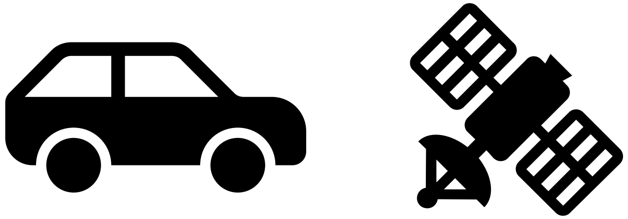
Arithmetic-Intensity-Guided Fault Tolerance for Neural Network Inference on GPUs

Jack Kosaian Carnegie Mellon University
Rashmi Vinayak Carnegie Mellon University

Contact: jkosaian@cs.cmu.edu, rvinayak@cs.cmu.edu

Efficiently detect silent data corruption in neural network inference by exploiting trends in neural network design and GPUs

Many ML systems demand high reliability



Autonomous edge systems

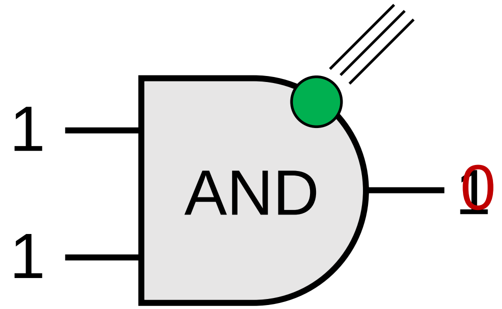
Financial systems

Scientific discovery

Cybersecurity

Soft errors threaten reliability

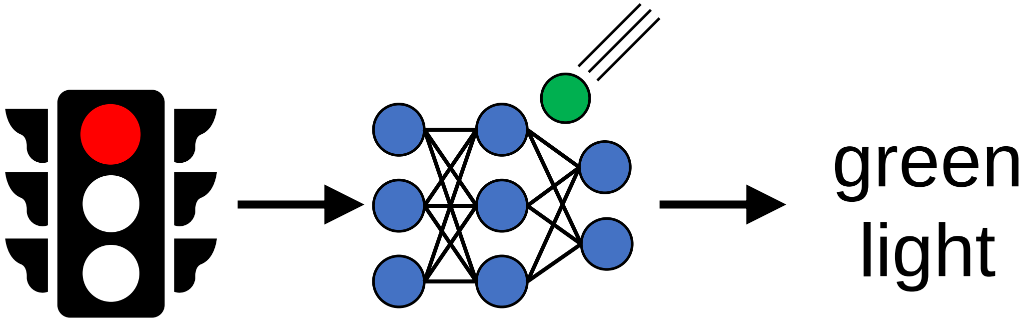
- *Soft error*: transient hardware error causing incorrect execution
  - Incorrect execution (i.e., silent data corruption): e.g., 1 + 1 = 3
  - Transient: may occur one cycle, but may not occur in next
- Many causes:
  - Cosmic-radiation-induced particle strikes
  - Aggressive voltage scaling
  - Hardware wear out



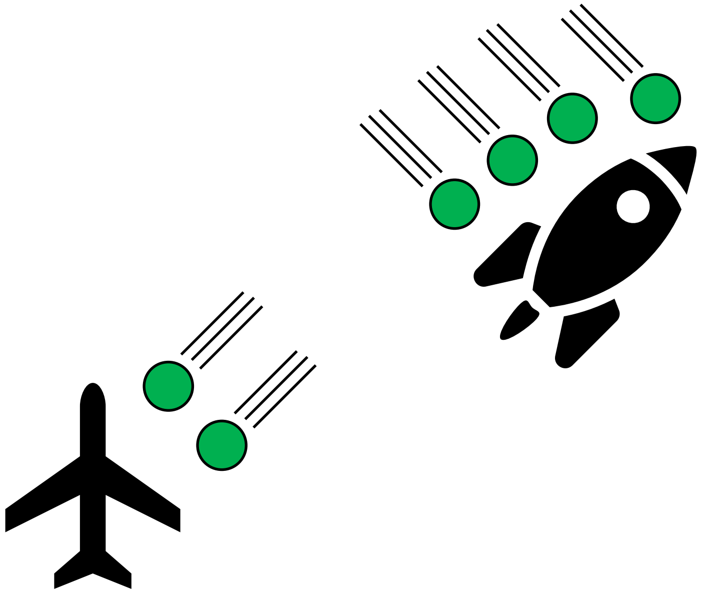
- Affect both memory and processing elements
- Rate off occurrence depends on setting
  - Infrequent terrestrially (though uptick noted recently in datacenters)
  - Rate increases with altitude, even more prevalent in space

When do soft errors matter for neural networks?

1. Safety-critical systems
   - Li et al., 2017: can cause misprediction rate that violates automotive safety standards (ISO 26262)



2. Environments with high error rates
   - Soft error rate increases with altitude
   - Even higher when operating in outer space



Sphere of focus for this talk

Detecting faults in processing logic in NN inference on GPUs

- *Detecting faults*: rare events, can fail over to reliable backup
  - Specifically, we focus on detecting a single fault occurring
- *Faults in processing logic:*
  - Memory faults are easier to handle via hardware protection (e.g., ECC)
  - Processing logic is less amenable to lightweight hardware protection
- **Goal:** minimize execution-time overhead

Algorithm-based fault tolerance (ABFT)

ABFT: add redundant computation carefully formed to introduce invariants into algorithm that can be used for fault tolerance

→ Less overhead than replication-based approaches

Algorithm-based fault tolerance (ABFT)

Example: detect faults in $F(x) = 2x$

$$x_1 \longrightarrow F \longrightarrow 2x_1$$

$$2x_1 + 2x_2$$

$$x_2 \longrightarrow F \longrightarrow 2x_2$$

compare

$$x_C = (x_1 + x_2) \longrightarrow F \longrightarrow 2x_C = 2(x_1+x_2)$$

“encoding”

Requires only one additional invocation of F

Algorithm-based fault tolerance (ABFT)

Example: detect faults in $F(\vec{x}) = A\vec{x}$

$$\overrightarrow{x_1} \longrightarrow F \longrightarrow A\overrightarrow{x_1}$$

$$\overrightarrow{x_2} \longrightarrow F \longrightarrow A\overrightarrow{x_2}$$

$$A\overrightarrow{x_1} + A\overrightarrow{x_2}$$

compare

$$\overrightarrow{x_C} = (\overrightarrow{x_1} + \overrightarrow{x_2}) \longrightarrow F \longrightarrow A\overrightarrow{x_C} = A(\overrightarrow{x_1} + \overrightarrow{x_2})$$

“encoding”

Applies to any linear function *F*

Non-linear functions are harder to support

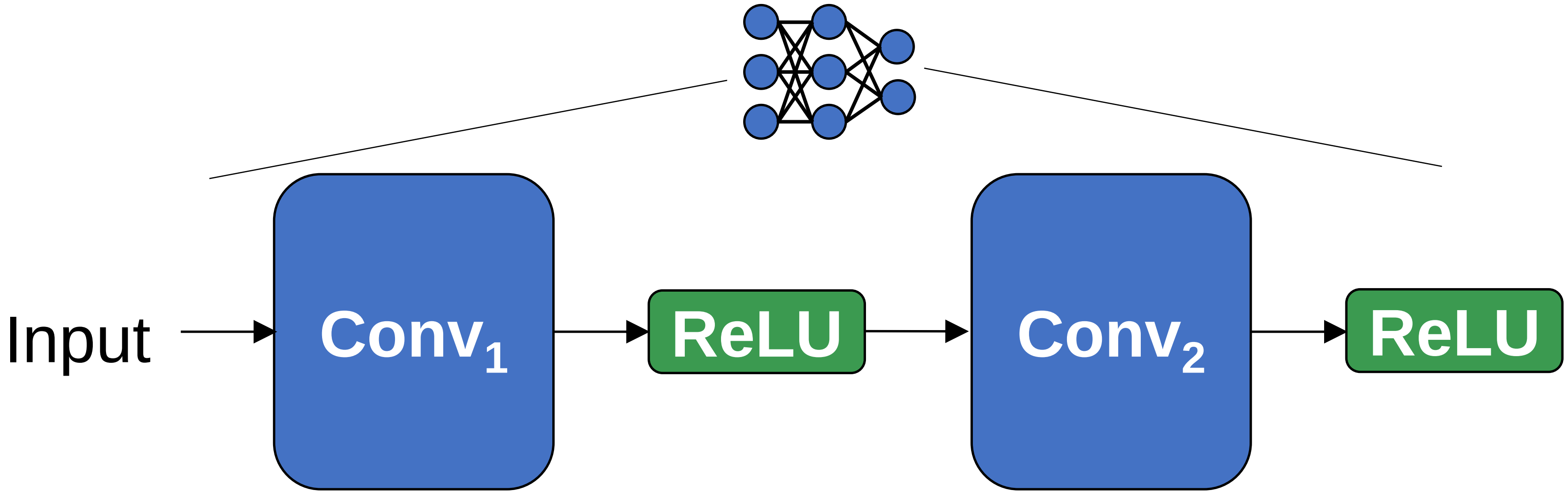
ABFT has been widely studied

- Traditional HPC applications
  - Linear algebra
  - Iterative methods
  - Sorting

