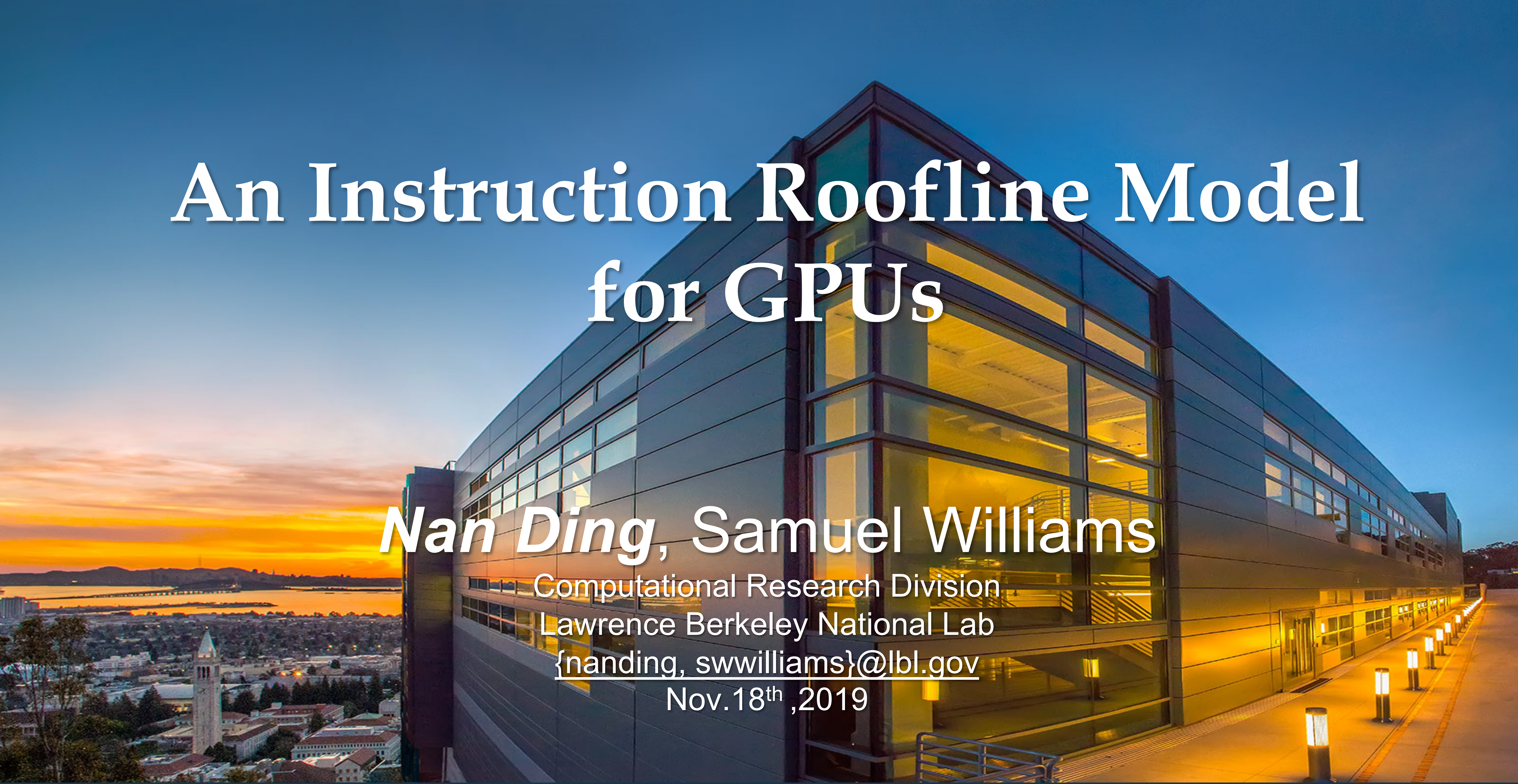


An Instruction Roofline Model for GPUs

Nan Ding, Samuel Williams

Computational Research Division
Lawrence Berkeley National Lab
{nanding, swwilliams}@lbl.gov

Nov.18th ,2019

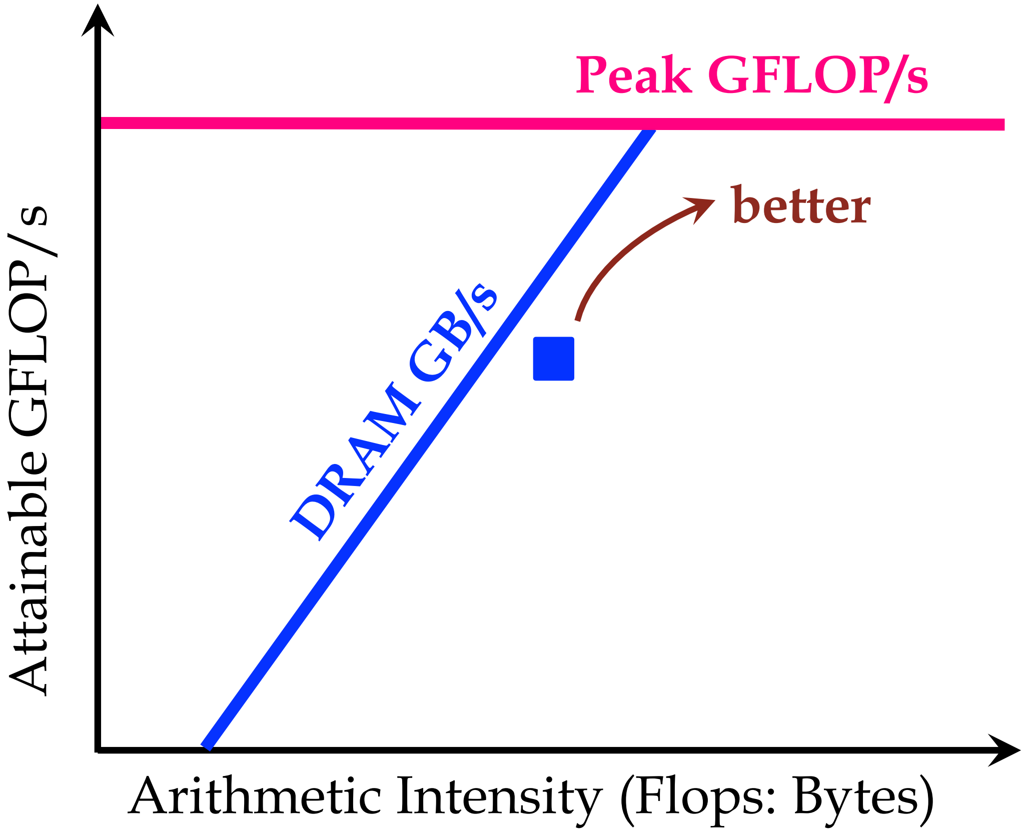
History of Roofline Models

- Sustainable performance is bound by

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

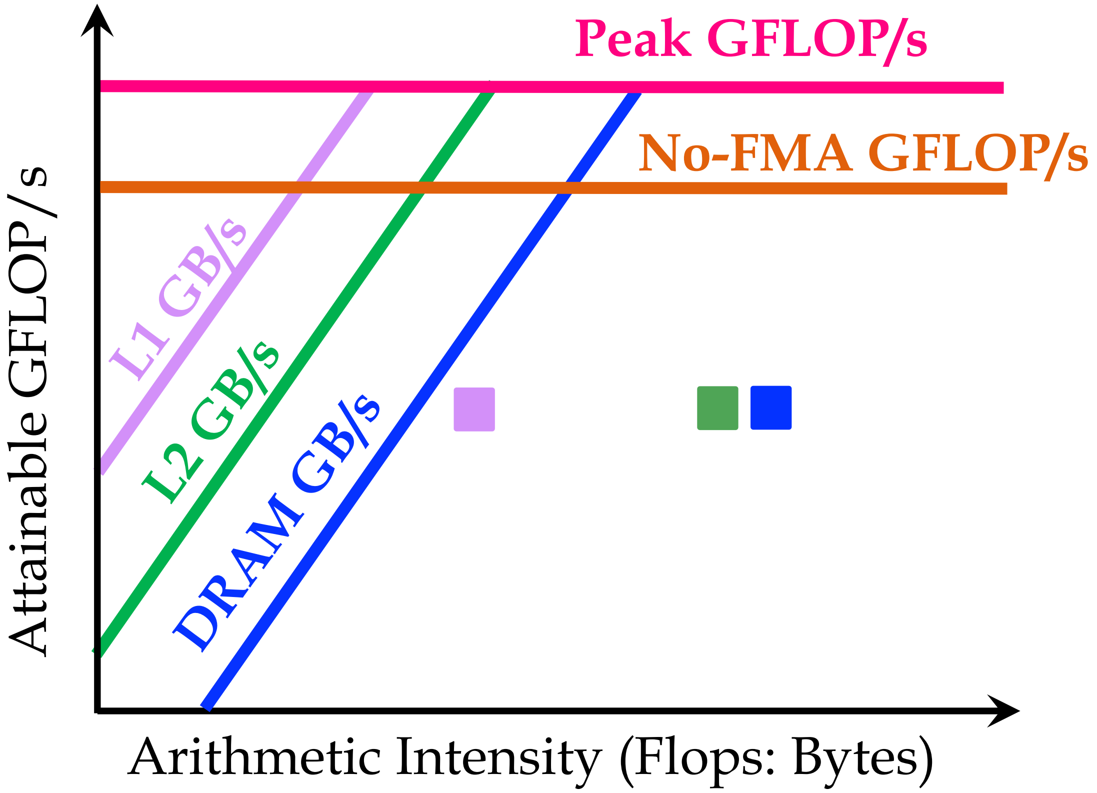
Roofline Model

- Arithmetic Intensity (AI) : FLOPs/Byte



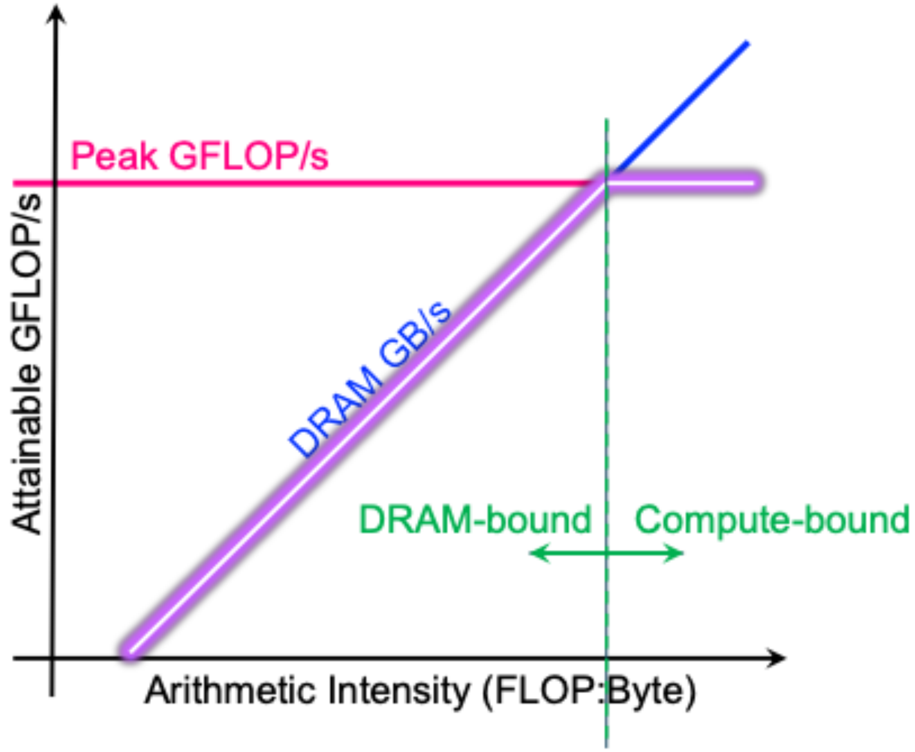
Hierarchical Roofline Model

- Arithmetic Intensity (AI) : FLOPs/Byte(L1/L2/DRAM)
- Additional compute ceilings: No-FMA peak

