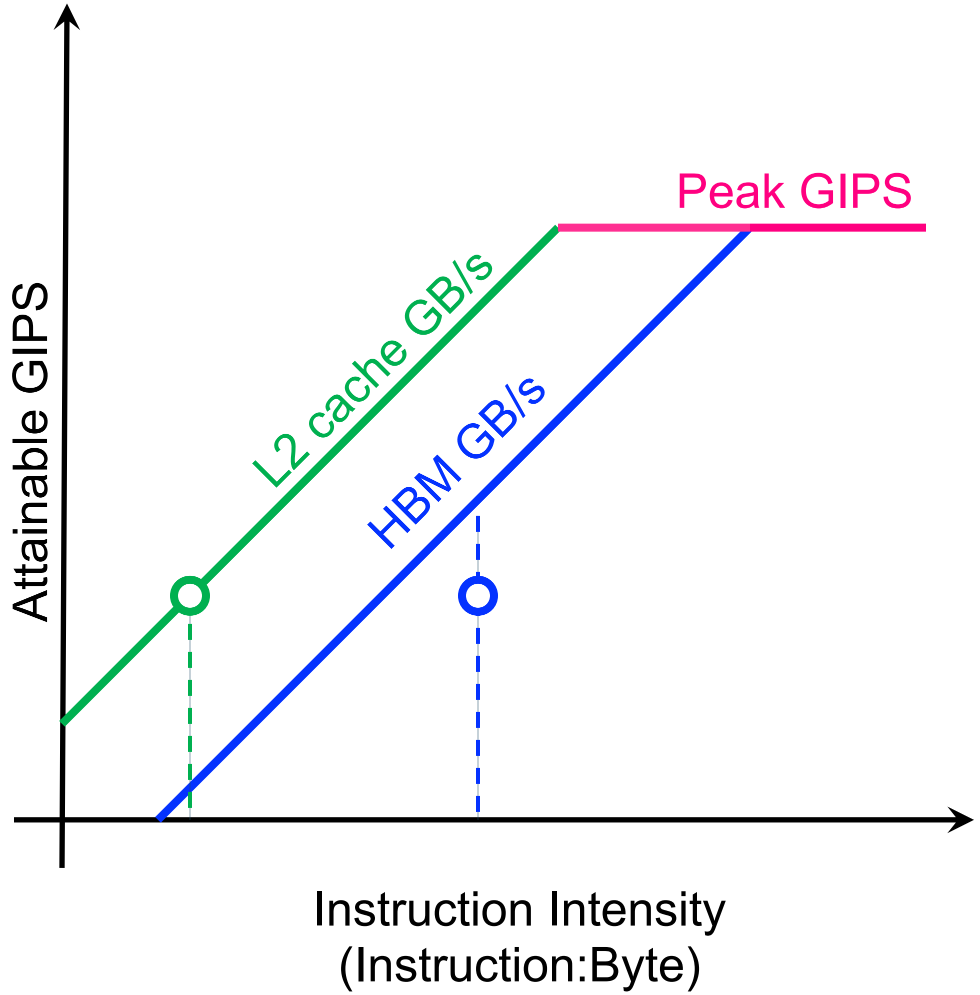
Nan Ding, Samuel Williams, "An Instruction Roofline Model for GPUs", PMBS, November, 2019.