- Neural networks
  - On GPUs (Hari et al., 2020)
  - On CPUs (Zhao et al., 2020; Li et al., 2021)
  - In hardware (Ozen et al., 2019)

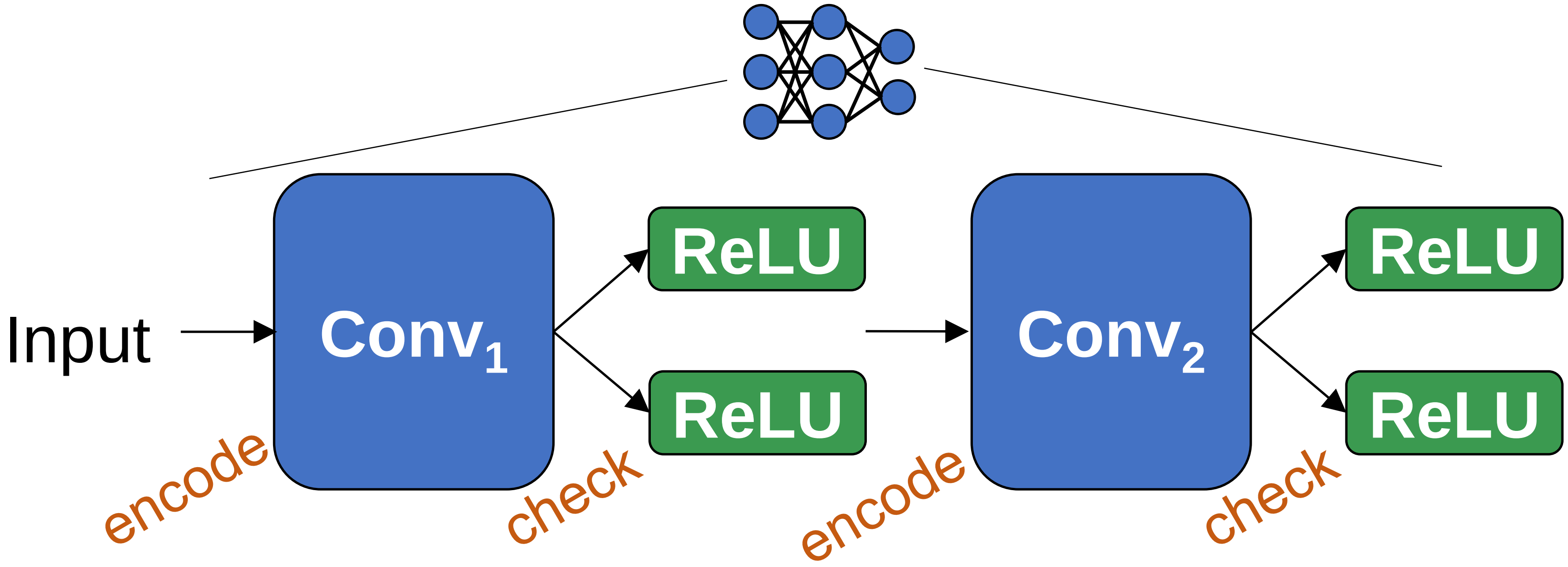
ABFT for neural networks

- **Problem:** ABFT not widely applicable to non-linear operations
- Neural networks contain:
  - Linear layers (e.g., convolutions, fully-connected layers)
  - Non-linear layers (e.g., ReLUs, max pooling)



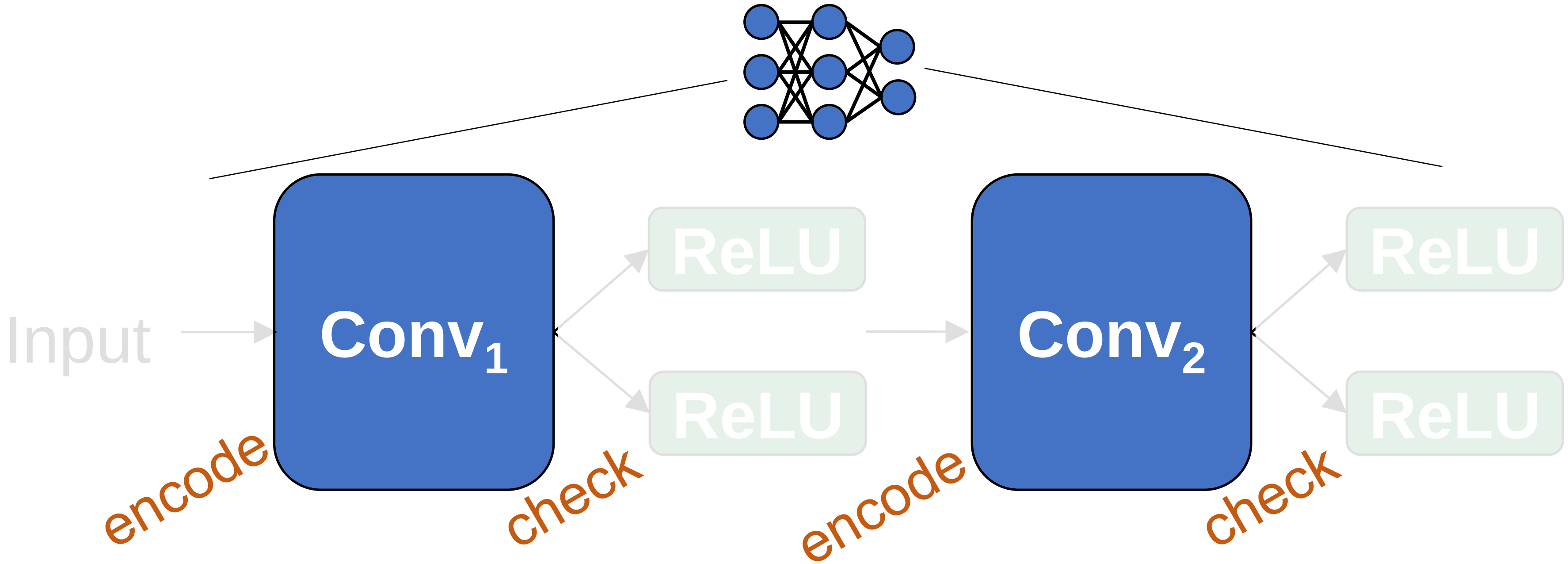
ABFT for neural networks

- Approach commonly used in prior work:
  - ABFT over linear layers
  - Replicate non-linear layers (which are cheap to begin with)



ABFT for neural networks

- Approach commonly used in prior work:
  - ABFT over linear layers
  - Replicate non-linear layers (which are cheap to begin with)
- **Our focus:** efficient ABFT for lin. layers (matrix multiplications)



ABFT for matrix multiplication

A × B = C

A:

a_1	a_2
a_3	a_4

$a_1 + a_3$	$a_2 + a_4$

column checksum

B:

b_1	b_2
b_3	b_4

$b_1 + b_2$
$b_3 + b_4$

row checksum

C:

$a_1b_1 + a_2b_3$	$a_1b_2 + a_2b_4$
$a_3b_1 + a_4b_3$	$a_3b_2 + a_4b_4$

ABFT for matrix multiplication

$$\underbrace{\begin{array}{|c|c|}\hline a_1 & a_2 \\ \hline a_3 & a_4 \\ \hline \end{array}}_{A} \times \underbrace{\begin{array}{|c|c|}\hline b_1 & b_2 \\ \hline b_3 & b_4 \\ \hline \end{array}}_{B} = \underbrace{\begin{array}{|c|c|}\hline a_1b_1 + a_2b_3 & a_1b_2 + a_2b_4 \\ \hline a_3b_1 + a_4b_3 & a_3b_2 + a_4b_4 \\ \hline \end{array}}_{C}$$