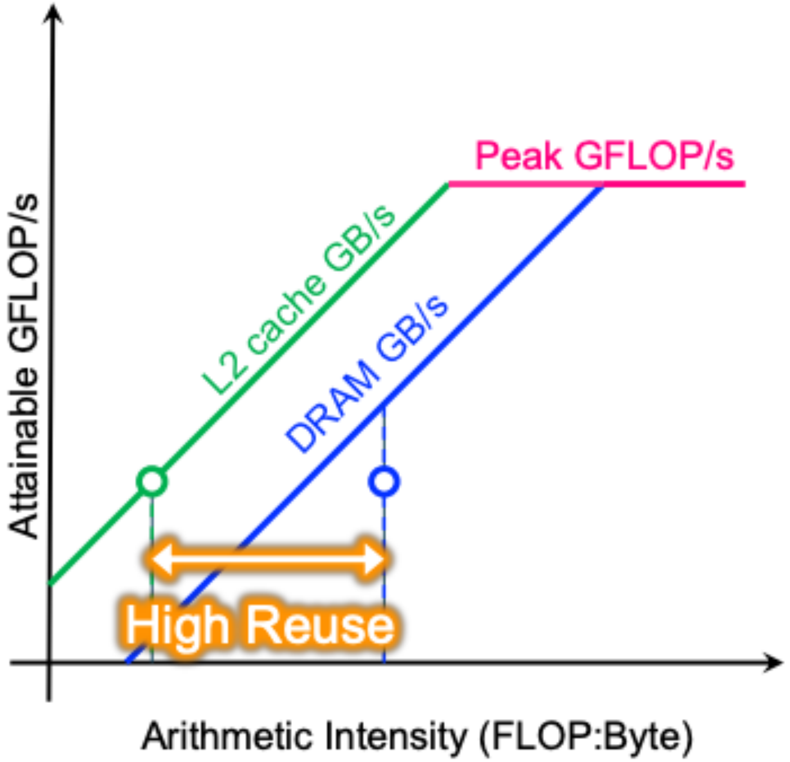


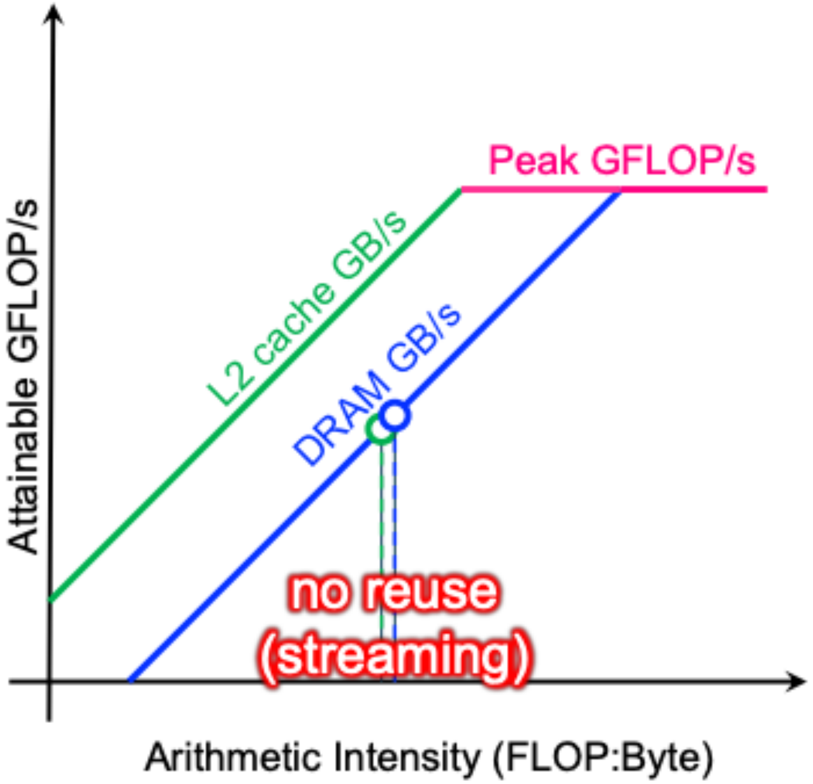
Roofline is Useful

Memory-bound or Compute-bound



Cache effects



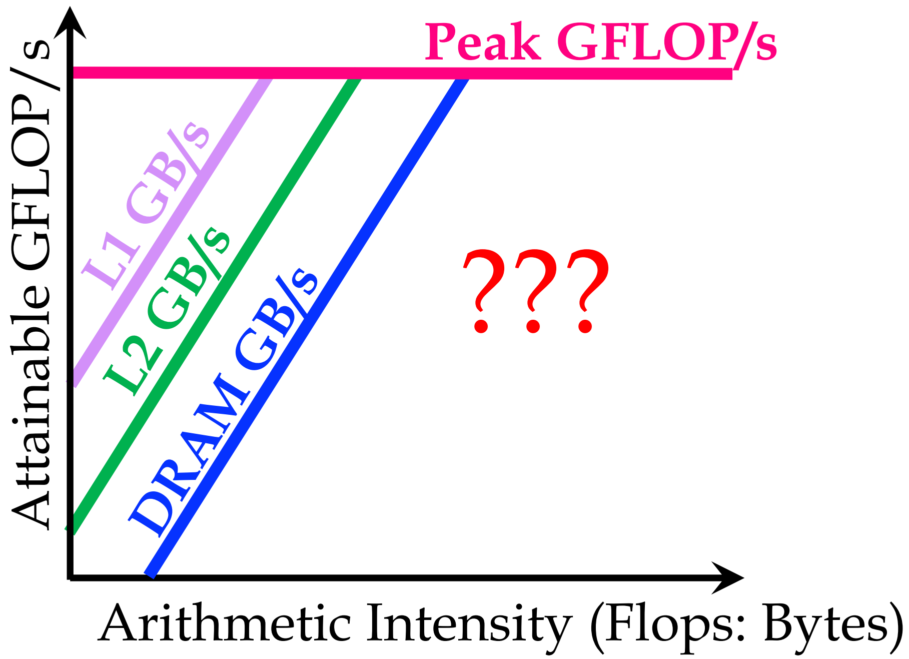


Driving performance optimization

However…

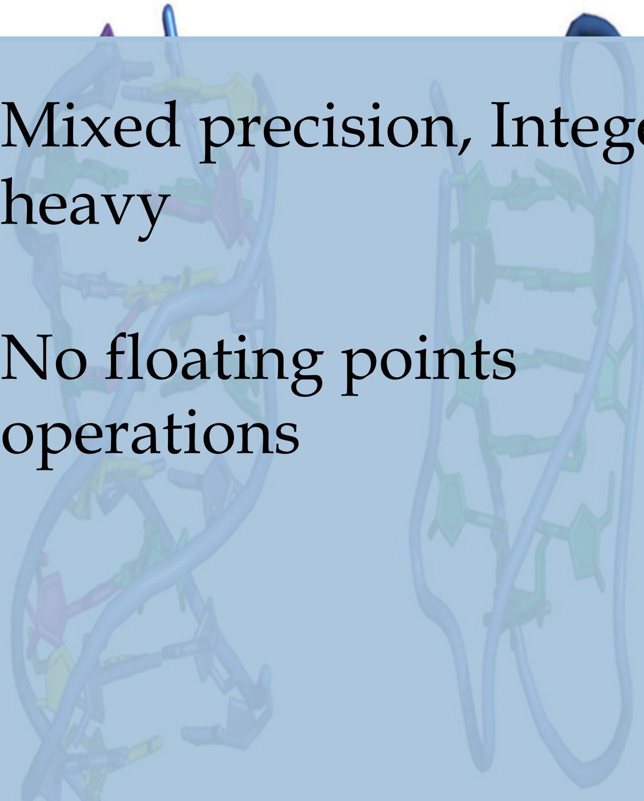
- Even with sufficient data locality, one cannot guarantee high performance
  - Pathological memory access patterns?
  - Re-design the data layout?
  - Limited by instruction throughput?

- Many applications perform more integer operations than floating-point/no-flops



Motivation for the Instruction Roofline Model

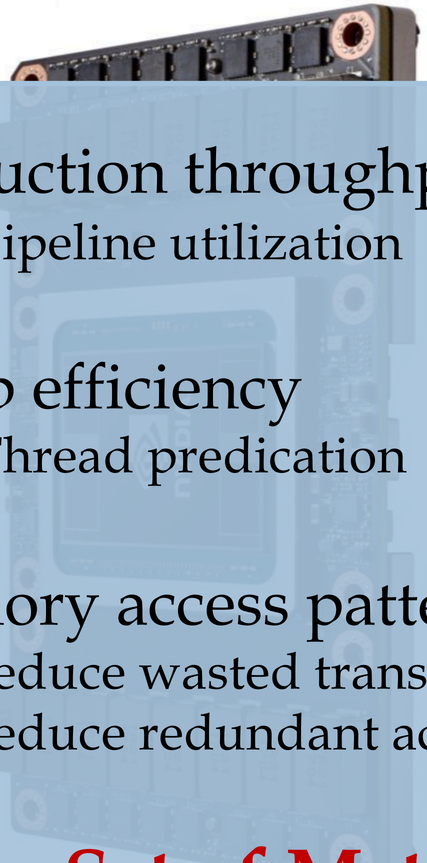
Emerging Domains



- Mixed precision, Integer-heavy
- No floating points operations

More than Flops

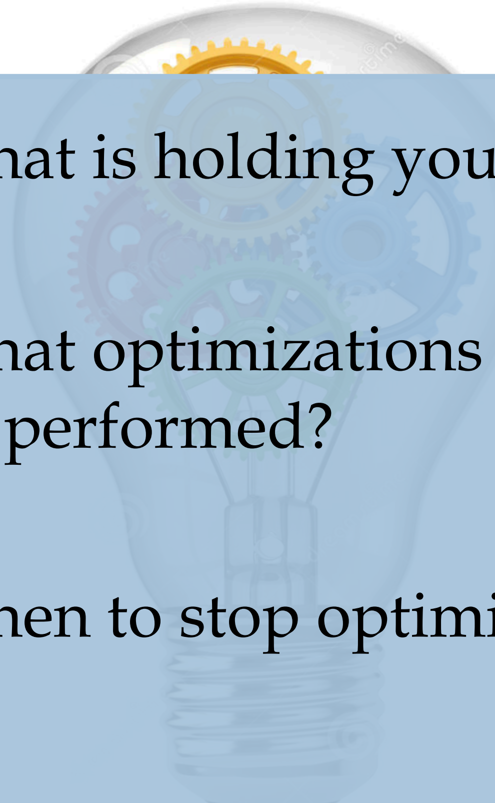
Architectural Evolution



- Instruction throughput
  - pipeline utilization
- Warp efficiency
  - Thread predication
- Memory access patterns
  - reduce wasted transactions
  - reduce redundant access

A New Set of Metrics

Practical Use



- What is holding you back?
- What optimizations should be performed?
- When to stop optimization?

Drive Code Optimization in a Good Visual Manner

The First Step to Instruction Roofline Model

- Sustainable performance is bound by

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

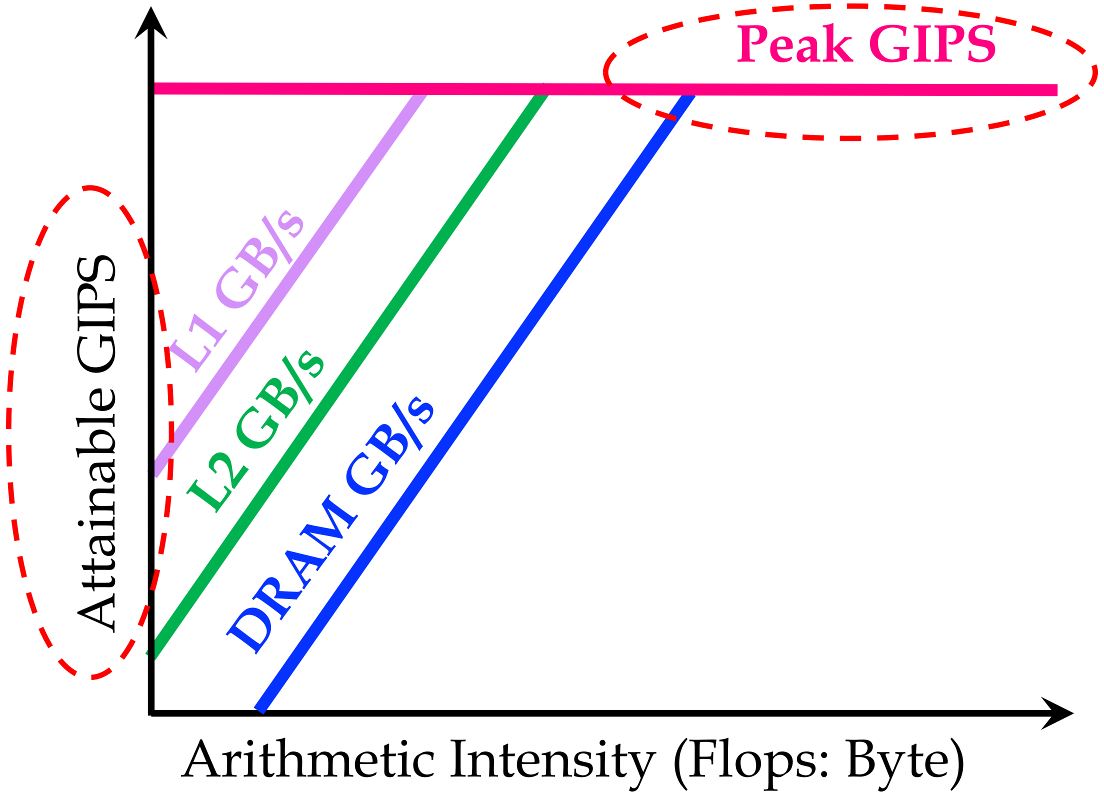
⬇

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

Expanding the applicability of roofline to several emerging computational domains

Form the basis for several subsequent Instruction Roofline-oriented performance analysis technologies

- **Identify fetch-decode-issue bottlenecks**
- **Function unit utilization (FPU, tensor, integer, etc...)**



The Second Step to Instruction Roofline Model

- Sustainable performance is bound by

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

$$\Downarrow$$

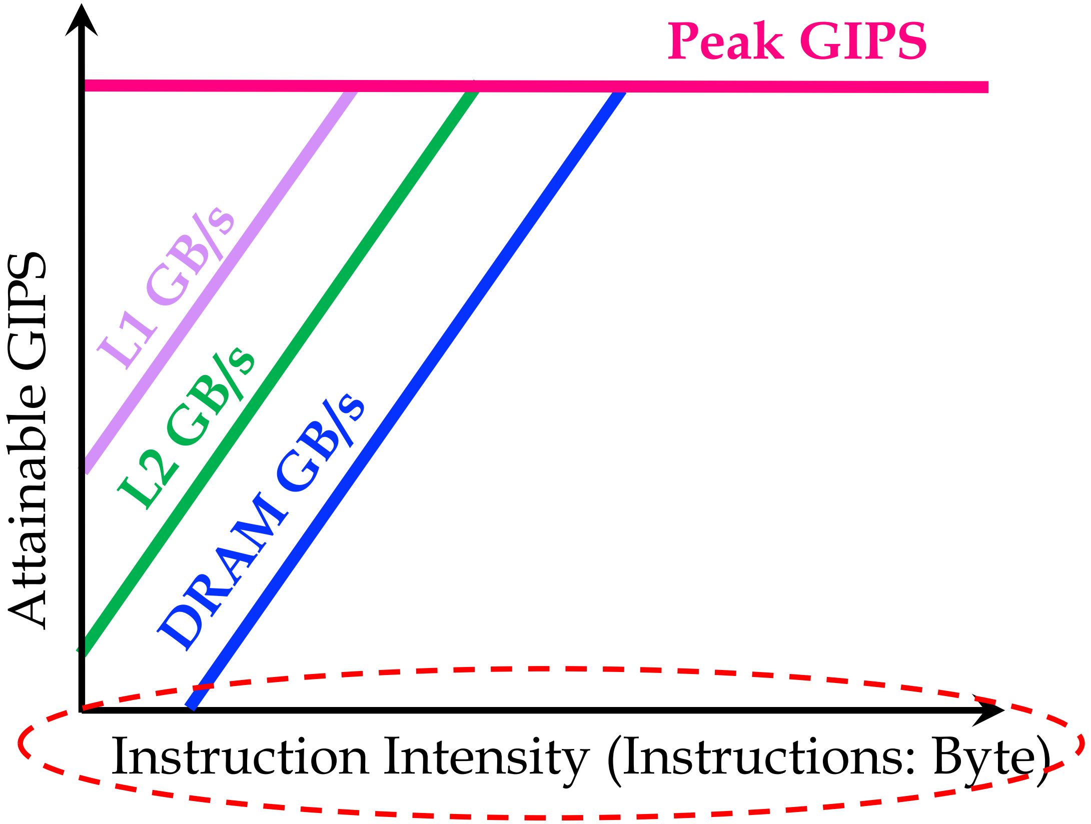
$$\text{GIPS} = \min \begin{cases} \text{Peak GIPS} \\ \text{II} * \text{GB/s} \end{cases}$$

- **Instruction Intensity**: Instructions per Byte

Expanding the applicability of roofline to several emerging computational domains

Limitation:

Hard to motivate more performance analysis techniques, such as memory pattern access