# Instruction Roofline on GPUs

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

Warp Instructions

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



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

# Instruction Roofline on GPUs

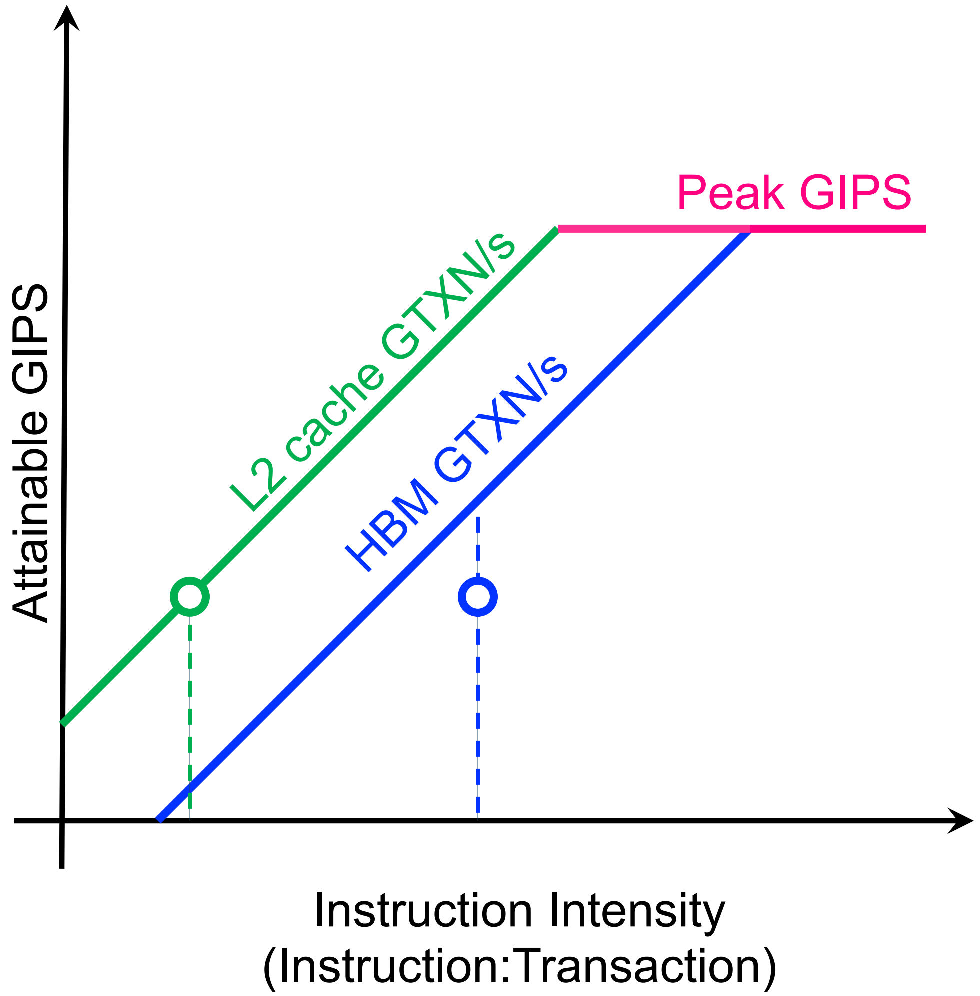
$$\text{GFLOP/s} = \min\begin{cases}\text{Peak GFLOP/s}\\ \text{AI}_{\text{DRAM}} * \text{DRAM GB/s}\end{cases}$$