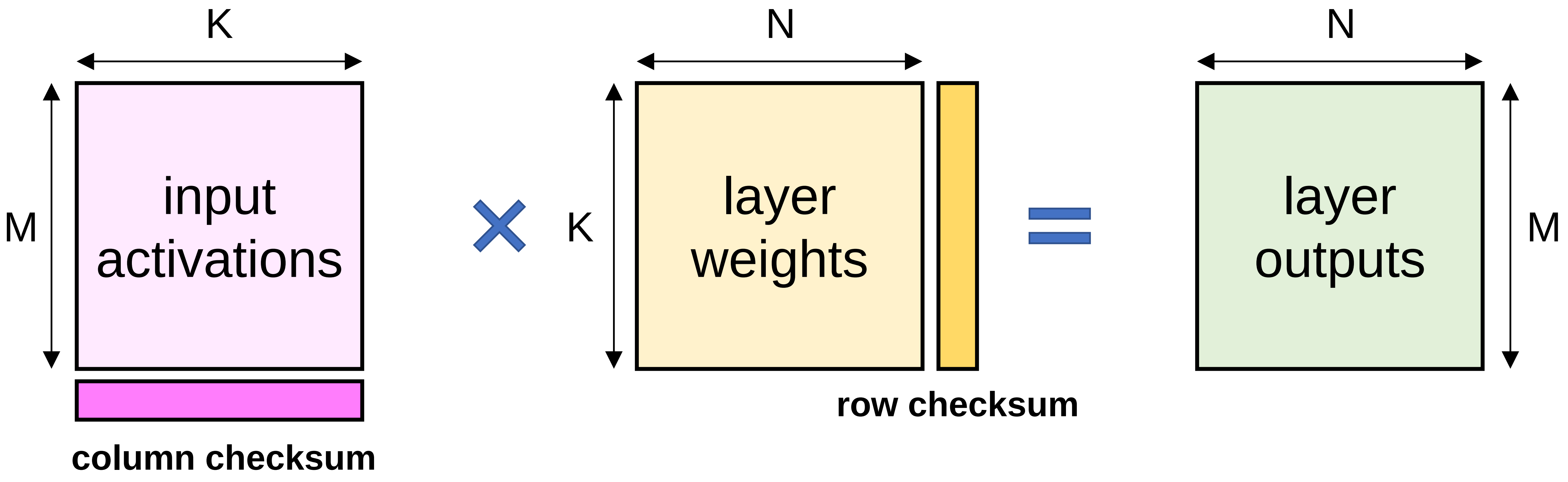
sum

$$\begin{array}{|c|c|}\hline a_1 + a_3 & a_2 + a_4 \\ \hline \end{array} \times \begin{array}{|c|}\hline b_1 + b_2 \\ \hline b_3 + b_4 \\ \hline \end{array} = (a_1+a_3)(b_1+b_2) + (a_2+a_4)(b_3+b_4)$$

column checksum

row checksum

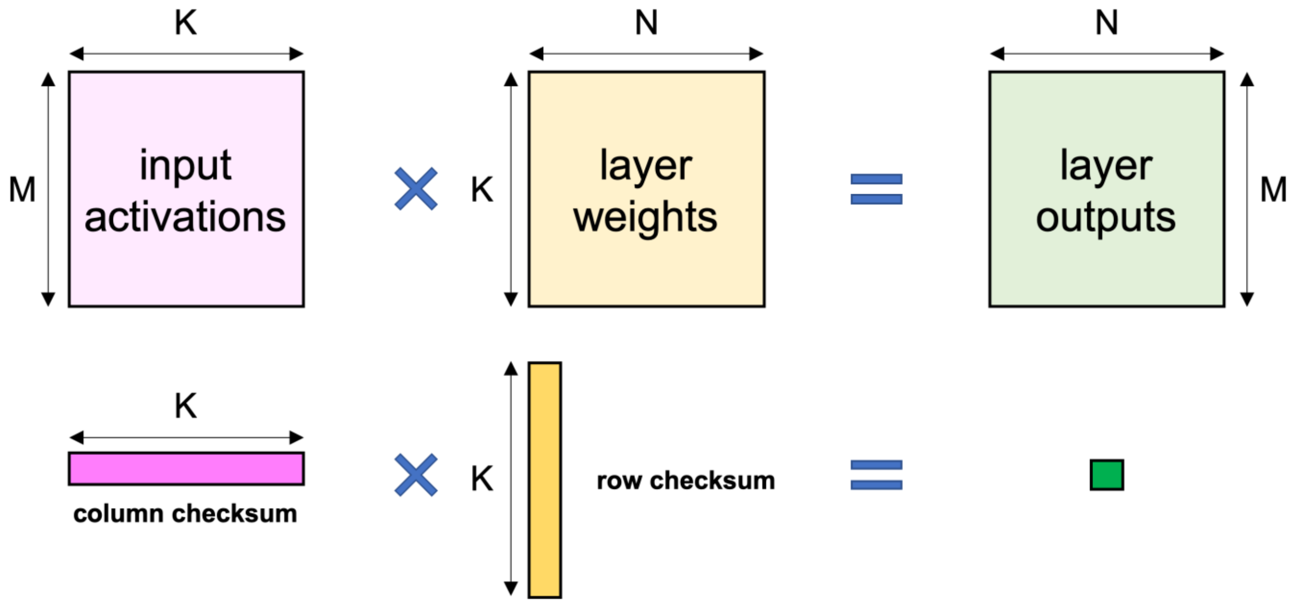
ABFT for linear layers in neural networks



ABFT for linear layers in NNs

“Global ABFT”

- Approach used by prior work
- Generates checksums over entire matrices
- Minimizes redundant computation performed in checksum dot products



Is global ABFT efficient for all linear layers on GPUs?

What is needed for efficient error detection?

Goal: minimize execution-time overhead of error detection

- Must understand resource bottlenecks to reduce overhead
  - *Compute-bound:* minimize additional operations performed
  - *Memory-bandwidth-bound:* minimize additional loads/stores
    - Compute units underutilized → opportunities for fine-grained redundancy

Our contributions

- Analyze trends in NN design and GPU hardware
- Make a case for prevalence of bandwidth-bound linear layers
  - Opens opportunities for efficient fault detection that prior ABFT can't exploit
- Investigate approaches to ABFT suitable for bandwidth-bound layers
- Develop arithmetic-intensity-guided ABFT
  - Adaptive approach that selects most efficient ABFT scheme for each layer

Determining whether compute or bandwidth bound

To be compute bound:

$$\frac{\text{FLOPs}}{\text{Bytes}} > \frac{\text{FLOPs/sec}}{\text{Bytes/sec}}$$

arithmetic intensity > **CMR: compute-to-memory-bandwidth ratio**

Comparing intensity and CMR for neural networks on GPUs

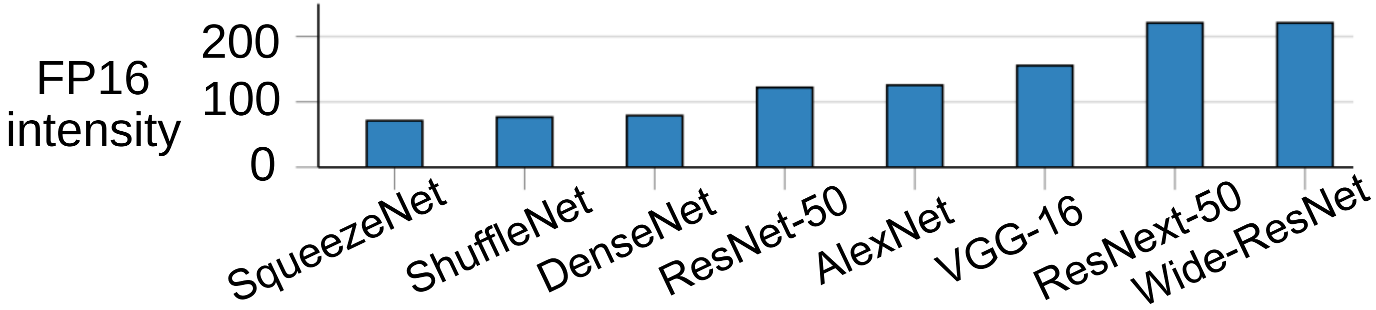
arithmetic intensity **vs.** **CMR: compute-to-memory-bandwidth ratio**

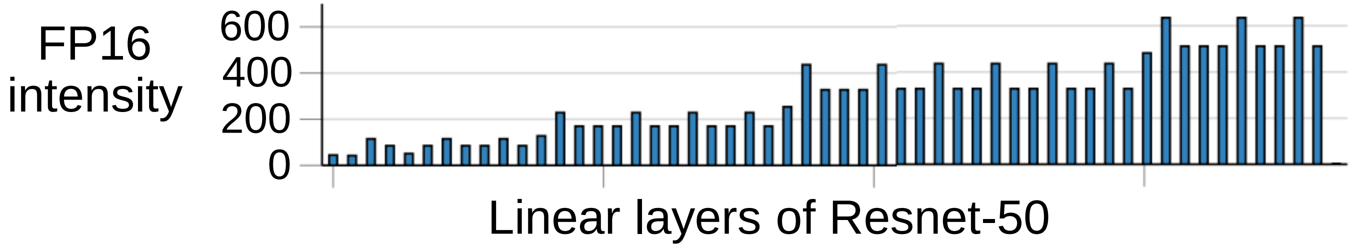
Comparing intensity and CMR for neural networks on GPUs

arithmetic intensity

- Variable across:
  - Neural networks as a whole
  - Layers within a single network
  - Deployment scenarios

$$\text{arithmetic intensity} = \frac{\text{FLOPs}}{\text{Bytes}}$$





Higher resolution → higher AI

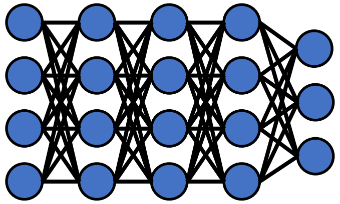
Larger batch size → higher AI

Comparing intensity and CMR for neural networks on GPUs

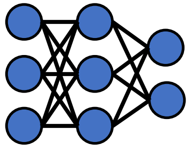
arithmetic intensity

- Variable across:
    - Neural networks as a whole
    - Layers within a single network
    - Deployment scenarios
- Trends in neural architecture design reduce intensity