A Final Step to Instruction Roofline Model on GPUs

- **Instruction Intensity**
  - Instructions per Transaction

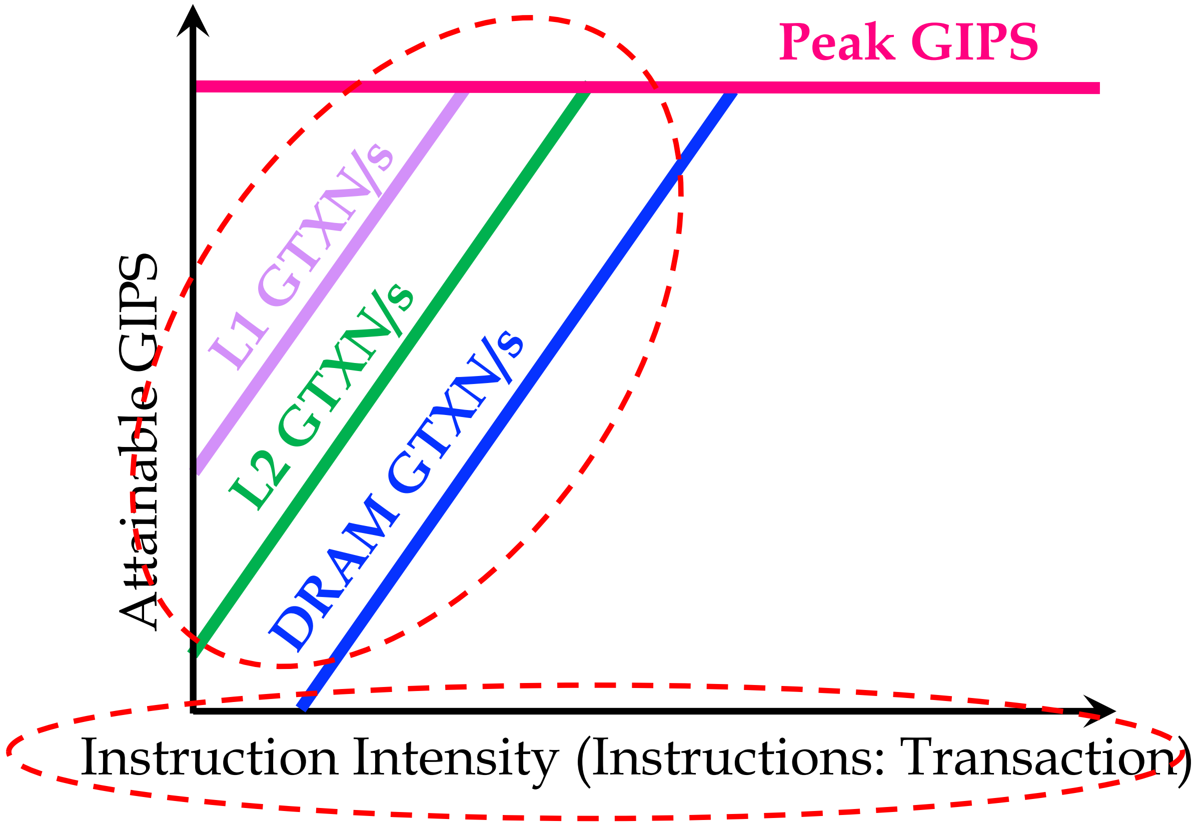
Expanding the applicability of roofline to more performance analysis technologies GPUs

Form the basis for several subsequent Instruction Roofline-oriented performance analysis technologies on GPUs:

- **Memory access patterns**

- **Memory Transaction**
  - the natural unit to access data on NVIDIA GPUs
  - the natural unit to **analyze memory access**
  - a warp-level load/store -> 1 - 32 transactions

$$\text{GIPS} = \min \begin{cases} \text{Peak GIPS} \\ \text{Instructions/Transaction} * \text{GTXN/s} \end{cases}$$



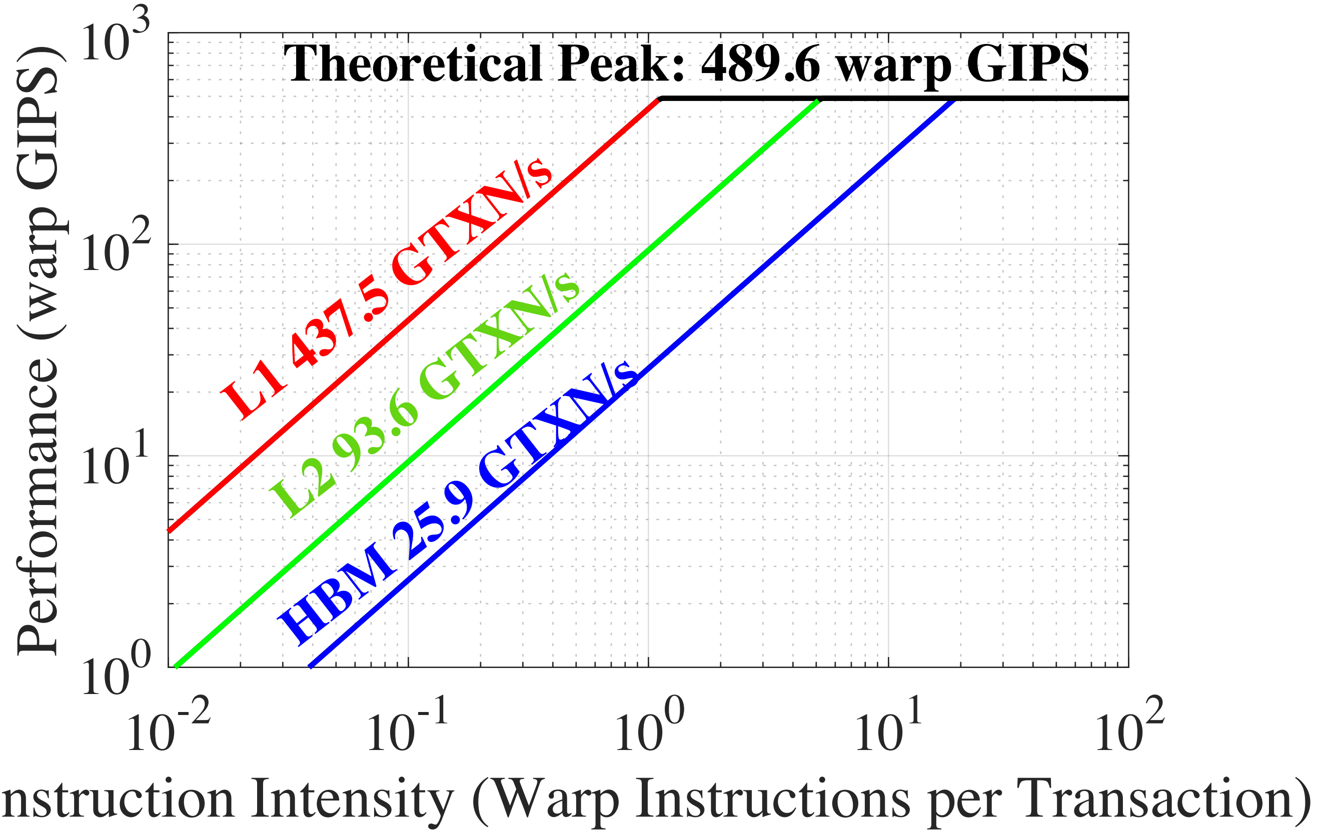
Instruction Roofline Performance Model

- **Sustainable performance of is bound by**

$$\text{GIPS} = \min \begin{cases} \text{Peak GIPS} \\ \text{Instruction Intensity} * \text{GTransaction/s} \end{cases}$$

- **Theoretical Peak on V100:**
  80 SMs x 4 warp scheduler x 1 inst/cyc x 1.53GHz = 489.6 GIPS

- **Memory ceilings on V100:**
  - Based on the GB/s from Empirical Roofline Toolkit[1]
  - Calculate the number of equivalent 32-byte transactions



[1] https://bitbucket.org/berkeleylab/cs-roofline-toolkit

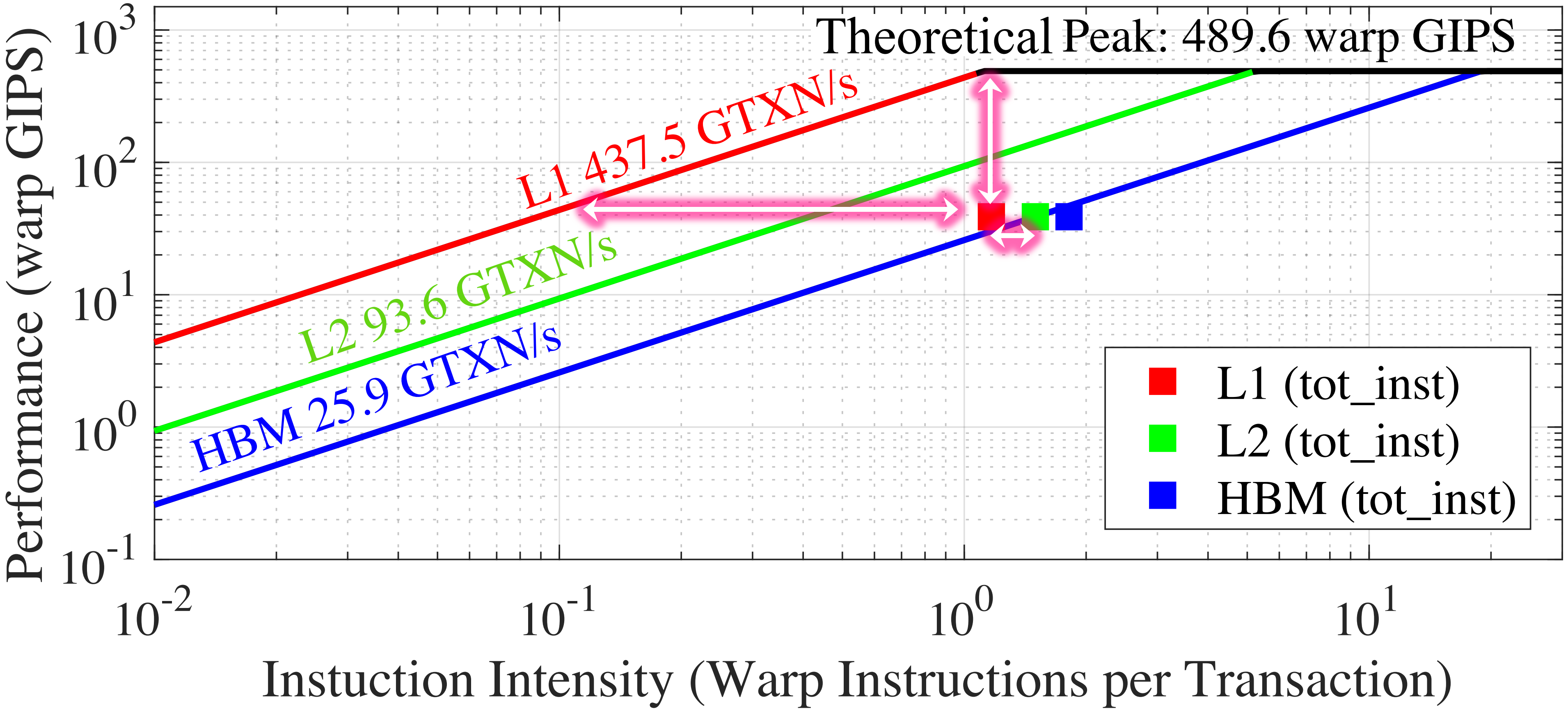
Capabilities of Instruction Roofline Performance Model (1/2)

Instruction Throughput

Capabilities of Instruction Roofline Model -- Instruction Throughput

- **Instruction throughput**

  All instruction, Transactions of each memory level(L1/L2/HBM), runtime

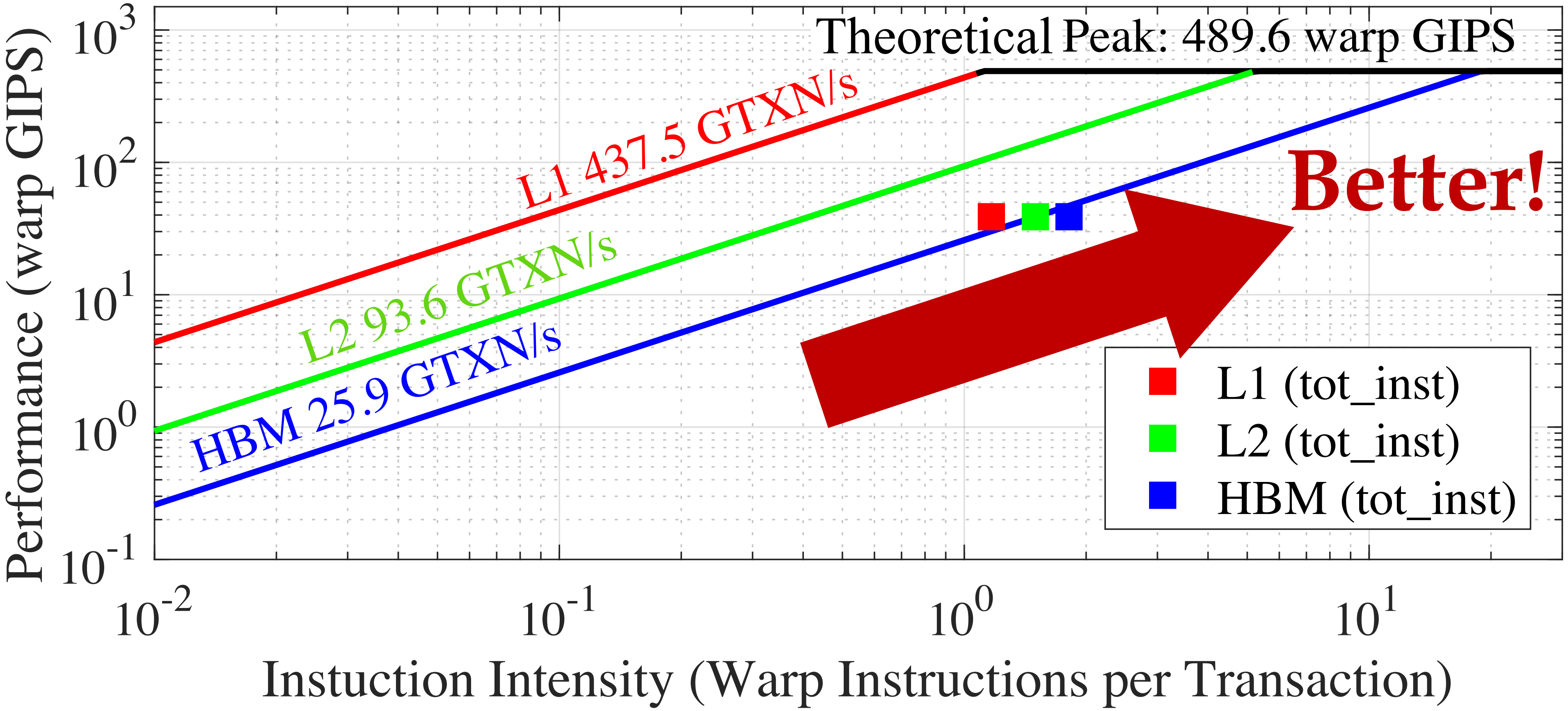


- **Insights:**
  1. Distance between the ceilings and dots can tell memory-bound or instruction-bound
  2. Distance between the two plots (different memory level) can tell the data reuse.