$$\Downarrow$$

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

$$\Downarrow$$

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



*$II_x$ (Instruction Intensity at level "x") =*

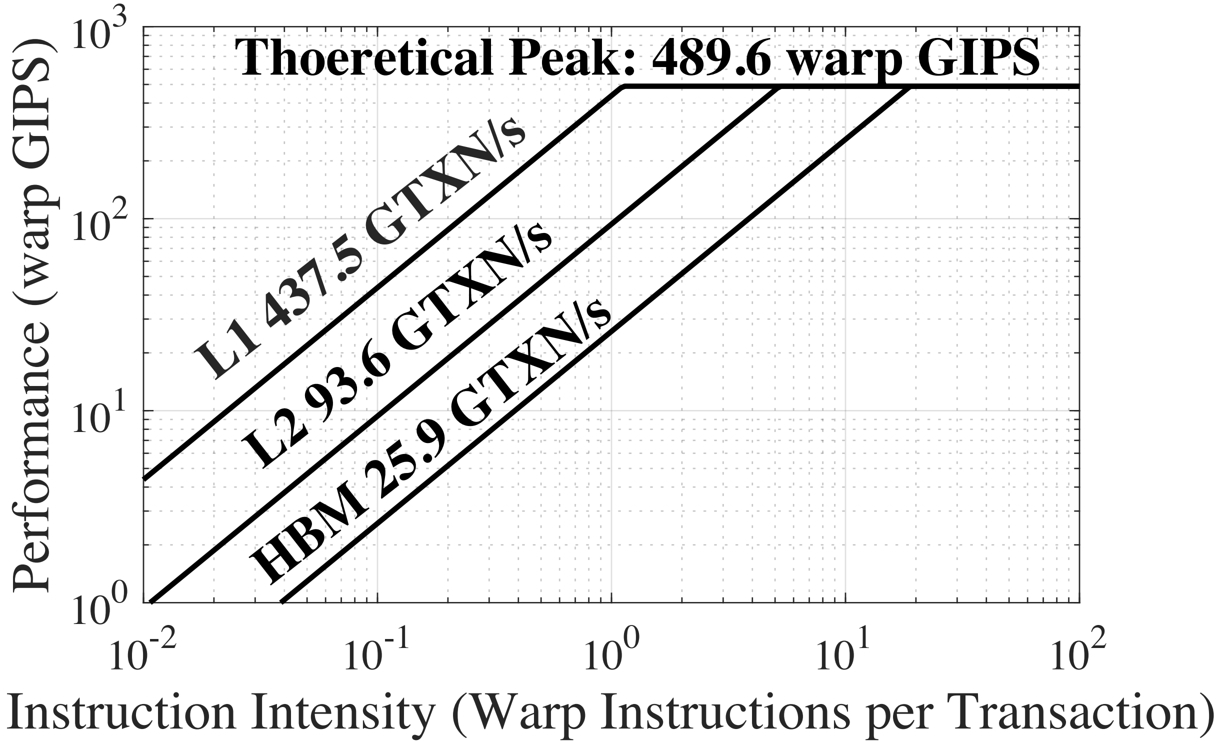
*Instructions / Transactions (to/from level "x" )*

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**)