Large DNNs:
High intensity



Small DNNs:
Low intensity



Techniques that improve throughput/latency, but reduce arithmetic intensity:

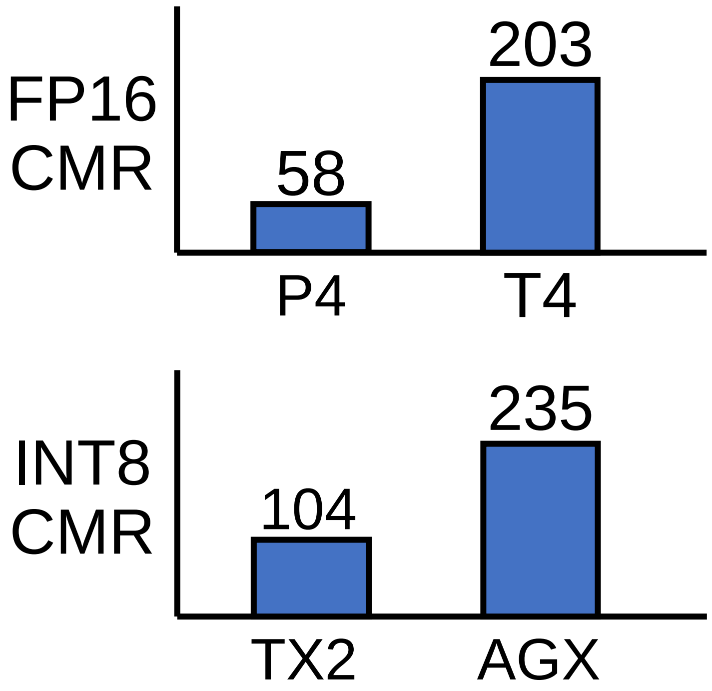
- Pruning
- Model specialization
- Model scaling (e.g., EfficientNets)

Comparing intensity and CMR for neural networks on GPUs

$$\uparrow \text{CMR} = \frac{\text{FLOPs/sec} \uparrow\uparrow}{\text{Bytes/sec} \uparrow}$$

Server-grade GPUs

Edge GPUs



CMR: compute-to-memory-bandwidth ratio

- Increasing with inference-optimized GPUs
  - Tensor Cores cause large increase in compute bandwidth
  - Memory bandwidth has not increased as rapidly

Comparing intensity and CMR for neural networks on GPUs

arithmetic intensity vs. **CMR: compute-to-memory-bandwidth ratio**

- Variable across:
  - Neural networks as a whole
  - Layers within a single network
  - Deployment scenarios
- Trends in neural architecture design decrease intensity

- Increasing with inference-optimized GPUs
  - Tensor Cores cause large increase in compute bandwidth
  - Memory bandwidth has not increased as rapidly

Implication: neural network inference is likely to contain *both* compute-bound and memory-bandwidth-bound layers.

Any one-size-fits all approach to fault detection will be inefficient.

Our approach: *arithmetic-intensity-guided ABFT*

Key idea: adapt the type of fault detection used depending on bottleneck of layer

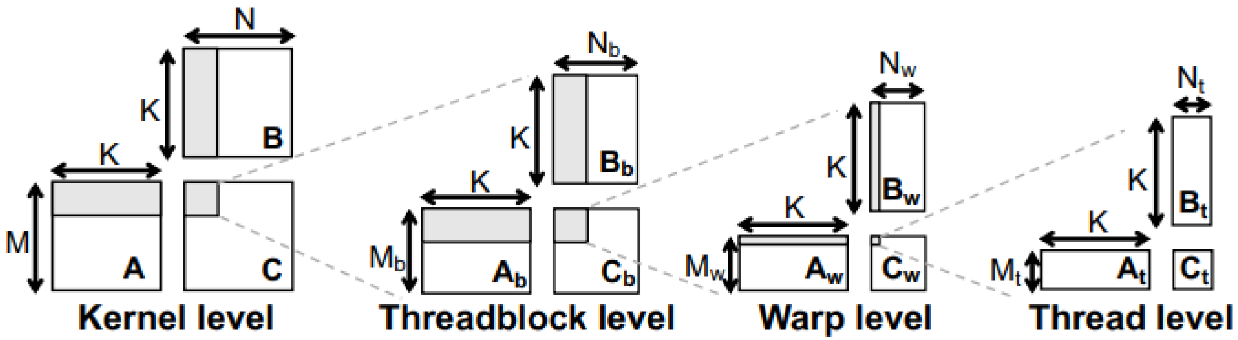
- Compute-bound layers: global ABFT
- Bandwidth-bound layers: ???

- We investigate and propose:
  - Thread-level ABFT: approach to ABFT for bandwidth-bound layers
  - Arithmetic-intensity-guided ABFT: adaptive approach to ABFT that selects between global and thread-level ABFT

Fault detection for bandwidth-bound layers

Design principle: avoid additional memory accesses whenever possible, even at the expense of additional computation

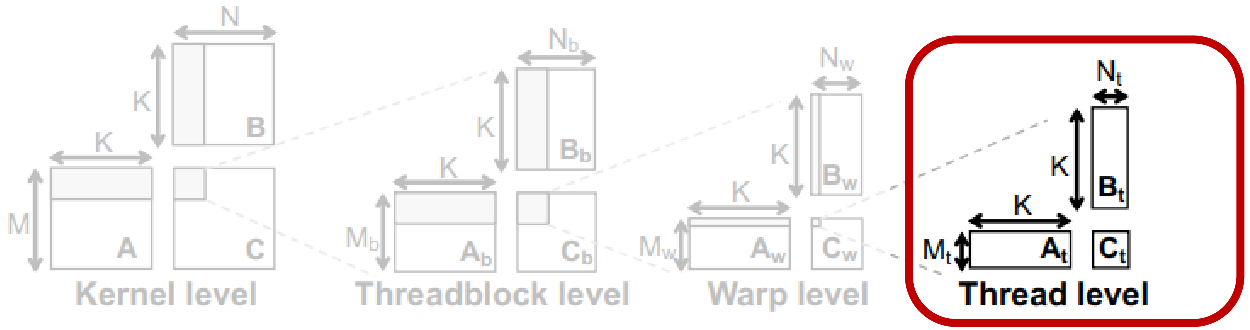
- Avoids competing with original layer for bottleneck resource: bandwidth
- Global ABFT requires additional loads/stores for inter-thread communication



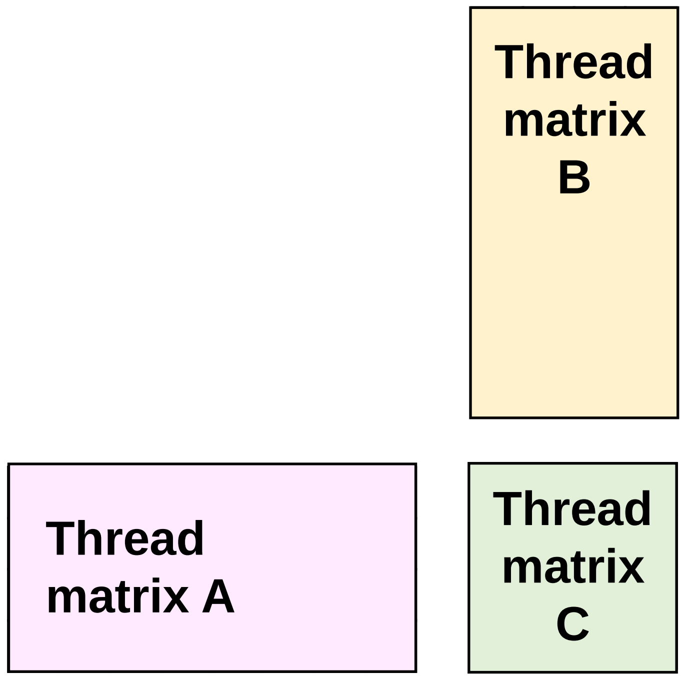
Opportunity: compute units will stall in bandwidth-bound layers