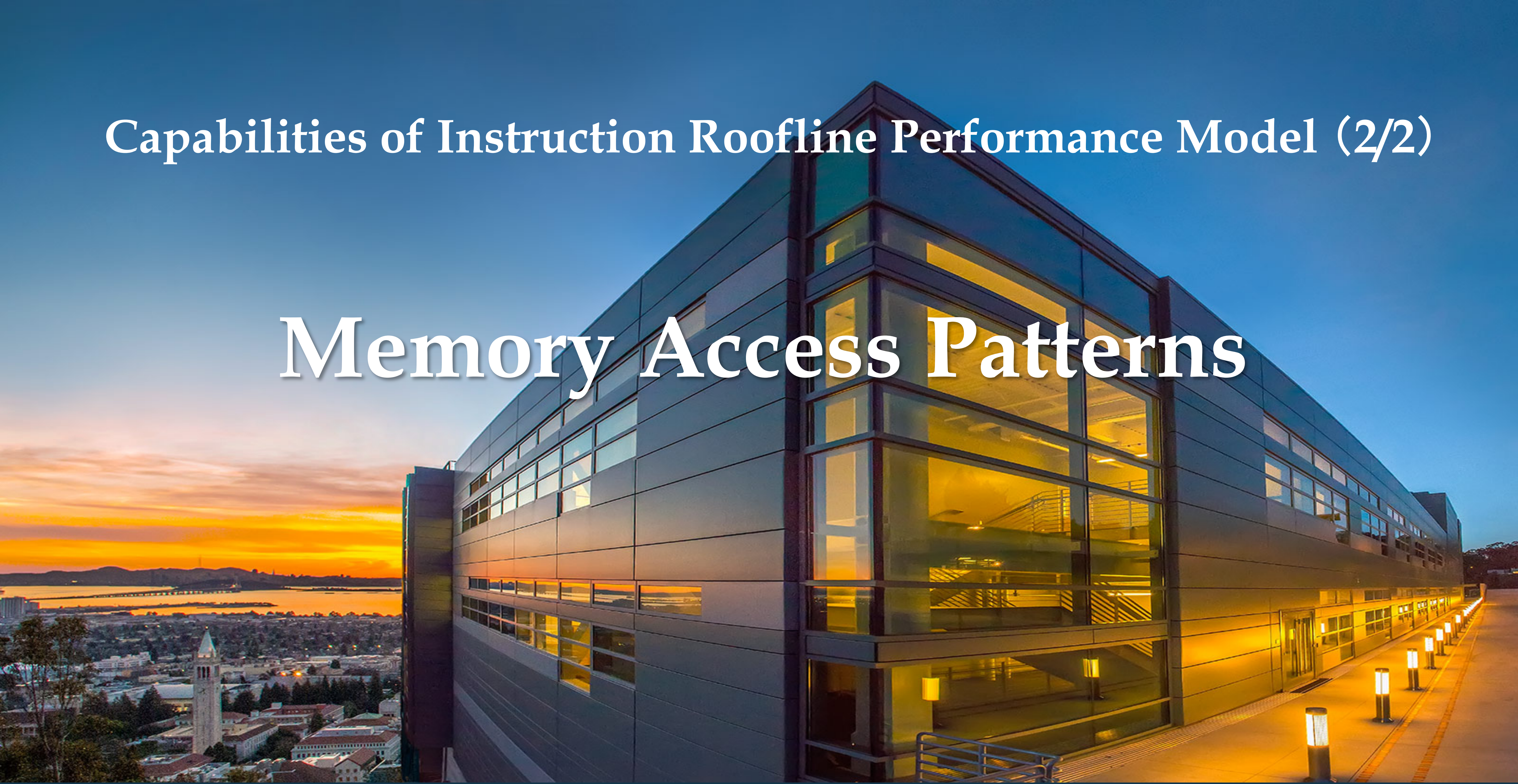
Capabilities of Instruction Roofline Model -- Instruction Throughput

- **Instruction throughput**

  All instruction, Transactions of each memory level(L1/L2/HBM), runtime



- **Insights:**
  1. Distance between the ceilings and dots can tell memory-bound or instruction-bound
  2. Distance between the two plots (different memory level) can tell the data reuse.



Capabilities of Instruction Roofline Performance Model (2/2)

Memory Access Patterns

Memory Access Pattern is Critical to Application Execution Time

Easy to code in an inefficient memory pattern

Low performance

Hidden deep in the code

Time consuming to reason the performance

Capabilities of Instruction Roofline Model -- Global Memory Patterns

1 warp-level load/store -> 1 to 32 transactions depending on memory patterns

"Stride-0"

1 warp Global LDST

1 Global Transaction

"Stride-8"

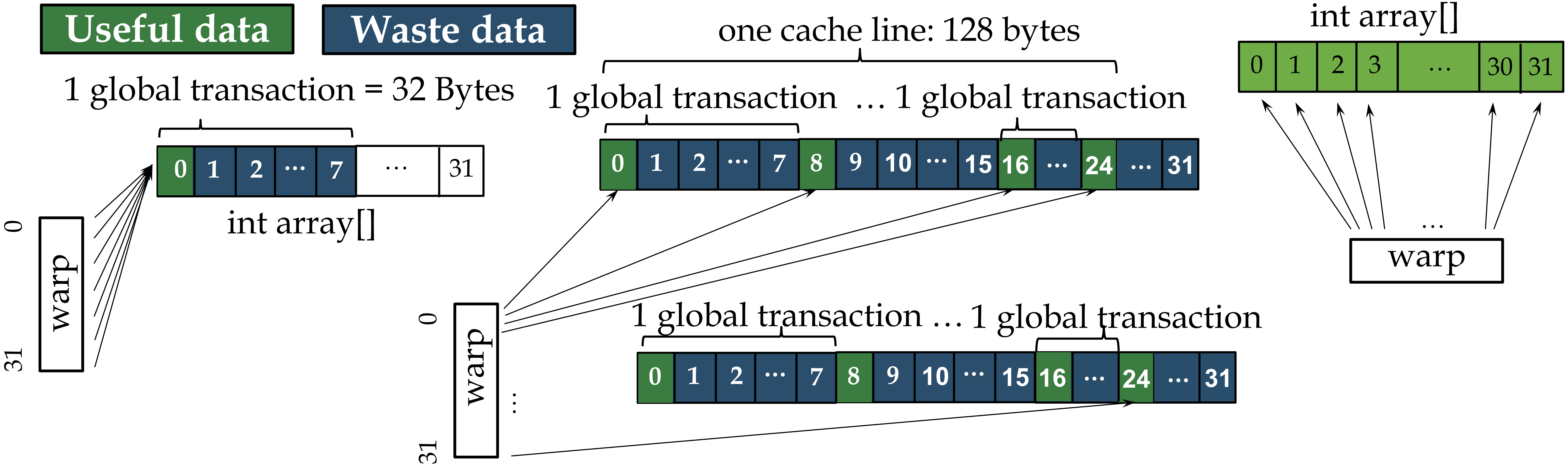
1 warp Global LDST

32 Global Transactions

"Stride-1" (Unit Stride)

1 warp Global LDST

4 Global Transactions



Useful data

Waste data

1 global transaction = 32 Bytes

int array[]

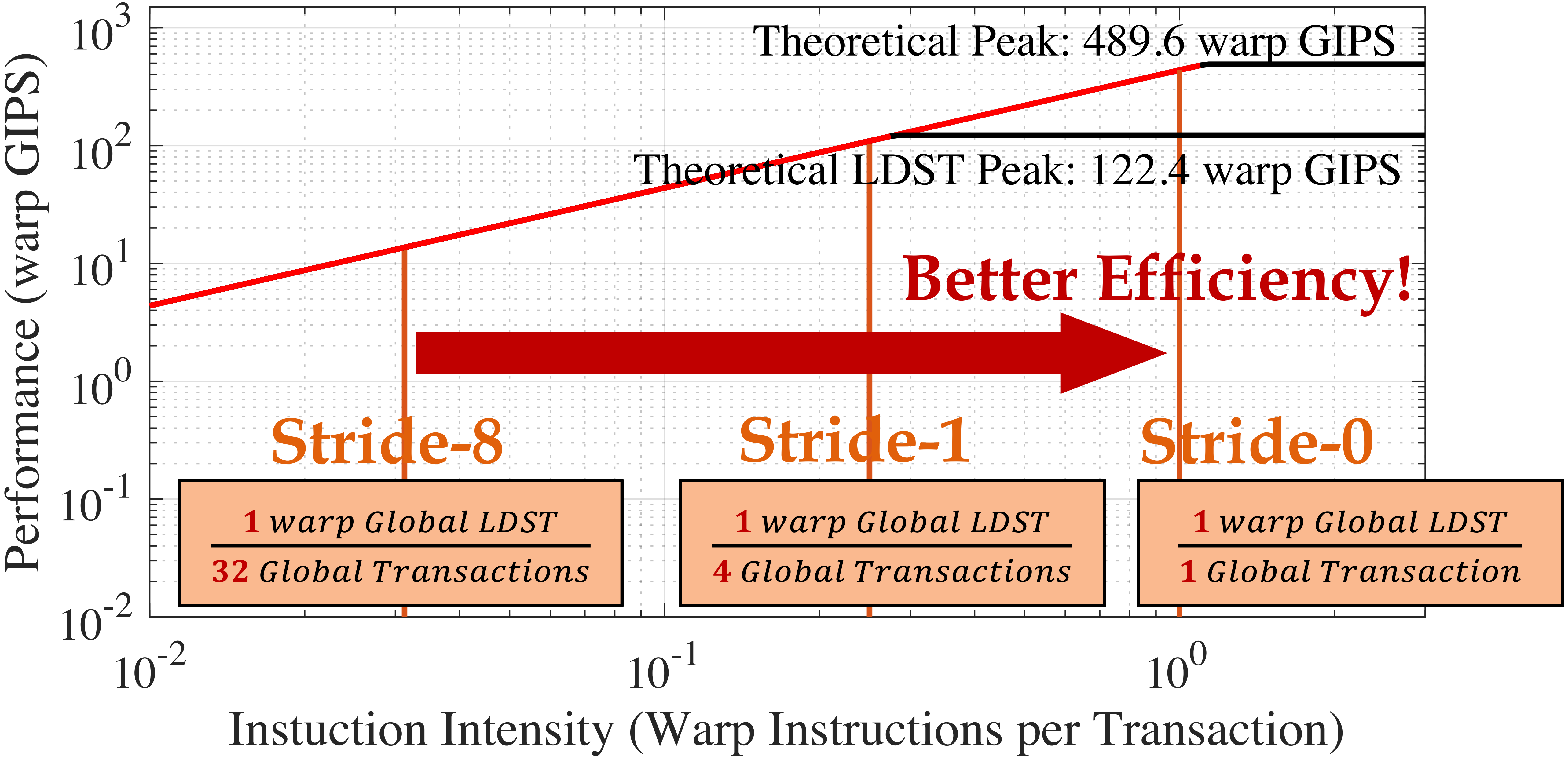
one cache line: 128 bytes

1 global transaction … 1 global transaction

1 global transaction … 1 global transaction

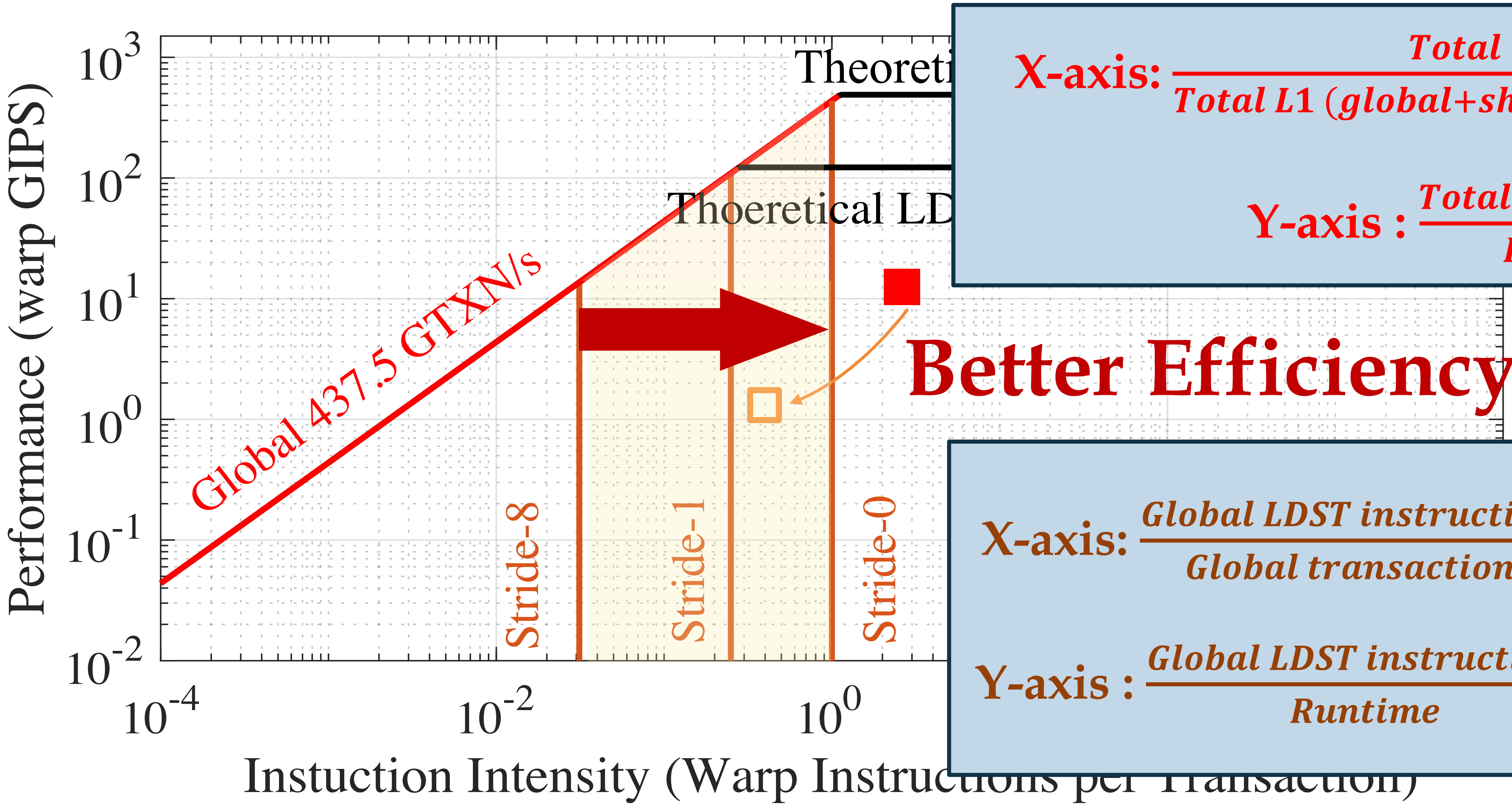
int array[]

Capabilities of Instruction Roofline Performance Model
---- Three Intensity ``Walls'' for Stride Global Memory Access Patterns



Capabilities of Instruction Roofline Performance Model
---- Characterize Global Memory Access Patterns

Breakdown the **L1 dot** into **Global Memory Only metrics -> Stride Global Memory Patterns** according to **Global Memory Walls**



Better Efficiency!