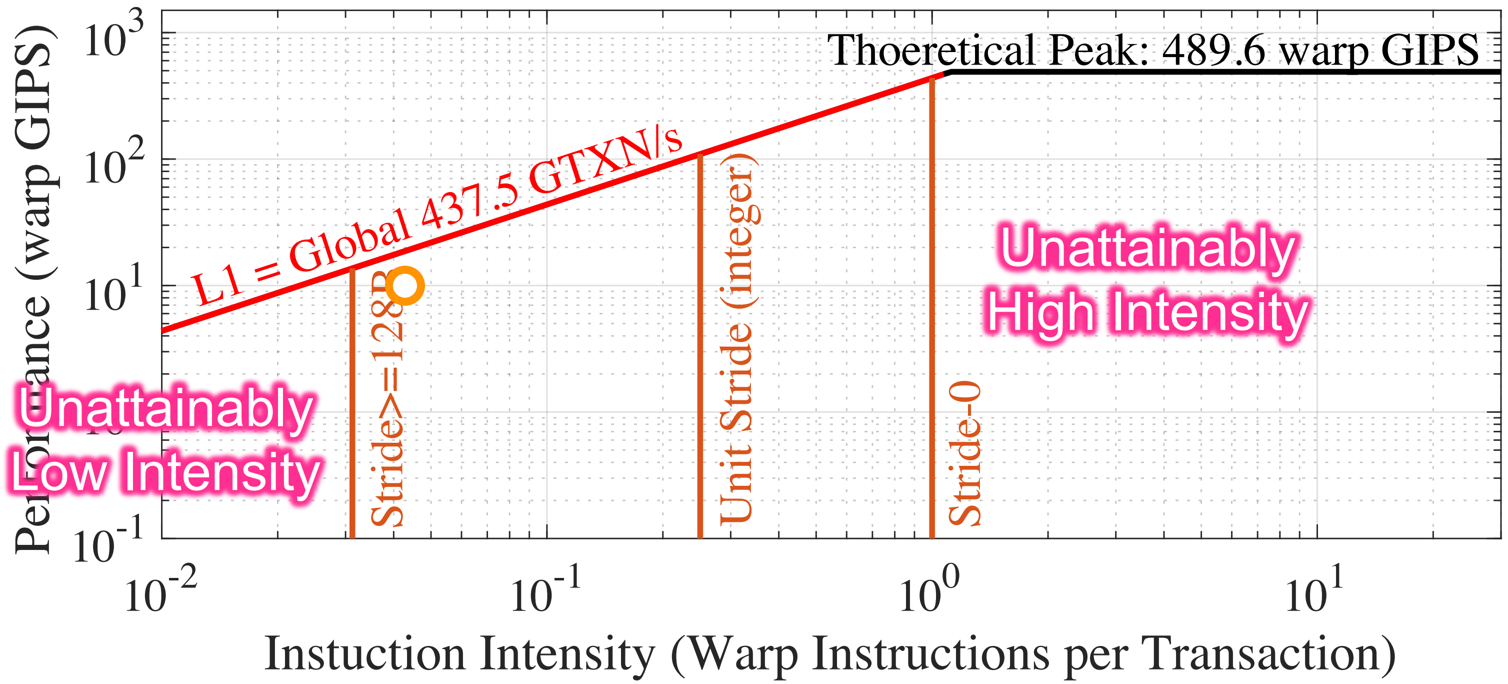
- Peak Performance and Peak Bandwidths
  - Instruction:
    - 80 SMs * 4 warps * 1.53GHz ~ 490 GIPS (warp-level)
  - Use ERT for memory (convert from GB/s)
    - L1: 80 SMs * 4 transactions/cycle * 1.53 GHz ~ 490 GTXN/s
    - L2: 94 GTXN/s (empirical)
    - HBM: 26 GTXN/s (empirical)



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