- Ideal approach will fill these stalls with fault detection

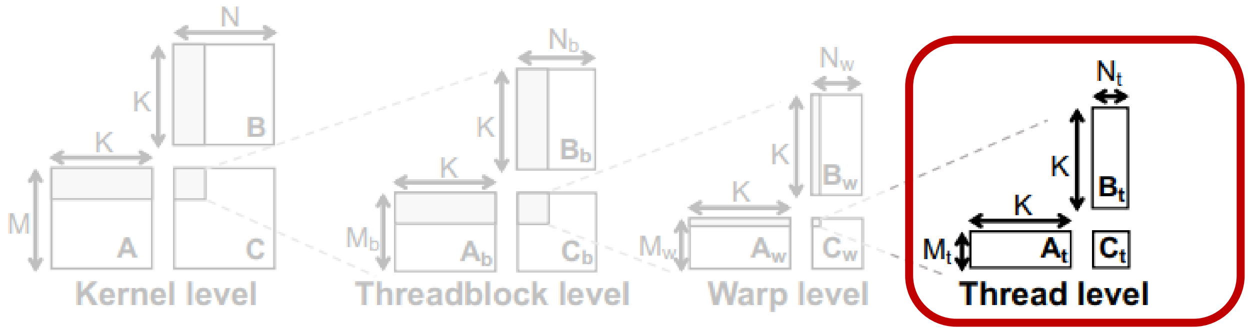
ABFT for bandwidth-bound linear layers

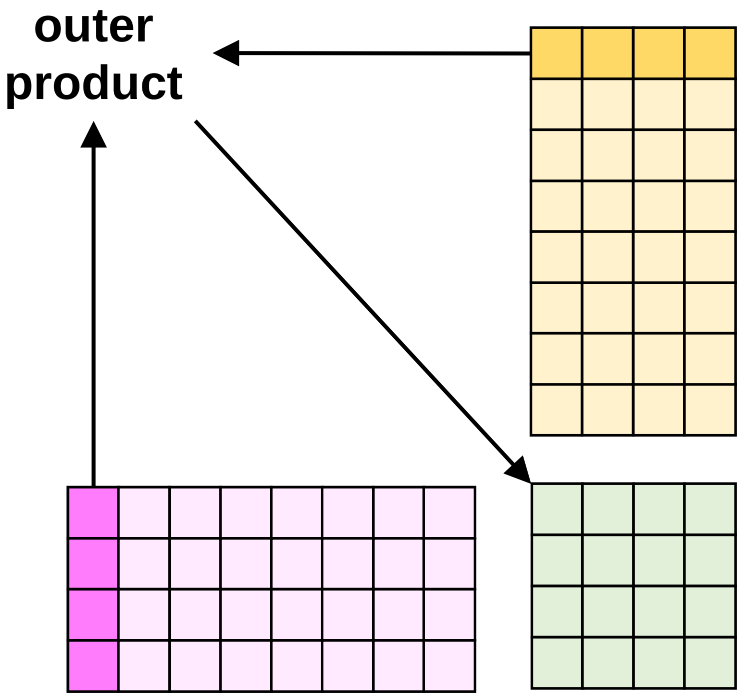


- *Thread-level ABFT:*
  - Each GPU thread performs thread-local ABFT alongside original mat. mult.



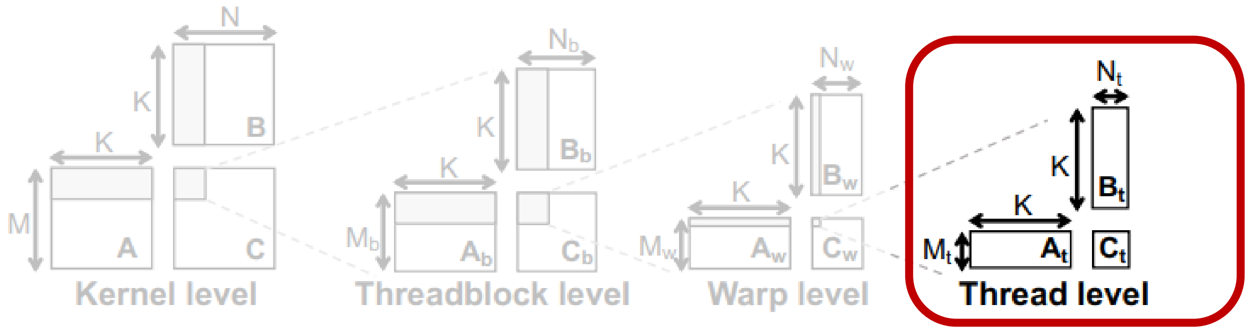
ABFT for bandwidth-bound linear layers

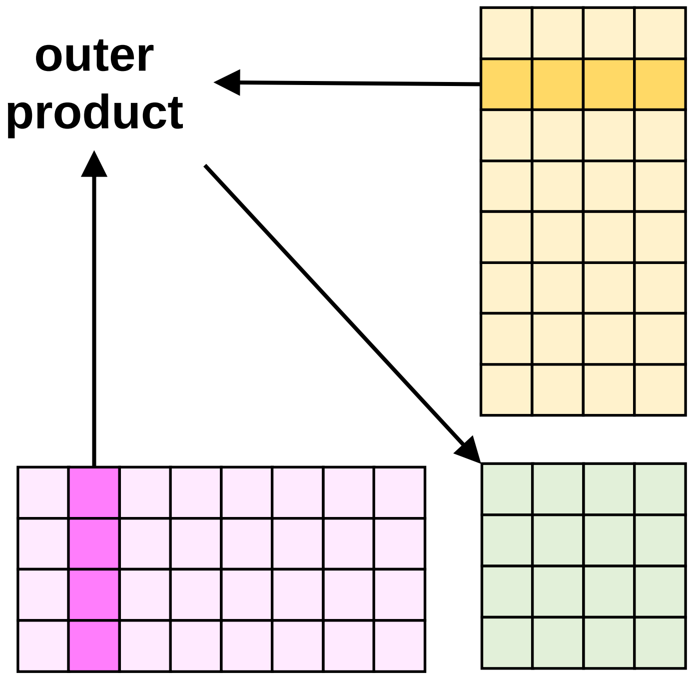




- *Thread-level ABFT:*
  - Each GPU thread performs thread-local ABFT alongside original mat. mult.

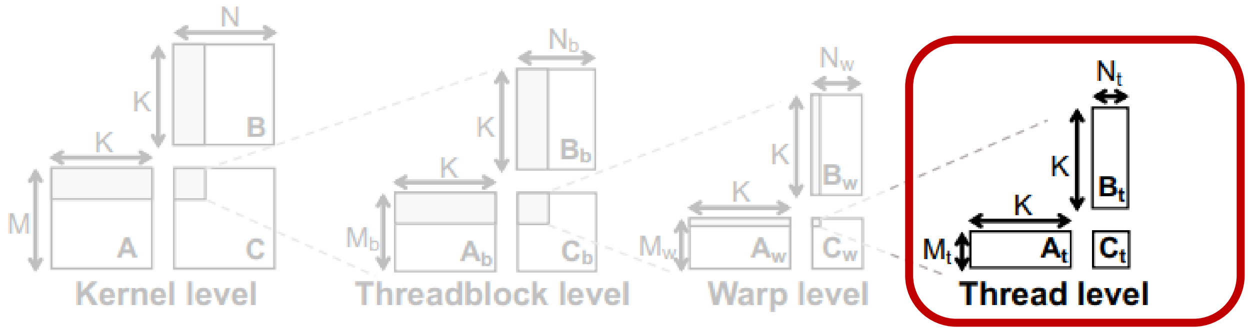
ABFT for bandwidth-bound linear layers

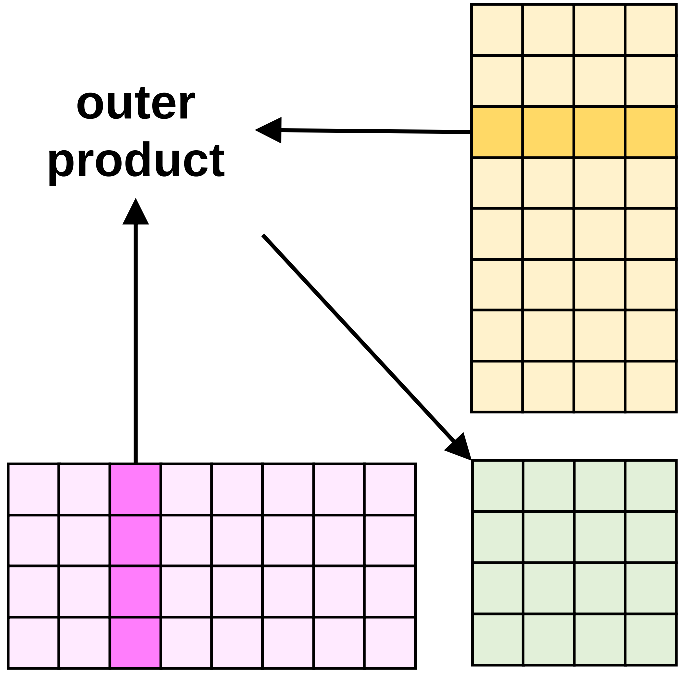




- *Thread-level ABFT:*
  - Each GPU thread performs thread-local ABFT alongside original mat. mult.