X-axis: $\dfrac{\textit{Total instructions}}{\textit{Total L1 (global+shared+ local) transactions}}$

Y-axis : $\dfrac{\textit{Total instructions}}{\textit{Runtime}}$

X-axis: $\dfrac{\textit{Global LDST instructions}}{\textit{Global transactions}}$

Y-axis : $\dfrac{\textit{Global LDST instructions}}{\textit{Runtime}}$

Capabilities of Instruction Roofline Performance Model
---- Shared Memory Access Patterns

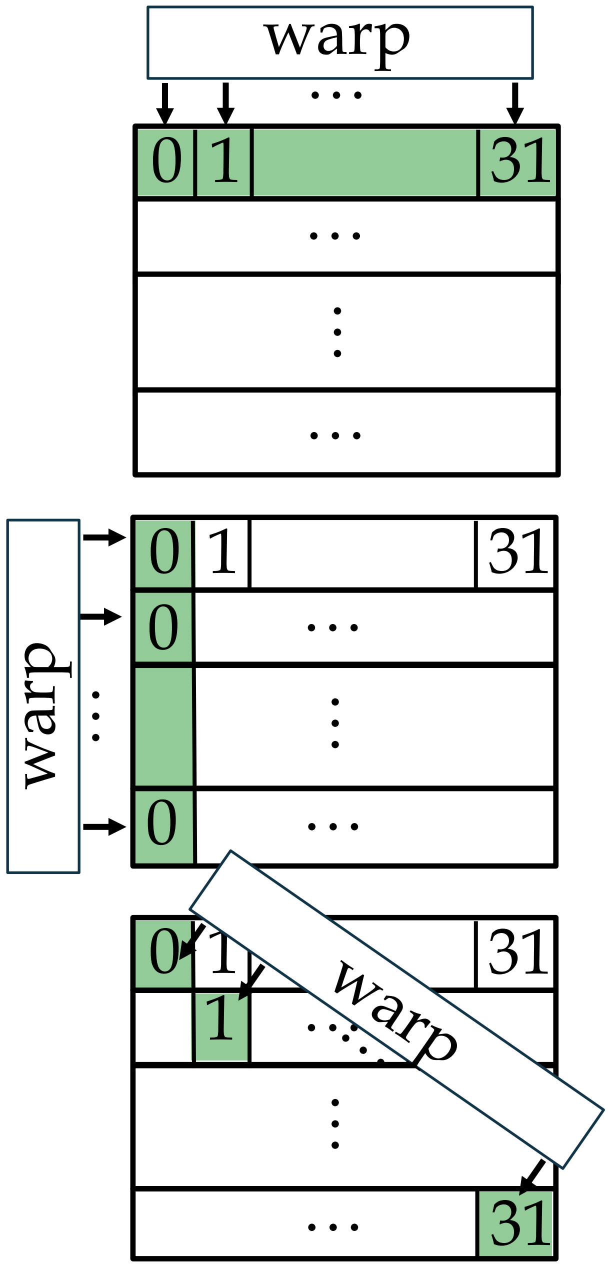
Notional

__shared__ int array[32][32]

__shared__ int array[32][32]

Reduce bank conflicts

__shared__ int array[32][32+1]



Physical

32 banks per shared memory row
Each bank is 4 Byte

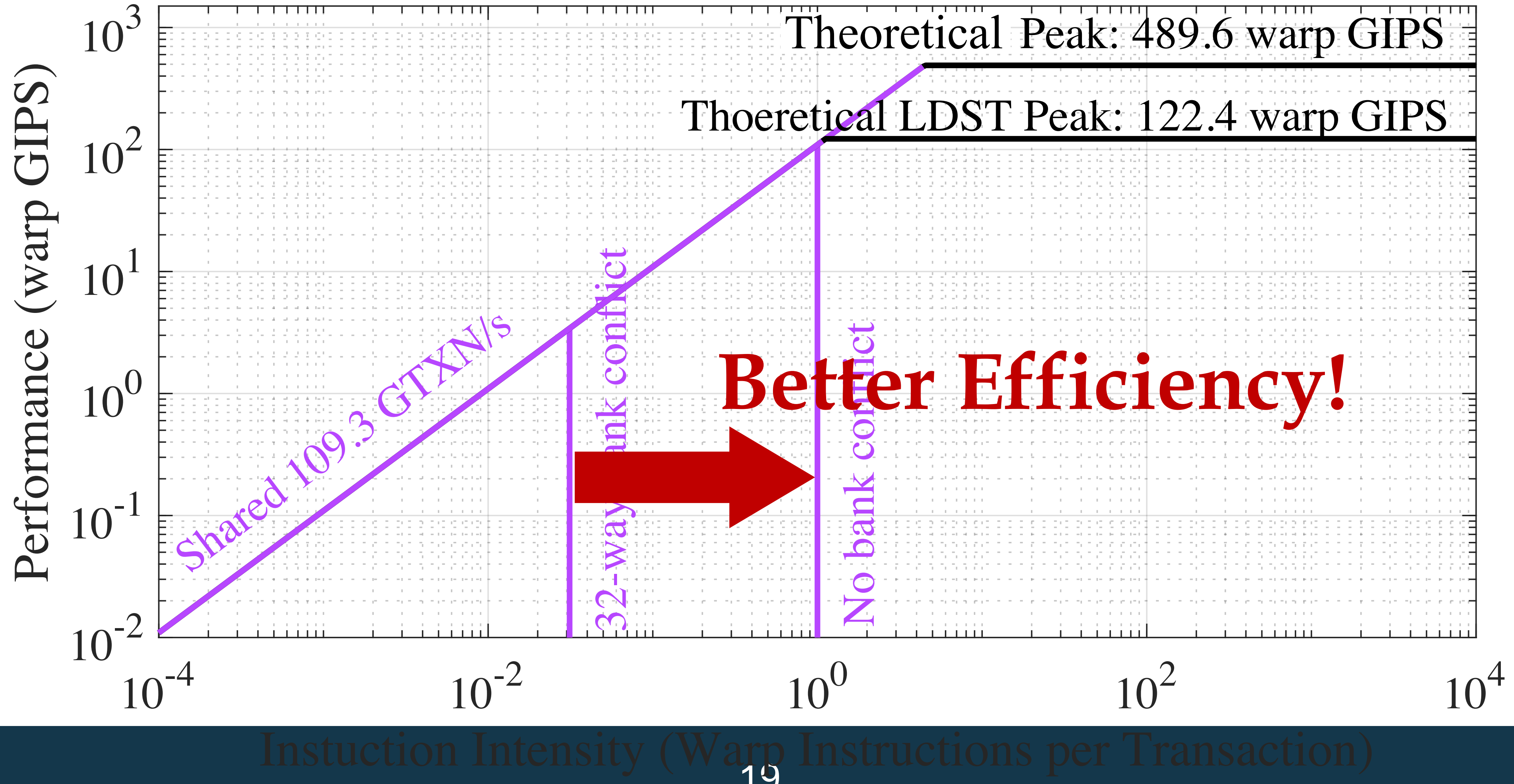
No bank conflict

32-way bank conflict

No bank conflict

Capabilities of Instruction Roofline Performance Model
---- Two Intensity ``Walls'' for Bank Shared Memory Access Patterns

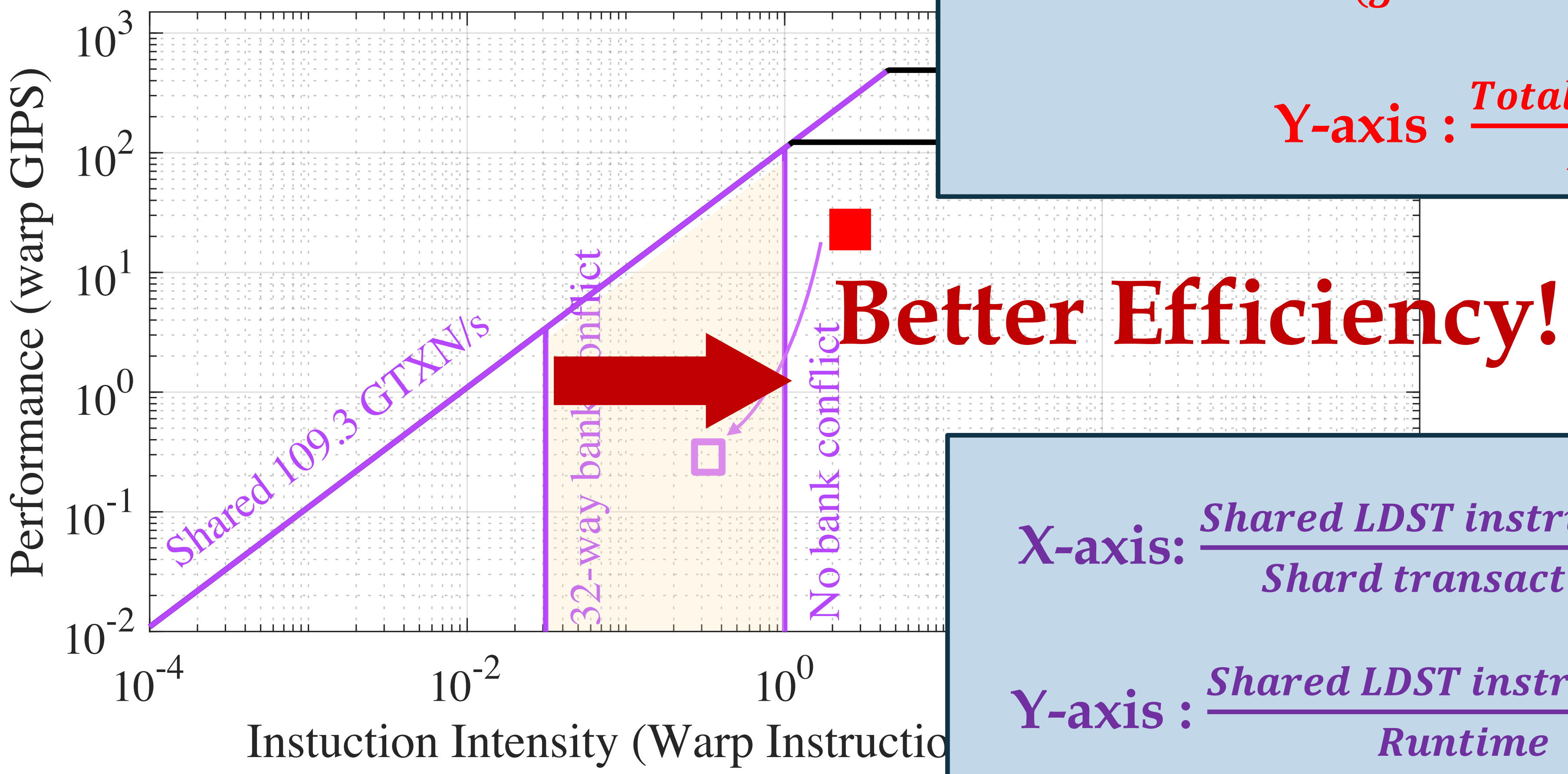
- “No bank conflict” = $\frac{1\, \textit{warp Shared LDST}}{1\, \textit{Shared Transaction}}$
  - different 4-byte word, different bank
  - same 4-byte word , same bank
- “32-way bank conflict” = $\frac{1\, \textit{warp Shared LDST}}{32\, \textit{Shared Transactions}}$
  - different 4-byte words, same bank



Better Efficiency!

Capabilities of Instruction Roofline Performance Model
---- Characterize Shared Memory Access Patterns

Breakdown the **L1 dot** into **Shared Memory Only metrics -> banked Shared Memory Patterns** according to **Shared Memory Walls**



Better Efficiency!

X-axis: $\dfrac{\textit{Total instructions}}{\textit{Total L1 (global + shared+ local) transactions}}$

Y-axis : $\dfrac{\textit{Total instructions}}{\textit{Runtime}}$

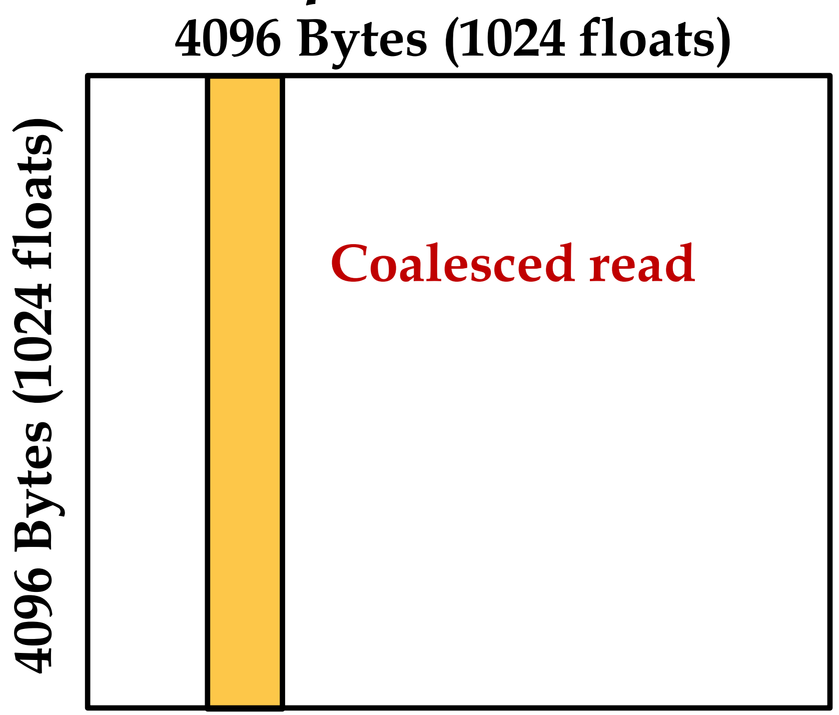
X-axis: $\dfrac{\textit{Shared LDST instructions}}{\textit{Shard transactions}}$

Y-axis : $\dfrac{\textit{Shared LDST instructions}}{\textit{Runtime}}$

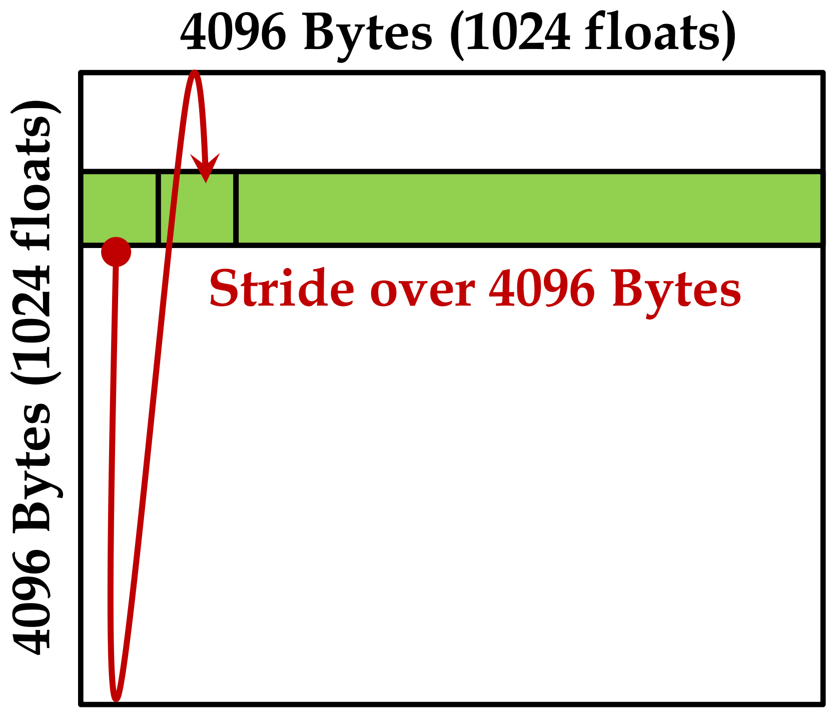
An example to understand the outputs from Instruction Roofline Model