# Efficiency of Global Memory Access

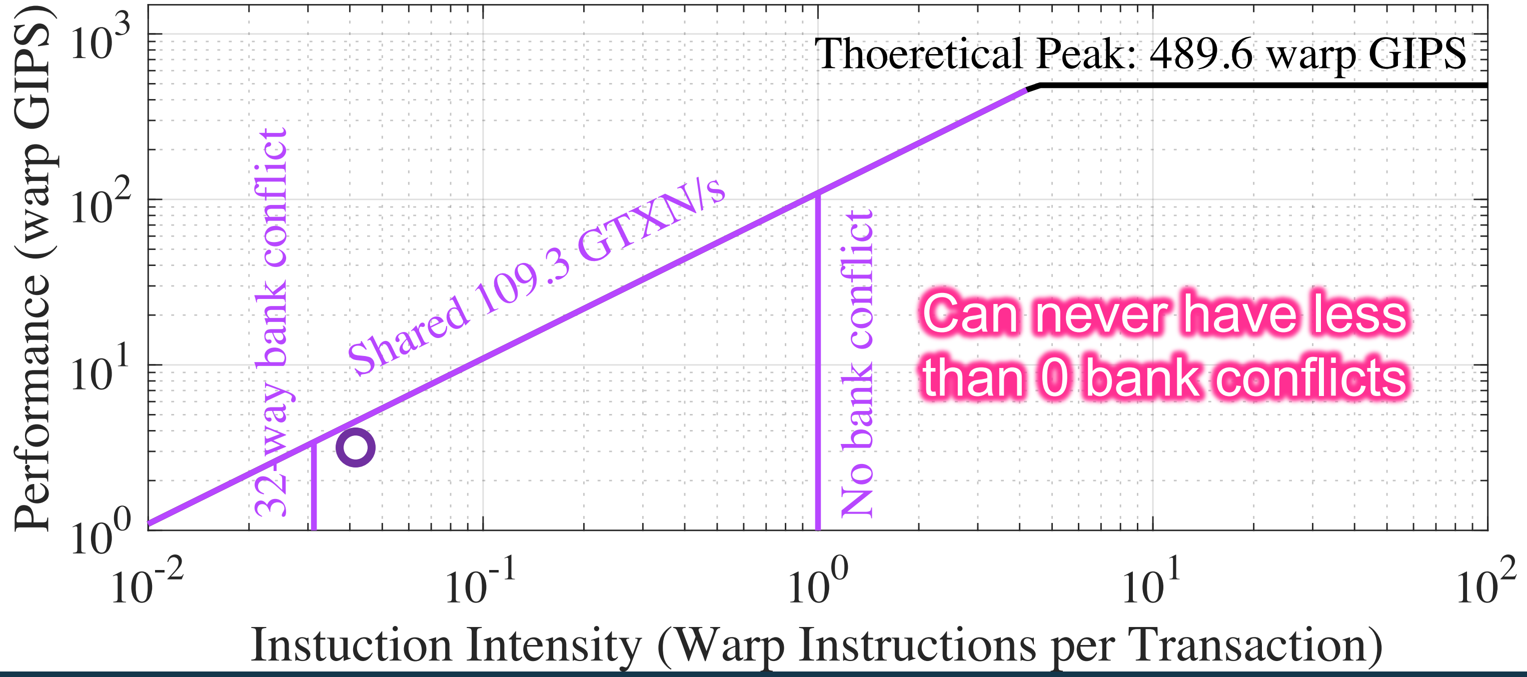
- (Global)LDST Instruction Intensity has a special meaning / use…
  - Global LDST instructions / Global transactions
  - Numerator lower than nominal II
  - Denominator can be lower than nominal L1 II (no local or shared transactions)
- Denotes efficiency of memory access
- 3 “**Walls**” of interest: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