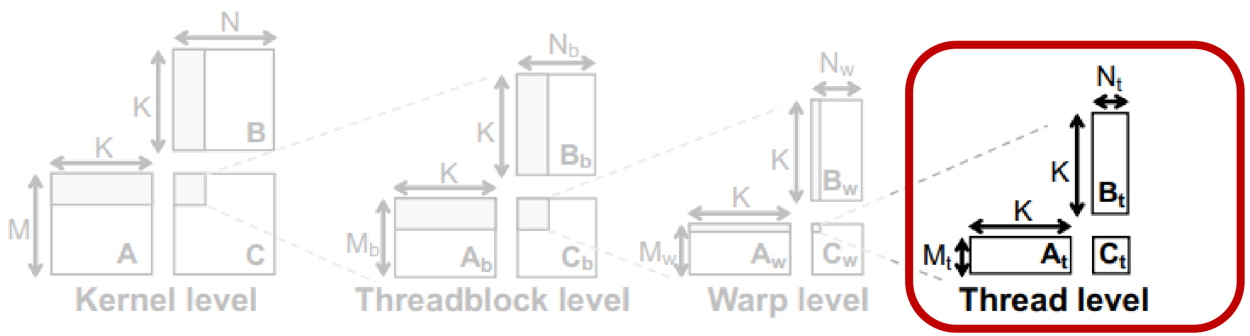
ABFT for bandwidth-bound linear layers

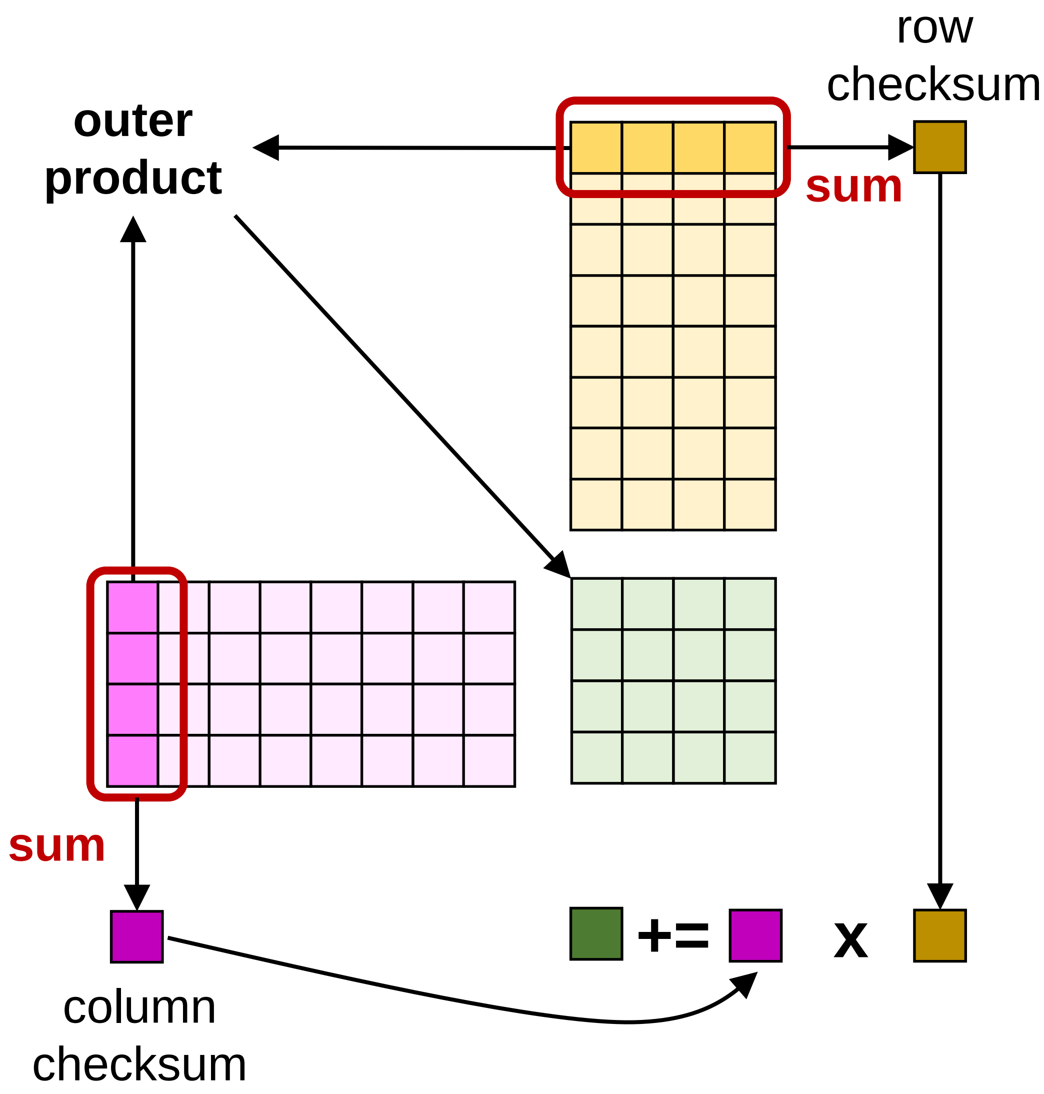




- *Thread-level ABFT:*
  - Each GPU thread performs thread-local ABFT alongside original mat. mult.