Example: Matrix Transpose			
Description	A-> A^T, stored in column major		
Matrix size	1024 x 1024		
Machine	NVIDIA's latest V100 GPU		
Implementations	Naive	Coalesced	Coalesced_NoBankConflict
	Simple copy	Coaleced global memory access	Based on "Coalesced" Reduce shared memory bank conflicts
	Using 32×8 thread blocks operating on 32×32 matrix tiles		

Naive Implementation

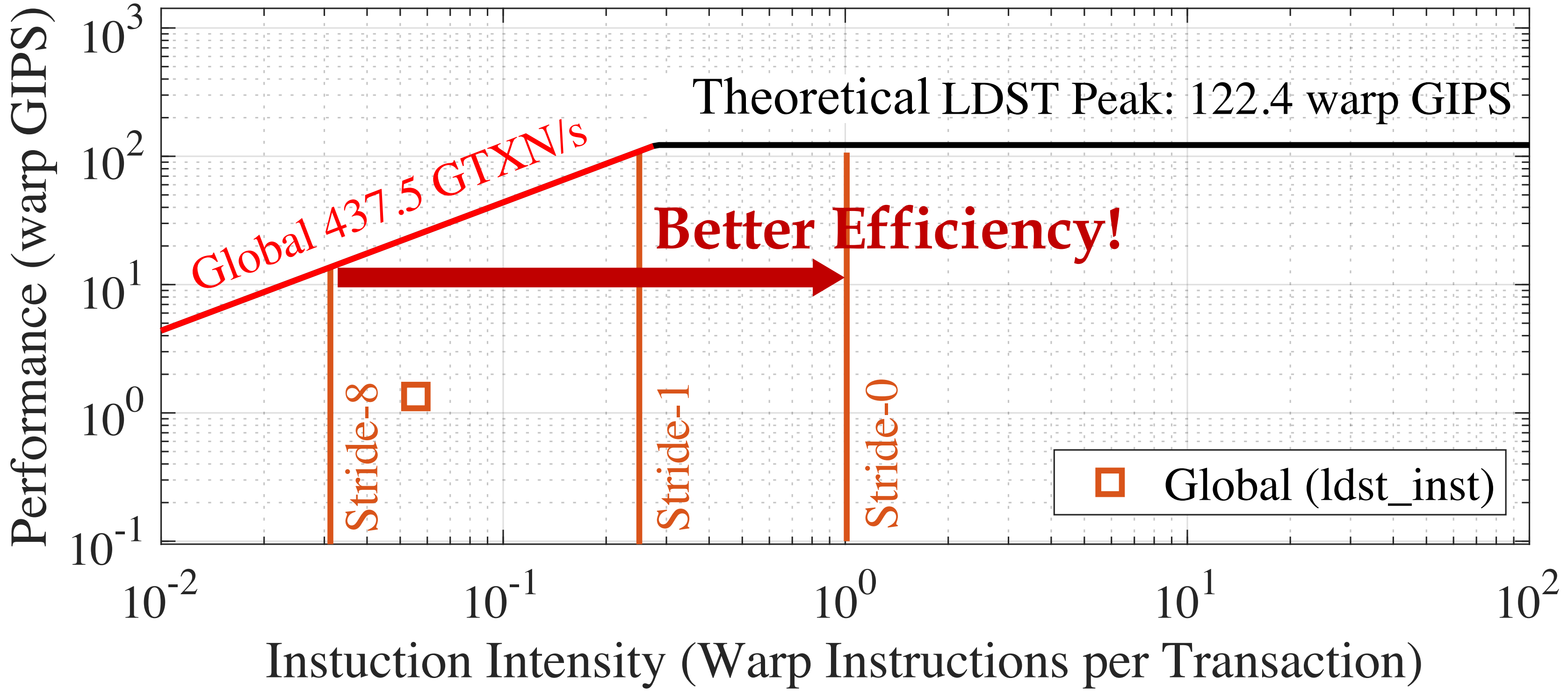


Input Matrix: A (column major)

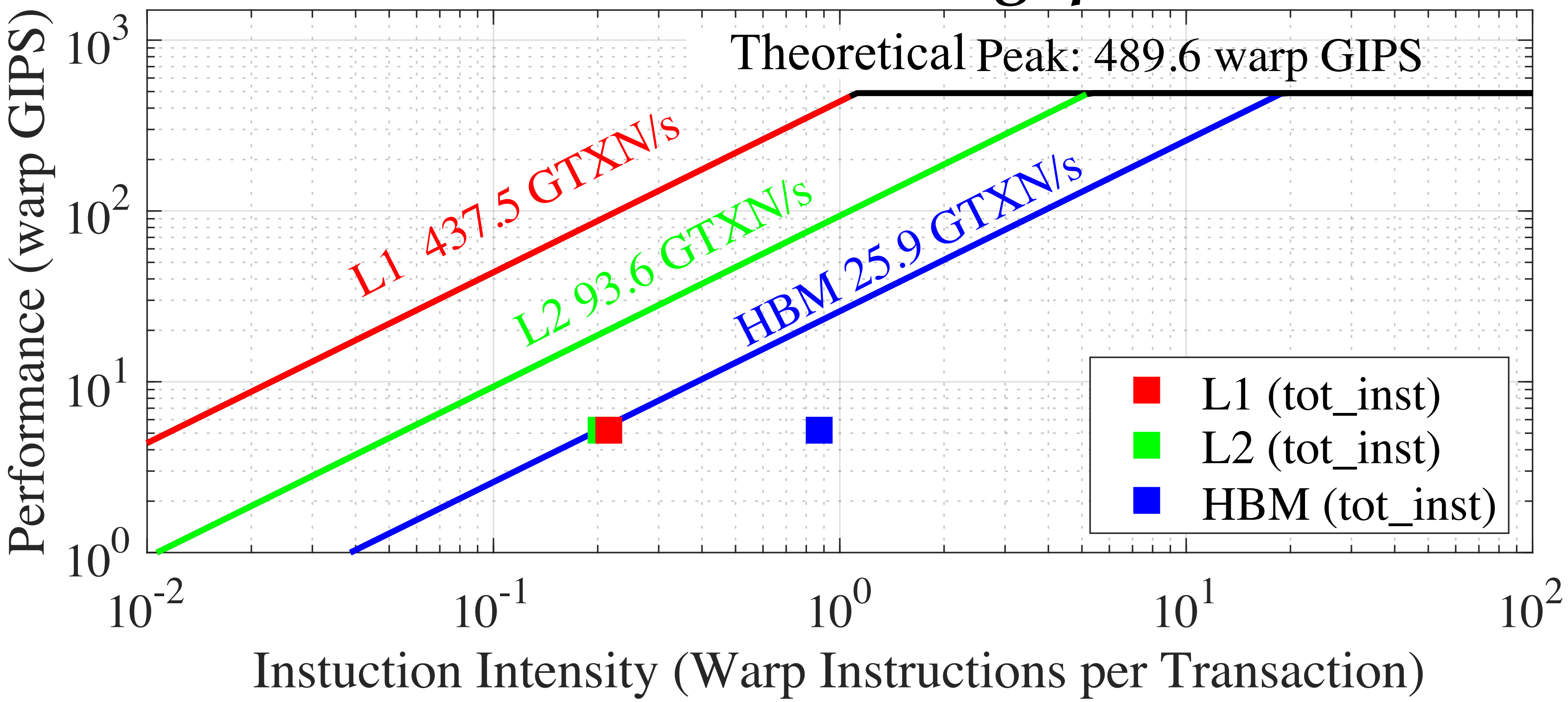


Output Matrix: A^T (column major)

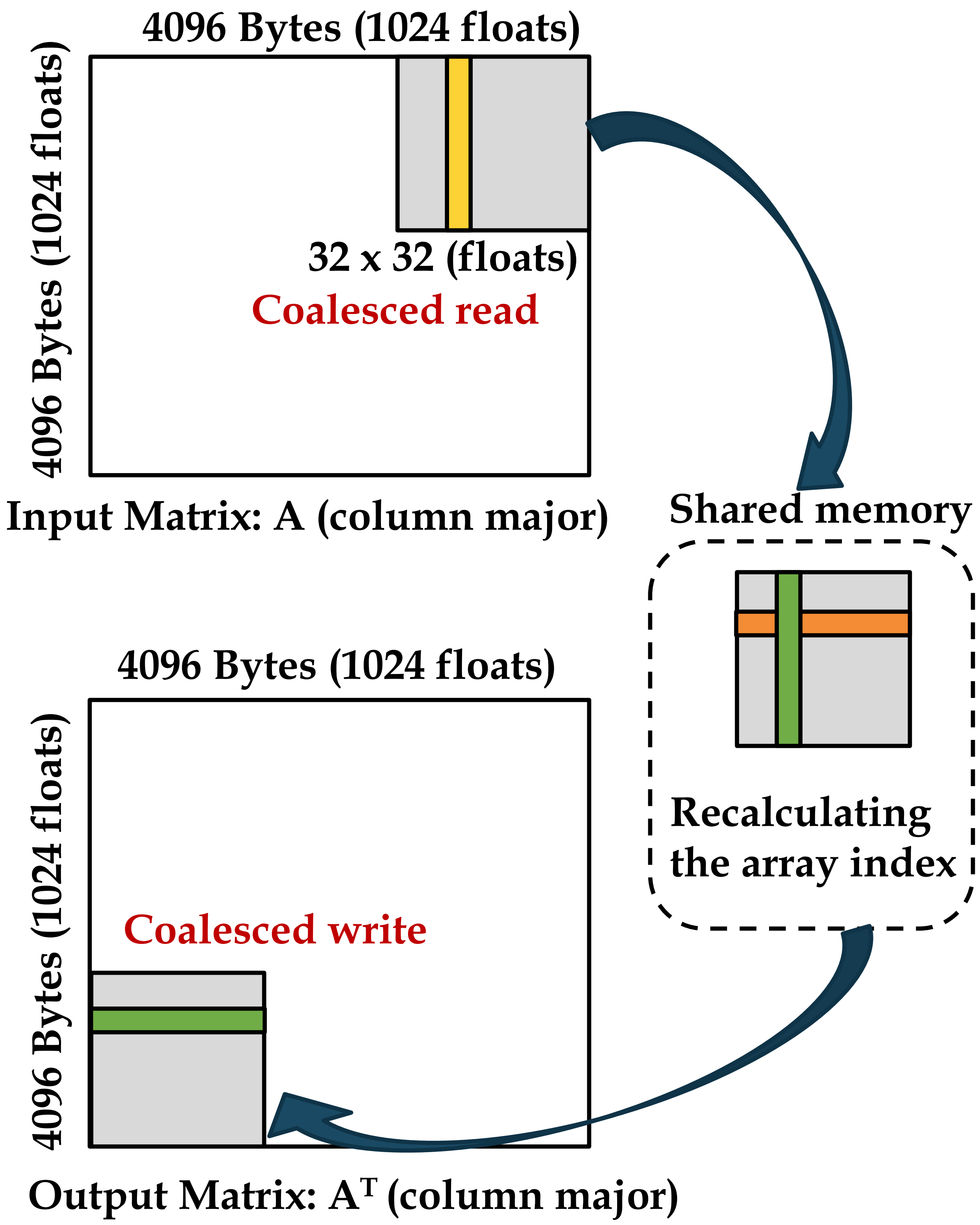
Global memory stride access



Instruction Throughput



Coalesced Implementation



4096 Bytes (1024 floats)

4096 Bytes (1024 floats)

32 x 32 (floats)

Coalesced read

Input Matrix: A (column major)

Shared memory

Recalculating the array index

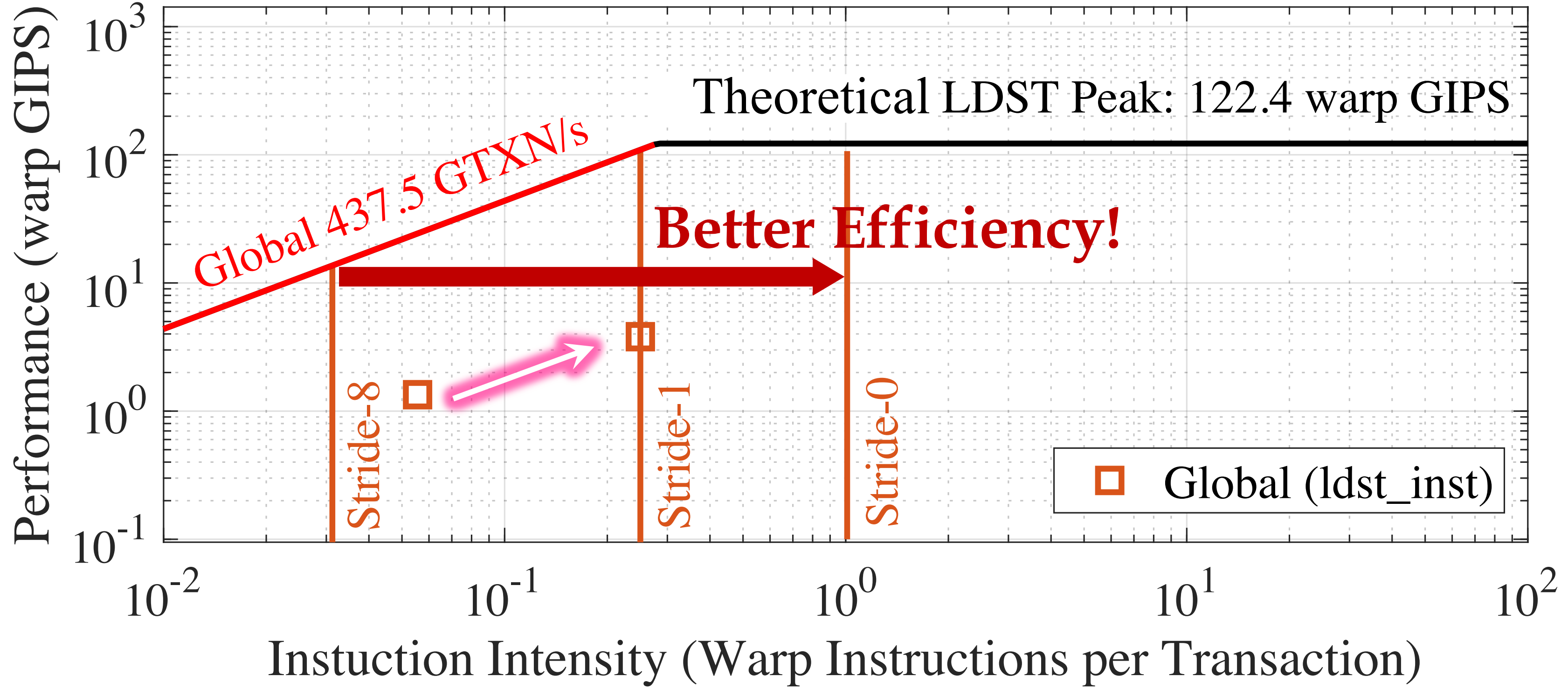
4096 Bytes (1024 floats)