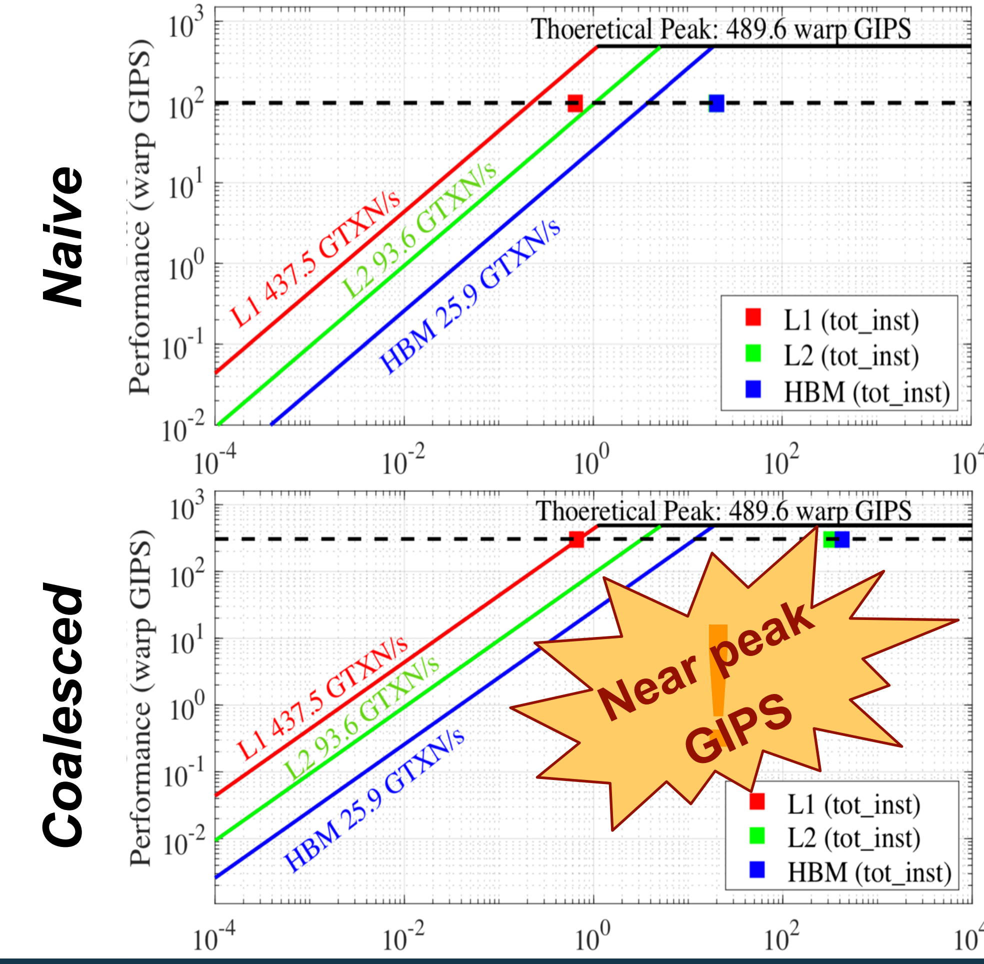


Nan Ding, Samuel Williams, "An Instruction Roofline Model for GPUs", PMBS, November, 2019.

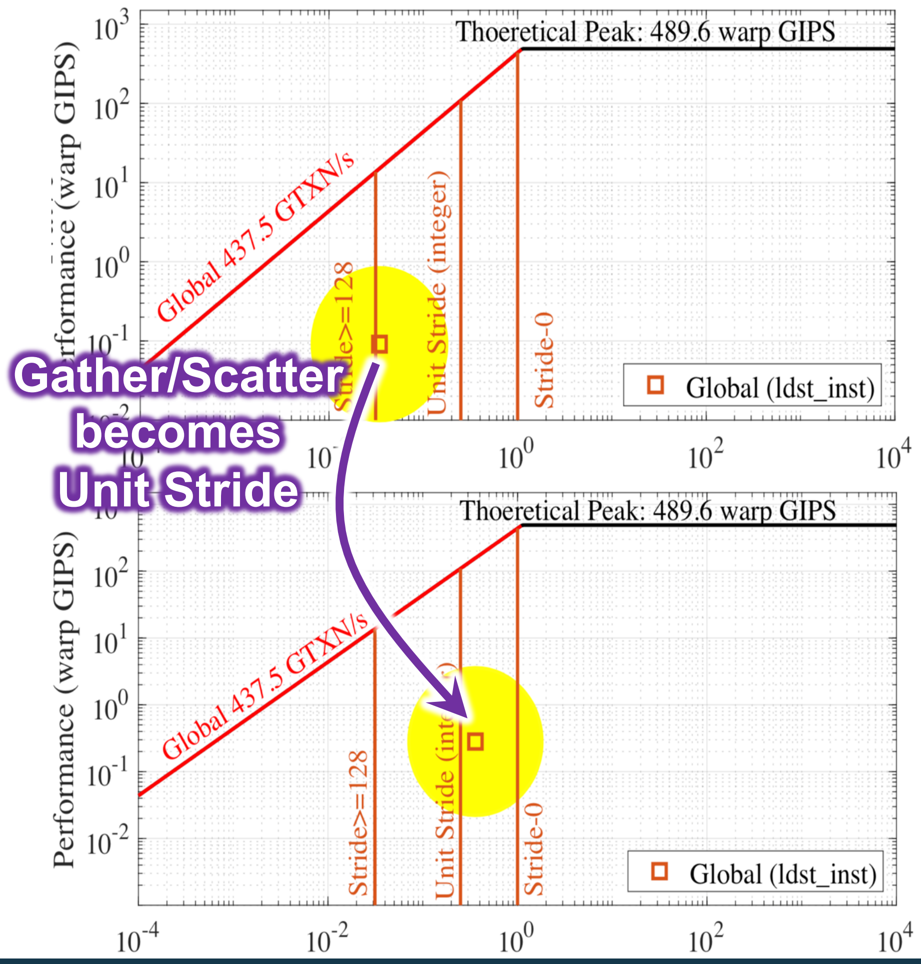
# Instruction Roofline for Smith-Waterman