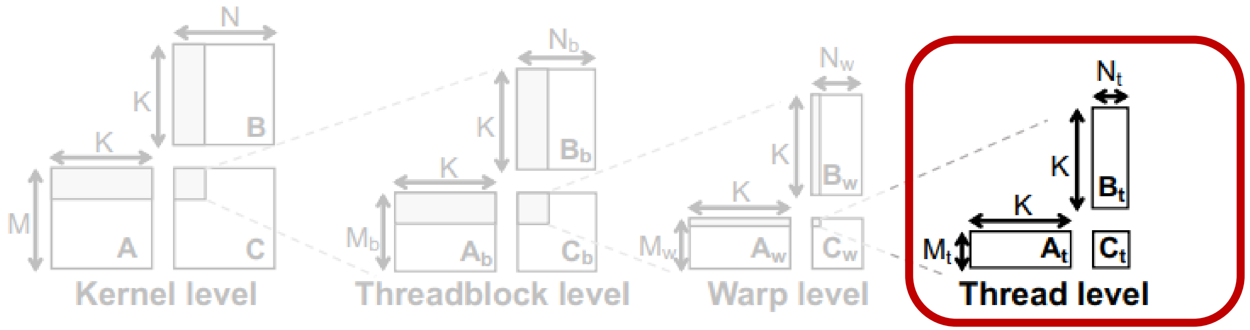
ABFT for bandwidth-bound linear layers



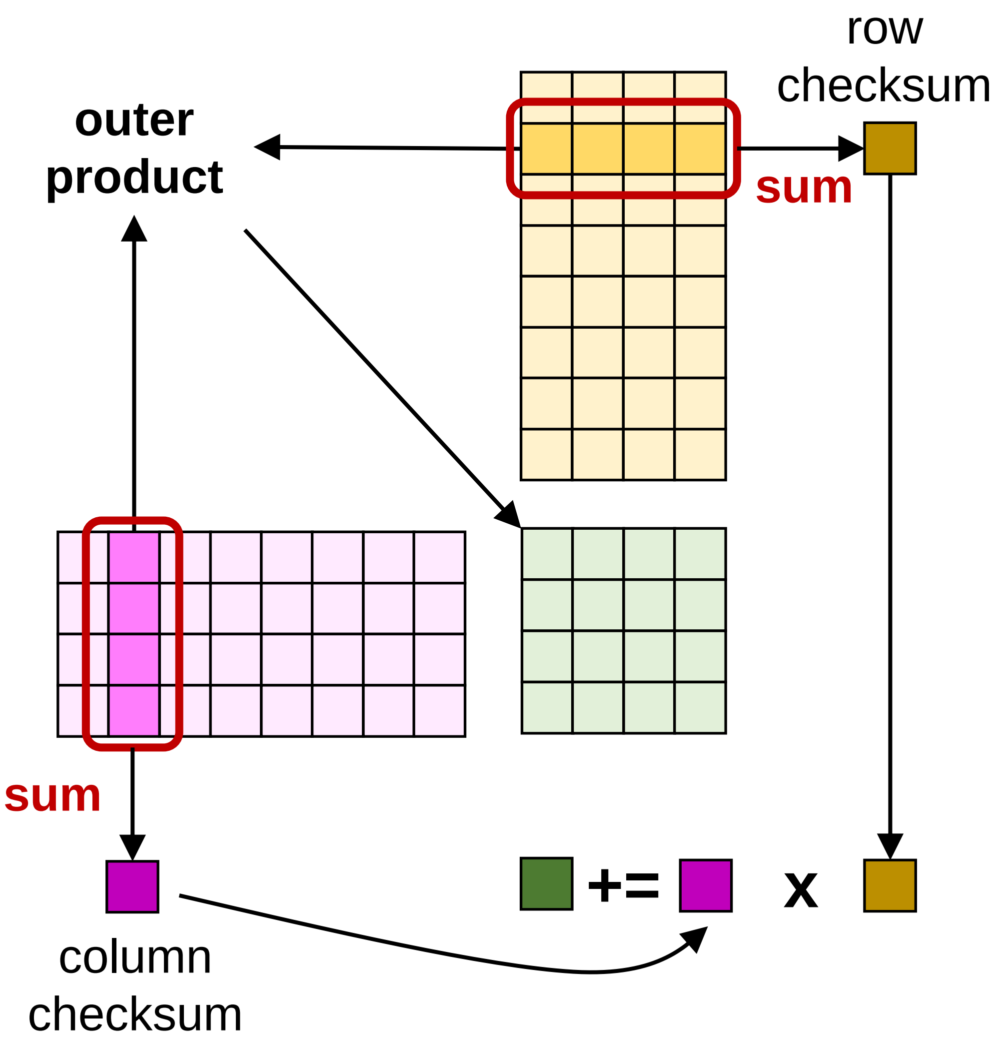


- *Thread-level ABFT:*
  - Each GPU thread performs thread-local ABFT alongside original mat. mult.

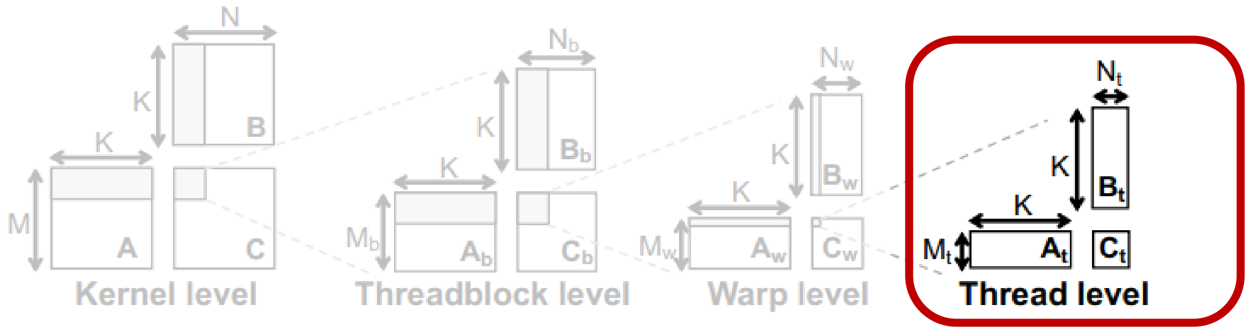
ABFT for bandwidth-bound linear layers



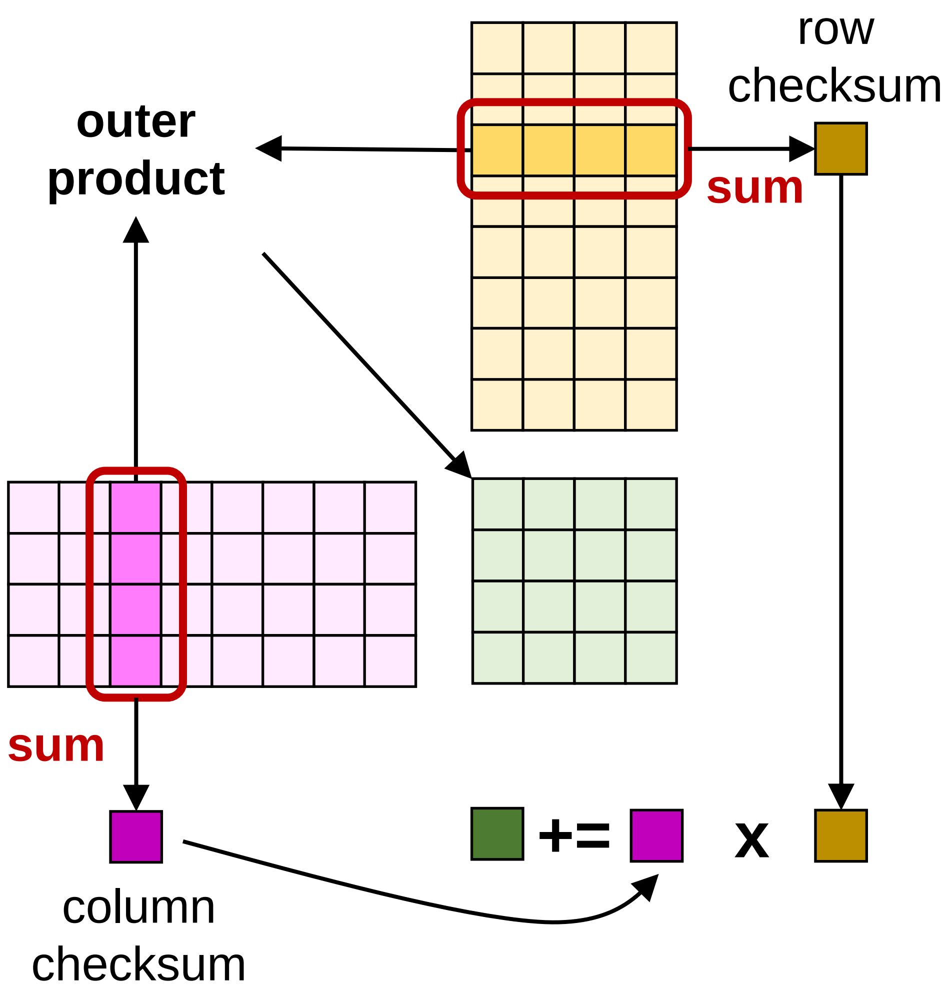
- *Thread-level ABFT:*
  - Each GPU thread performs thread-local ABFT alongside original mat. mult.



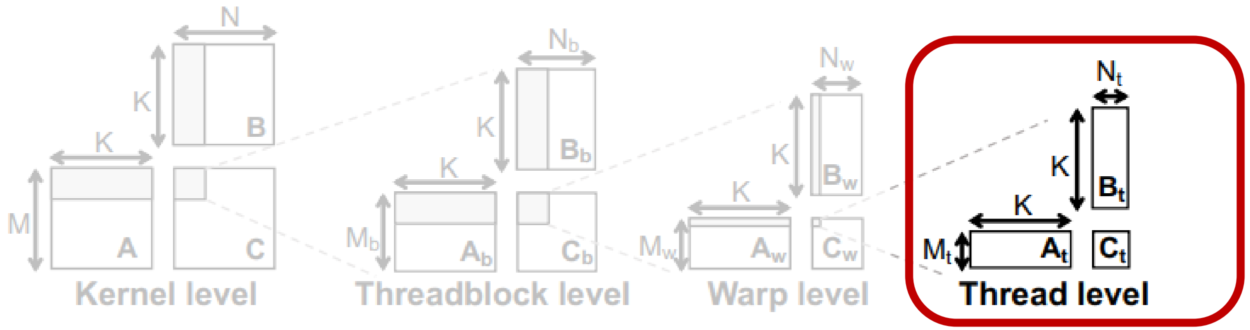
ABFT for bandwidth-bound linear layers



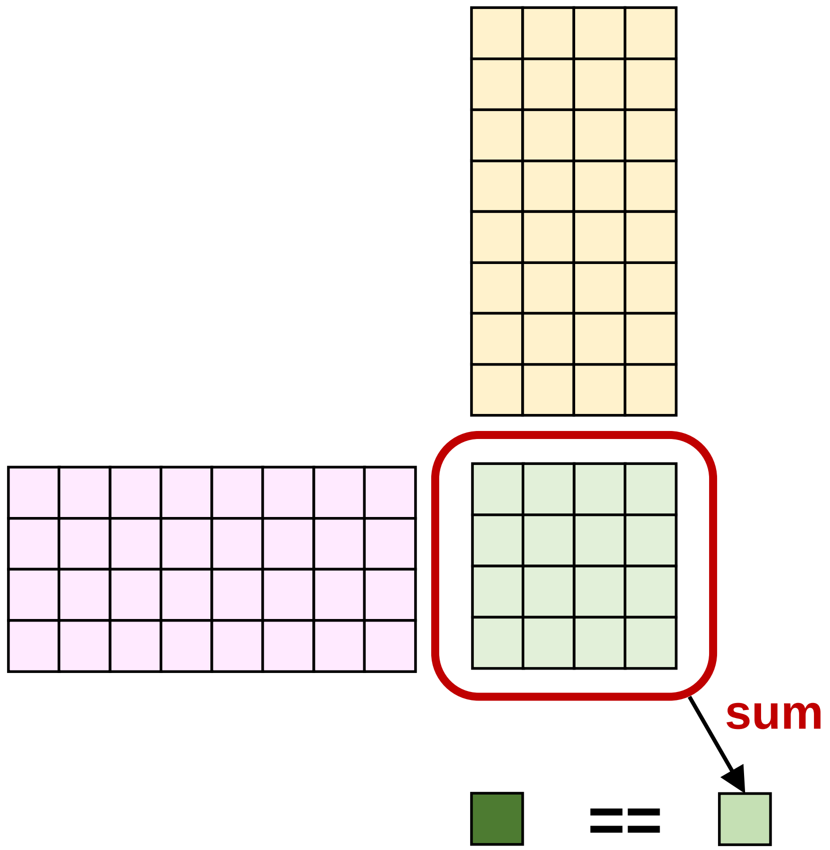
- *Thread-level ABFT:*
  - Each GPU thread performs thread-local ABFT alongside original mat. mult.