4096 Bytes (1024 floats)

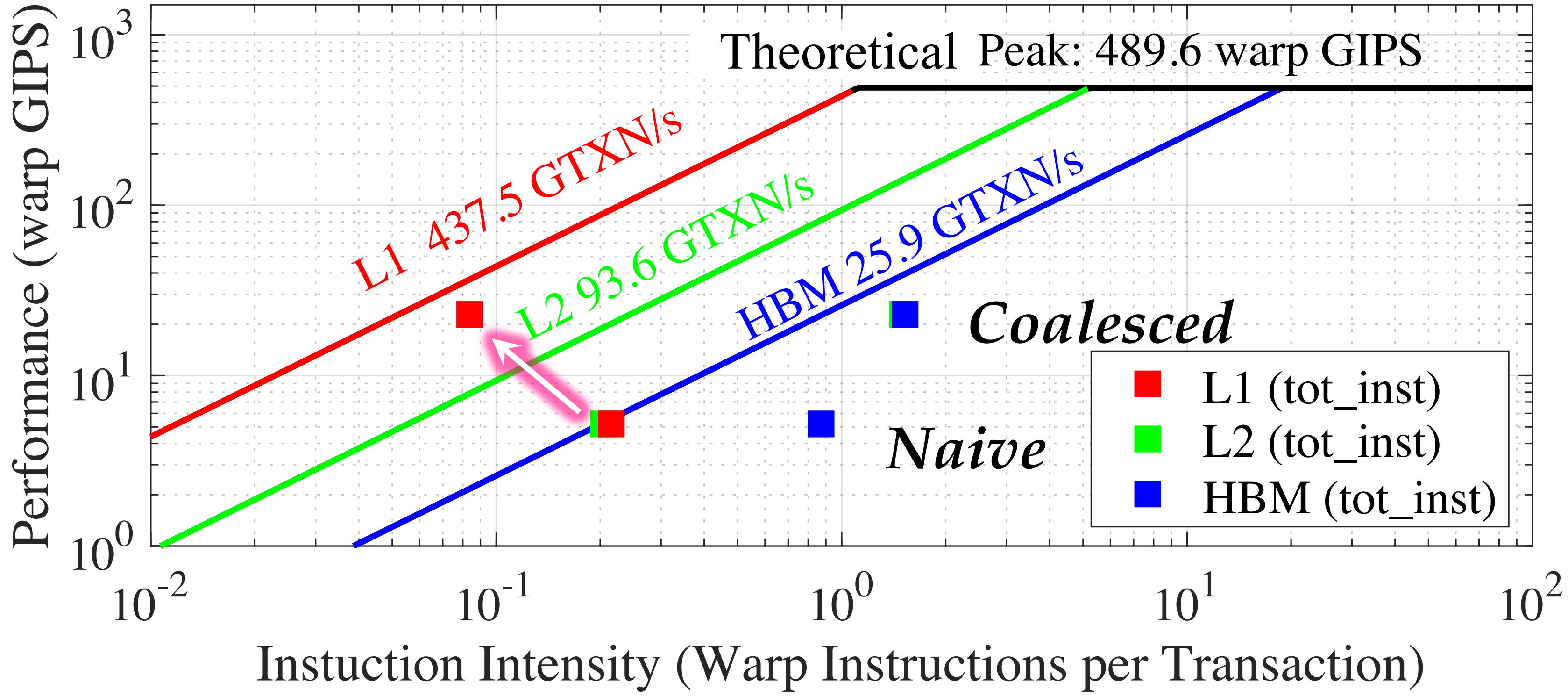
Coalesced write

Output Matrix: A^T (column major)