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

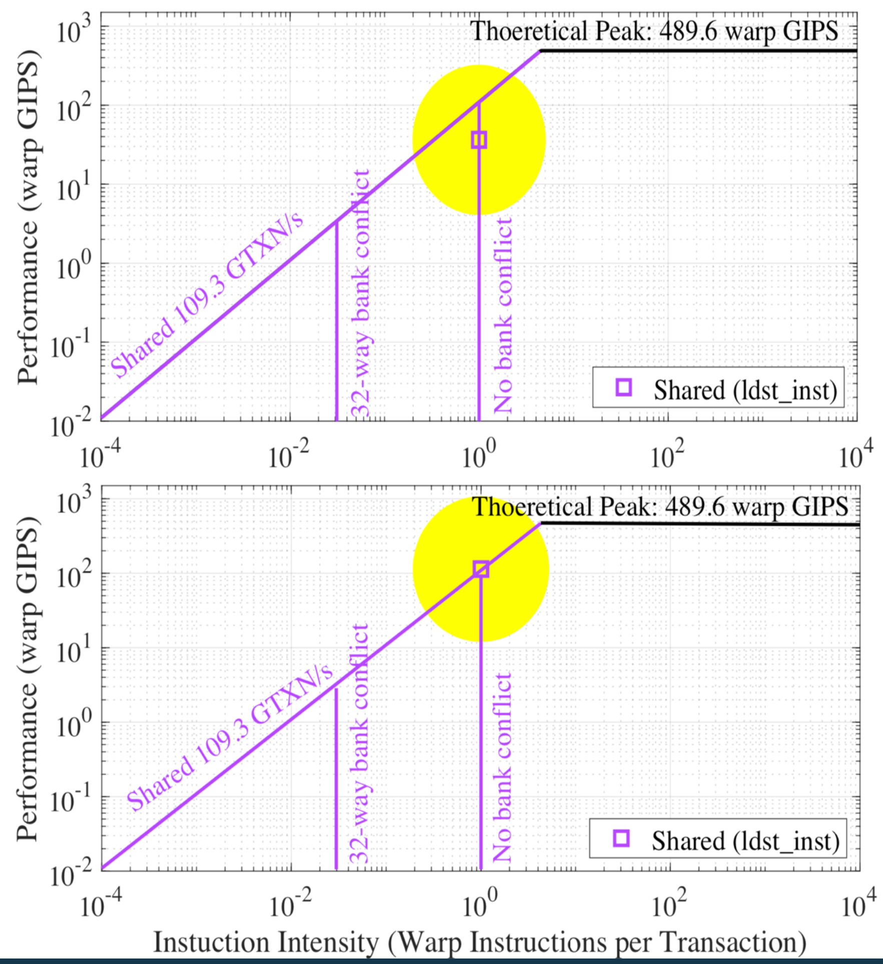
***Instruction Hierarchy & Thread Predication***



***Global Memory Efficiency***



***Shared Memory Efficiency***



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

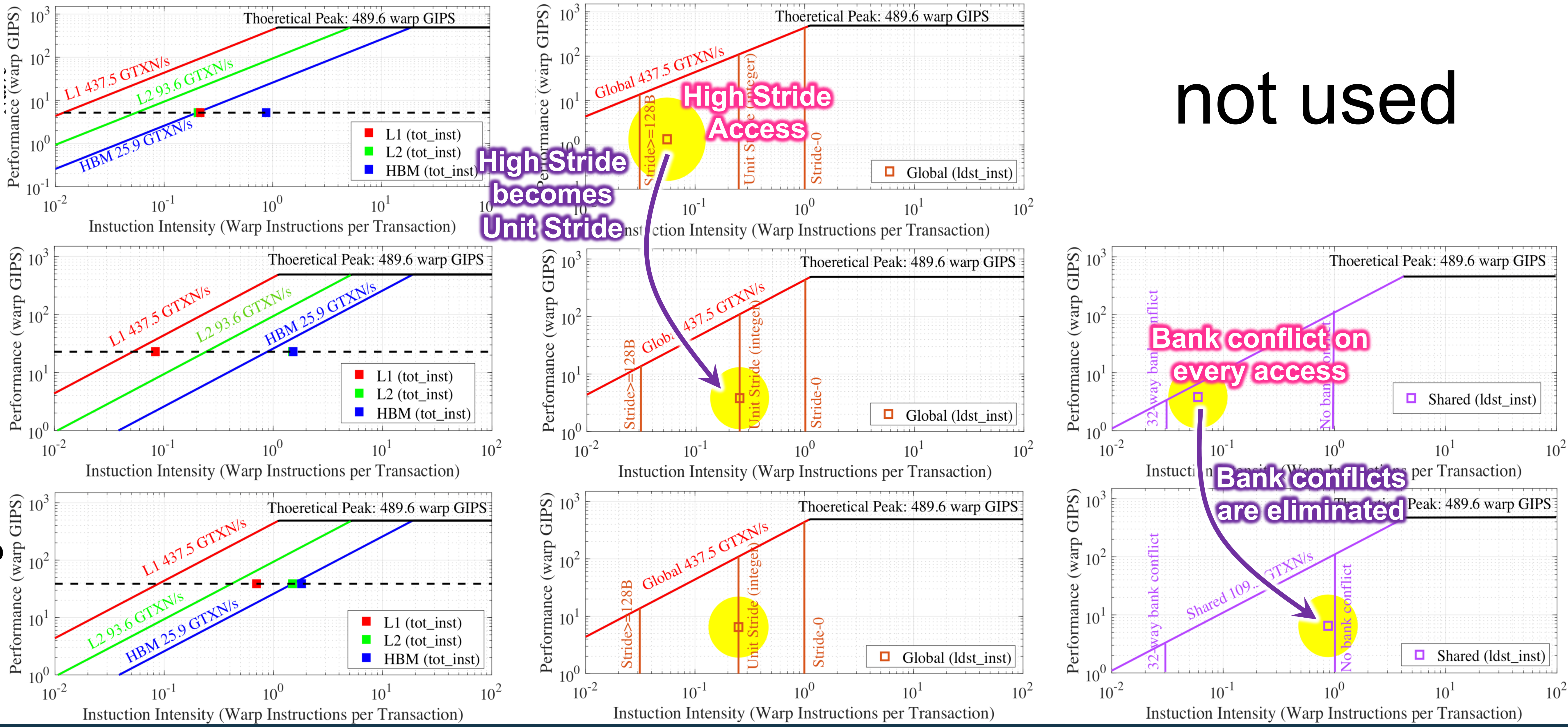
# Instruction Roofline for Matrix Transpose

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

***Naive***

***Transpose in Shared***

***Array Padding***



High Stride Access

High Stride becomes Unit Stride

not used

Bank conflict on every access

Bank conflicts are eliminated

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 effect 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 effect 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.

# Roofline for Performance Tuning

- Performance tuning
  - Is the application limited by a fundamental architectural limits (BW, flop rate)?
  - If not, what are the likely causes for inefficiency?
- Multiple techniques depending on the level of analysis
  - Hierarchal Roofline
  - Roofline Trajectories
  - Instruction Roofline
- Suggested recipe to leverage these techniques
  - Hierarchal Roofline (are you utilizing resources efficiently? If no, use trajectories)
  - Roofline Trajectories (is your problem warp efficiency or occupancy? If warp efficiency, do instruction roofline. If occupancy issue, how to manage or extract more parallelism?)
  - Instruction Roofline (What is the root cause for inefficient warp execution?)

# Questions?