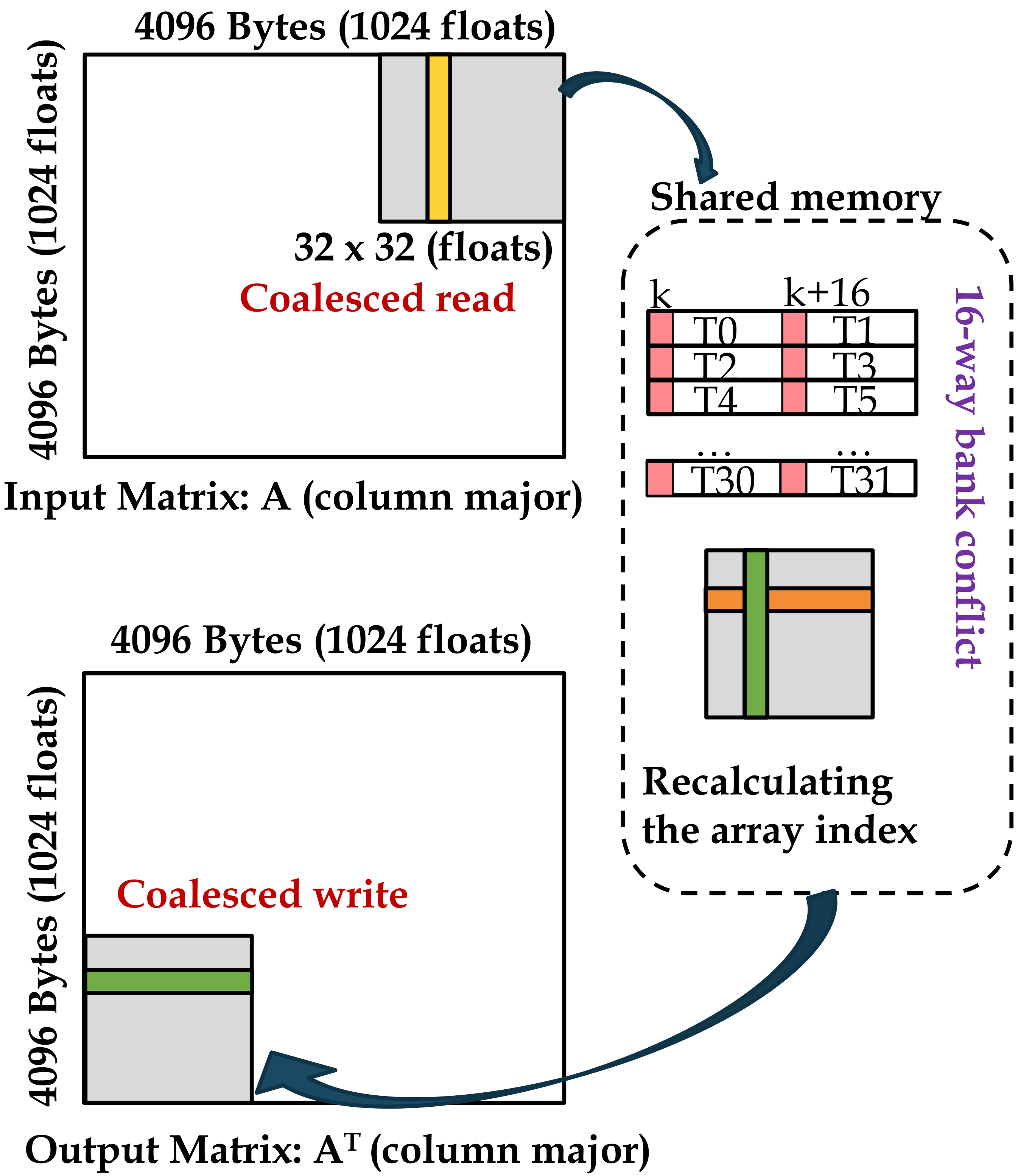
Global memory stride access



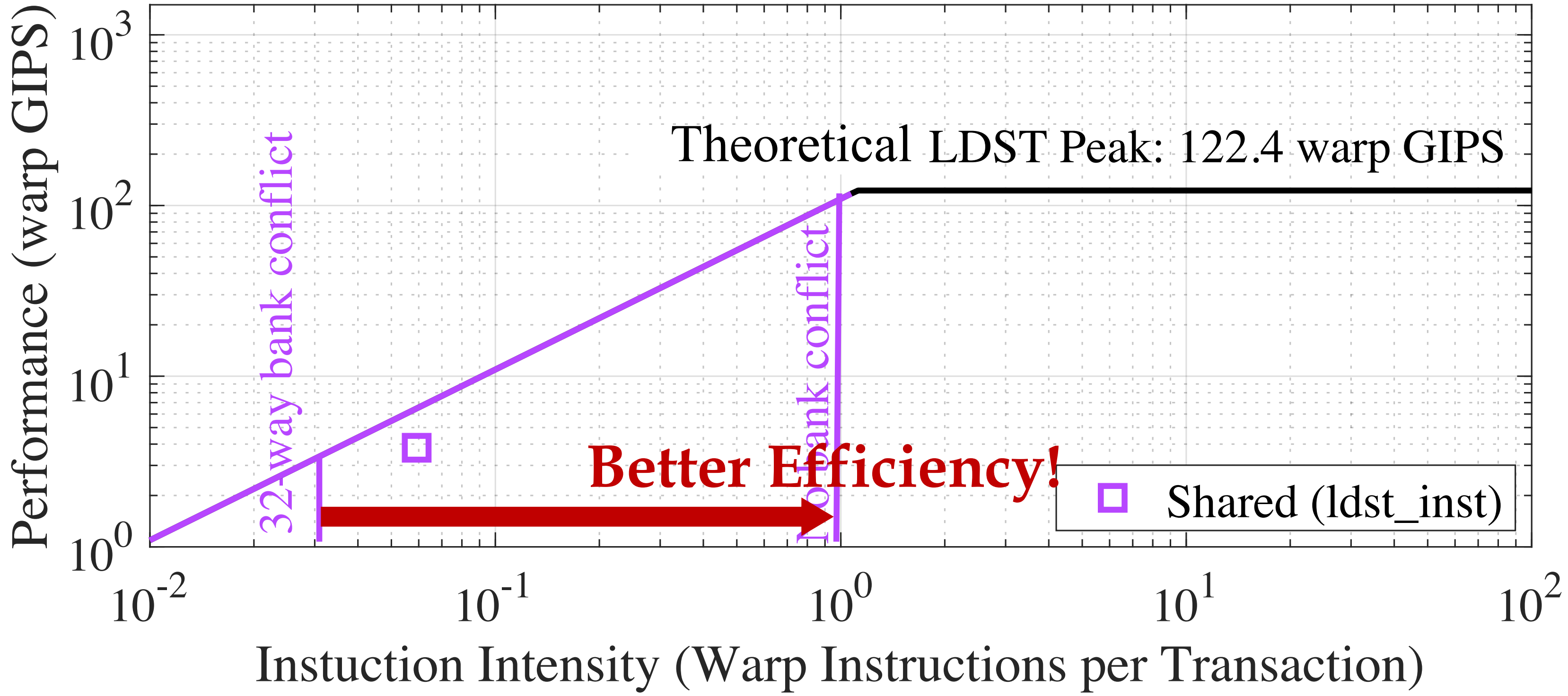
Instruction Throughput



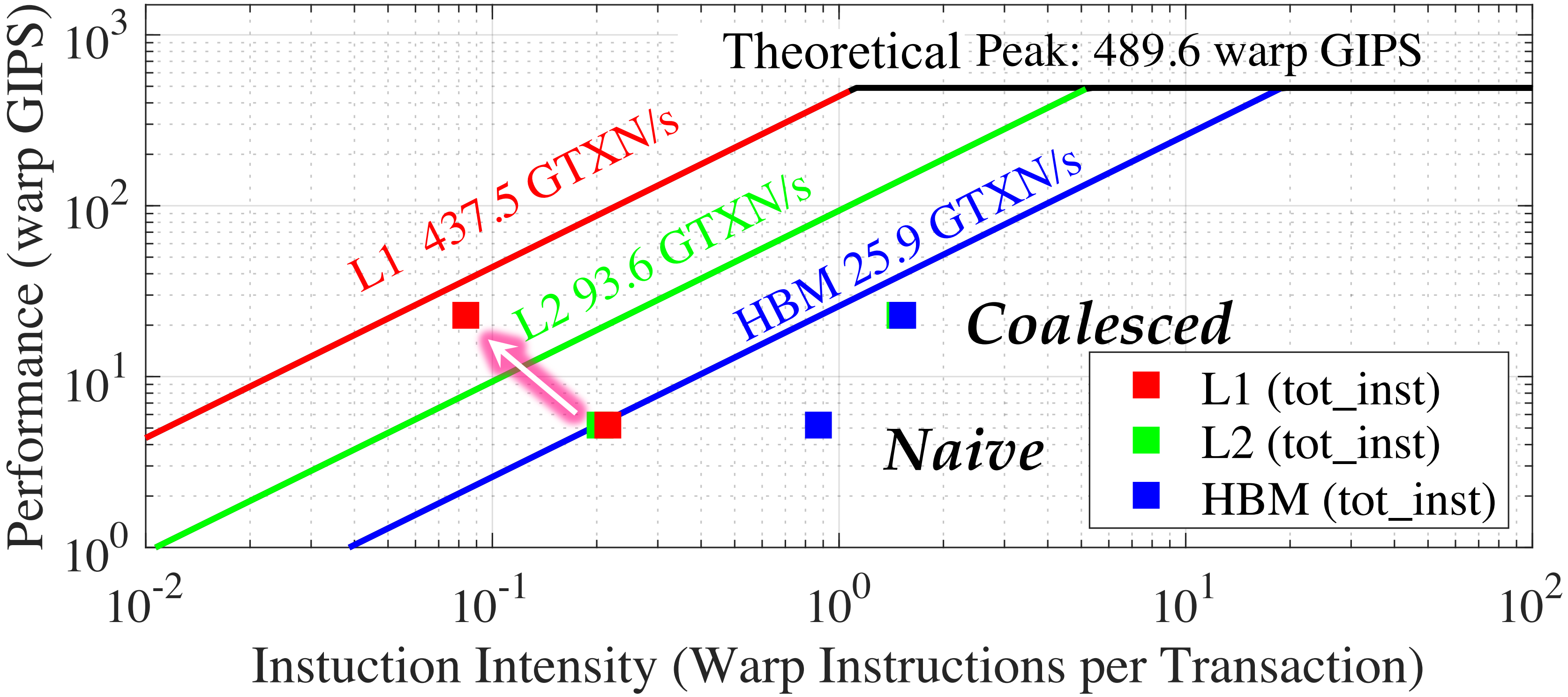
Coalesced Implementation



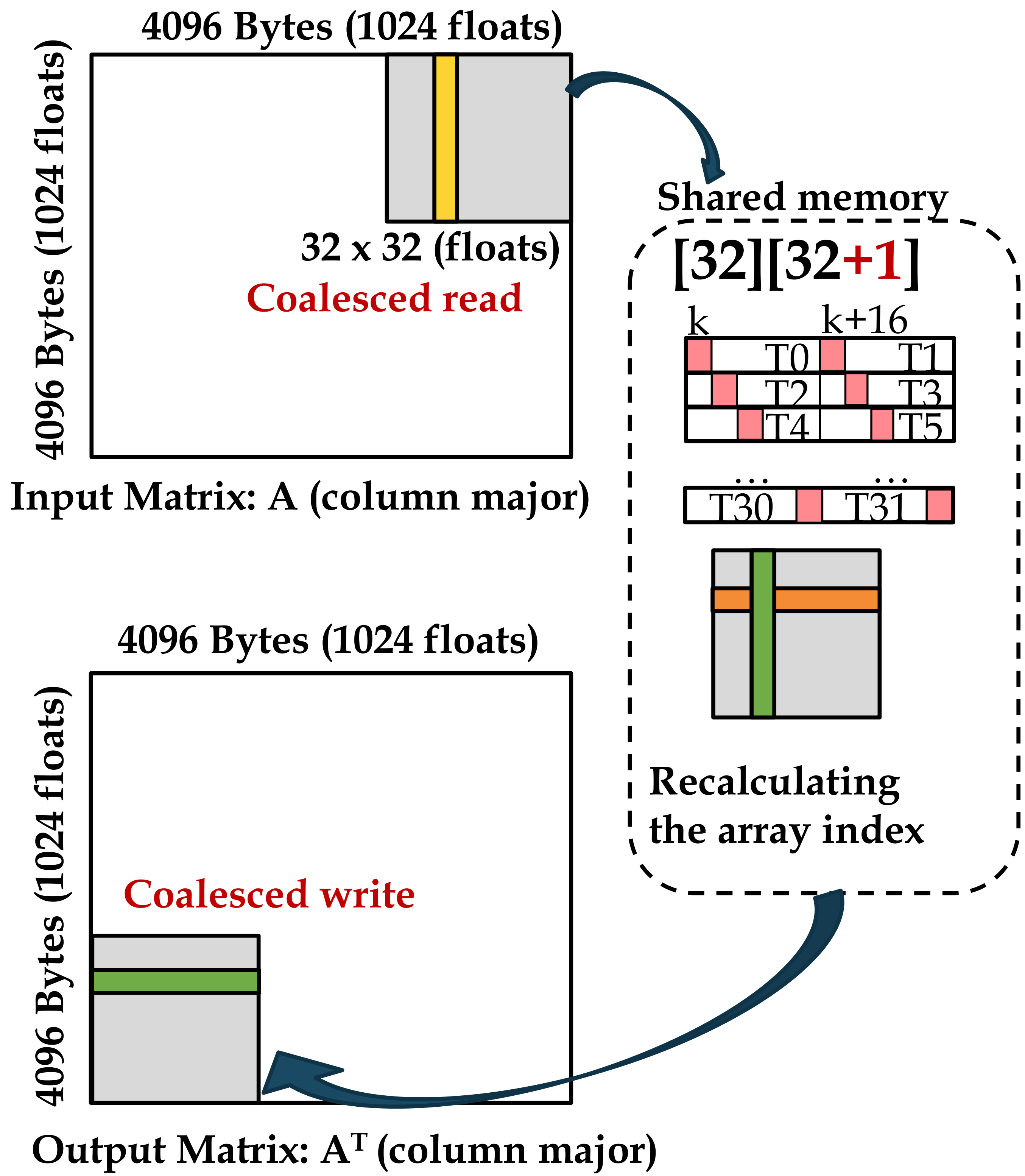
Shared memory bank access



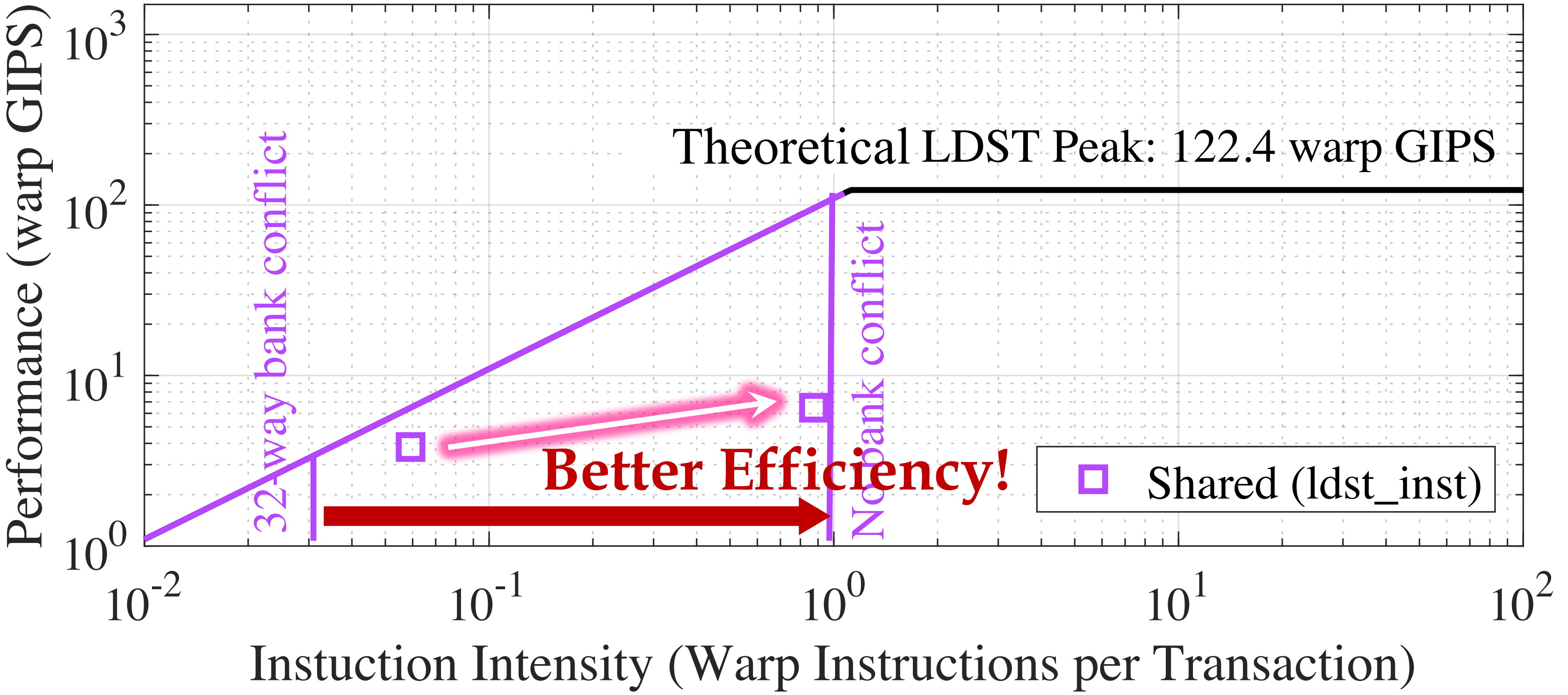
Instruction Throughput



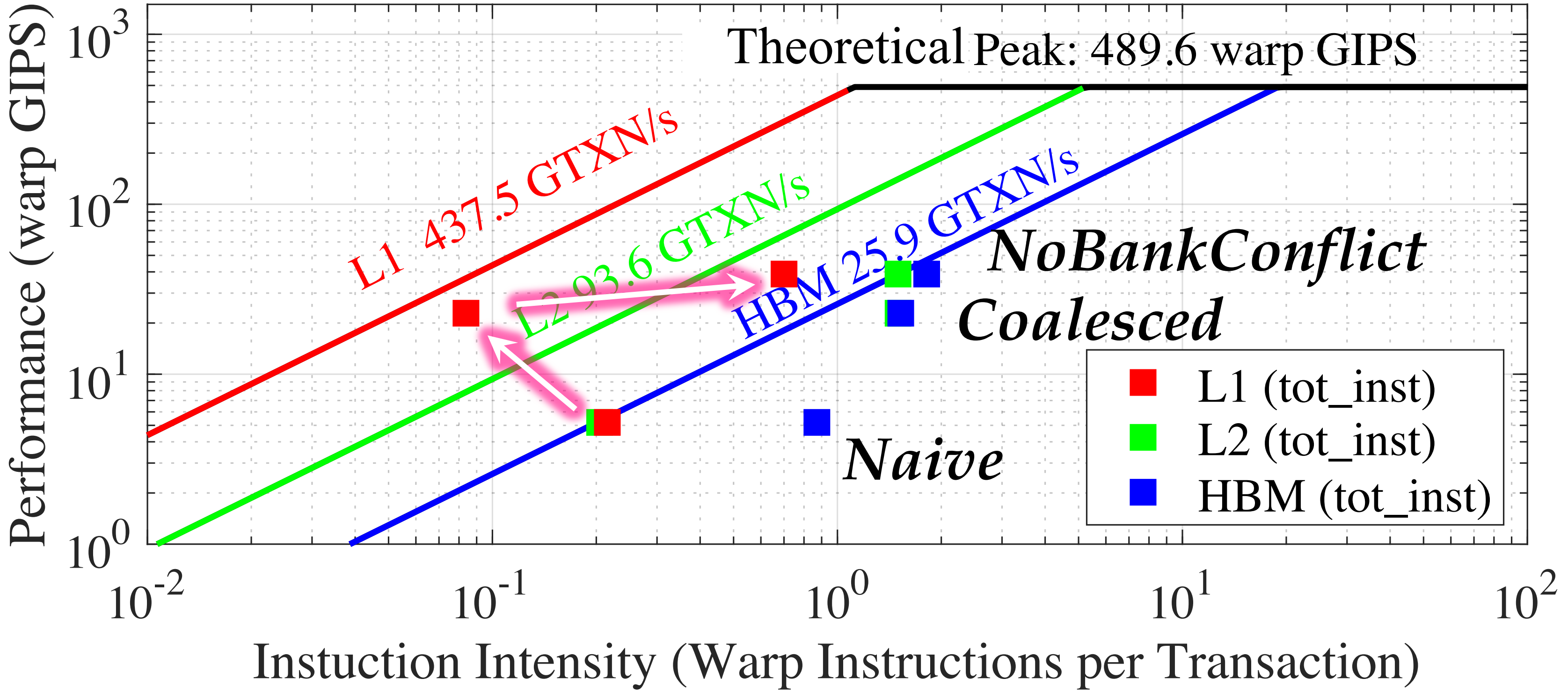
Coalesced_NoBankConflict Implementation



Shared memory bank access



Instruction Throughput



Summary

- **Instruction Throughput**
  - Expanding the applicability of roofline to several emerging computational domains

- **Global Memory Access Patterns**
  - Quantify the memory access pattern, e.g. unit-stride vs. gather/scatter

- **Shared Memory Access Patterns**
  - Denote the efficiency of shared memory access.

There's more in the paper !!

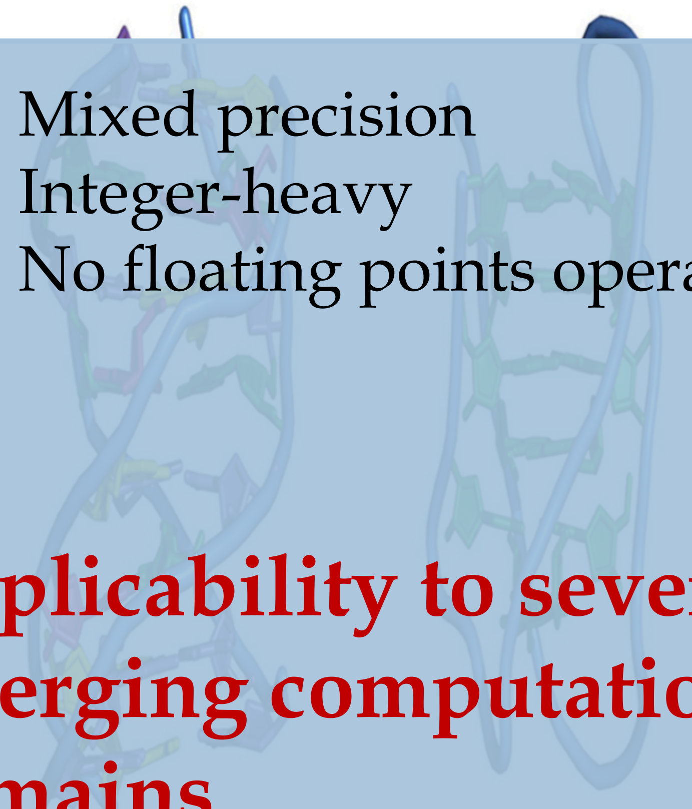
- **Thread Predication**
- **More Examples**
  - **HPGMG (mixed precision): three implementations.**
  - **BatchSW (integer-only): two implementations**

- Tensor core
  - WMMA
  - cuBLAS

Closing Thoughts

What the Instruction Roofline Models Tell us…

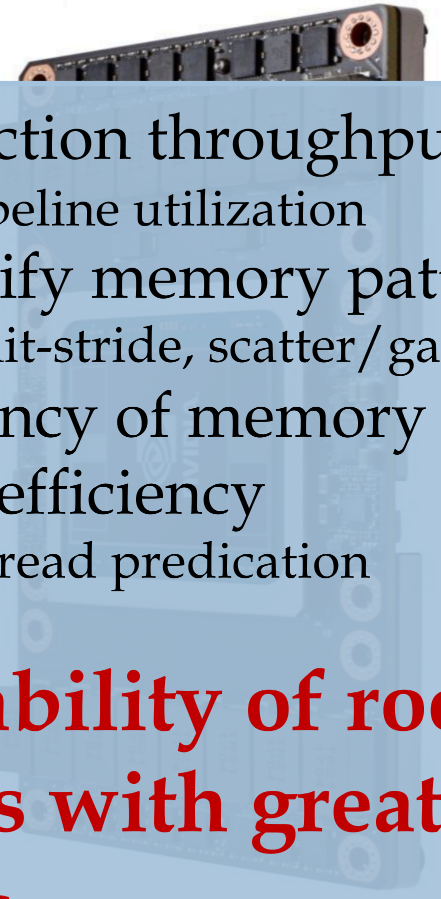
Emerging Domains



- Mixed precision
- Integer-heavy
- No floating points operations

Applicability to several emerging computational domains

Architectural Evolution



- Instruction throughput
  - pipeline utilization
- Quantify memory pattern
  - Unit-stride, scatter/gather
- Efficiency of memory access
- Warp efficiency
  - Thread predication

Applicability of roofline to GPUs with greater insights

Practical Use

- Unified visualization of bandwidth and access efficiency

Rapidly tell how different aspects of modern GPU architectures constrain performance.

Future Work

- Apply our methodology to other accelerated architectures
- Extend the access efficiency concept to networking, I/O, and Lustre file systems.

Acknowledgement

- 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 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 No. DE-AC02-05CH11231 and the Oak Ridge Leadership Facility which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
- We thank NVIDIA Corporation for their willingness to answer our myriad of questions on nvprof metrics.

Questions?

Backup