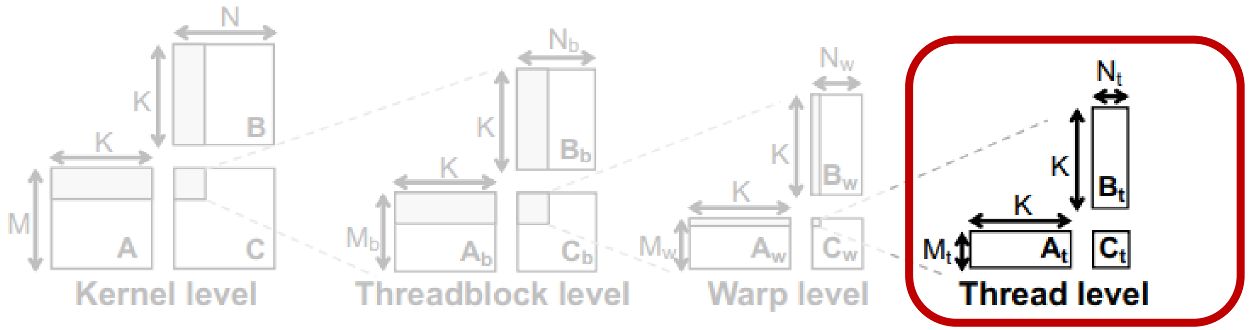
ABFT for bandwidth-bound linear layers

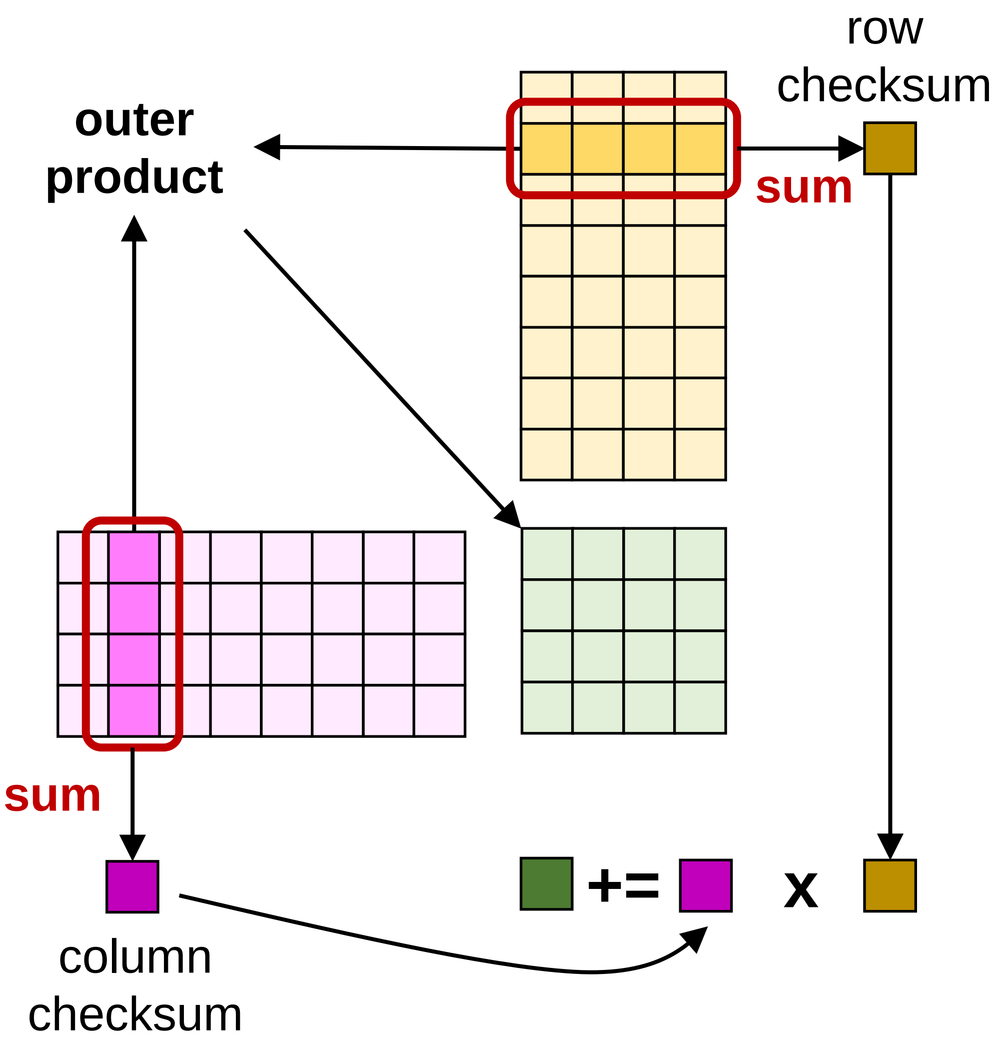


- *Thread-level ABFT:*
  - Each GPU thread performs thread-local ABFT alongside original mat. mult.



ABFT for bandwidth-bound linear layers

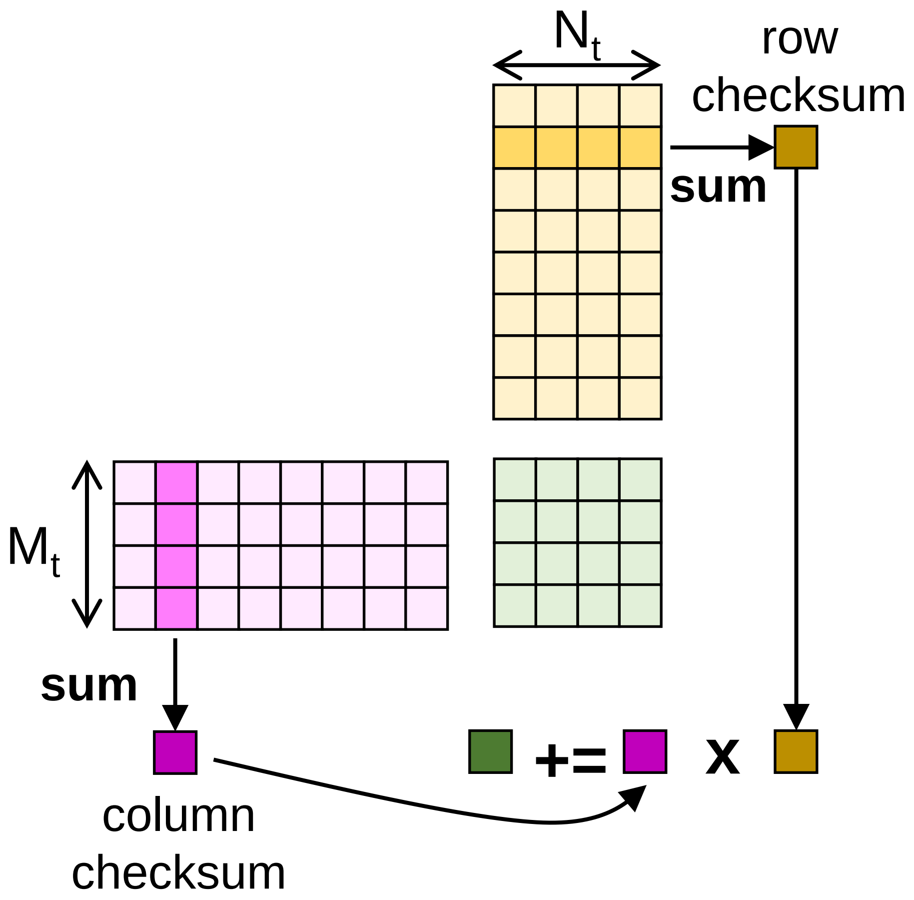




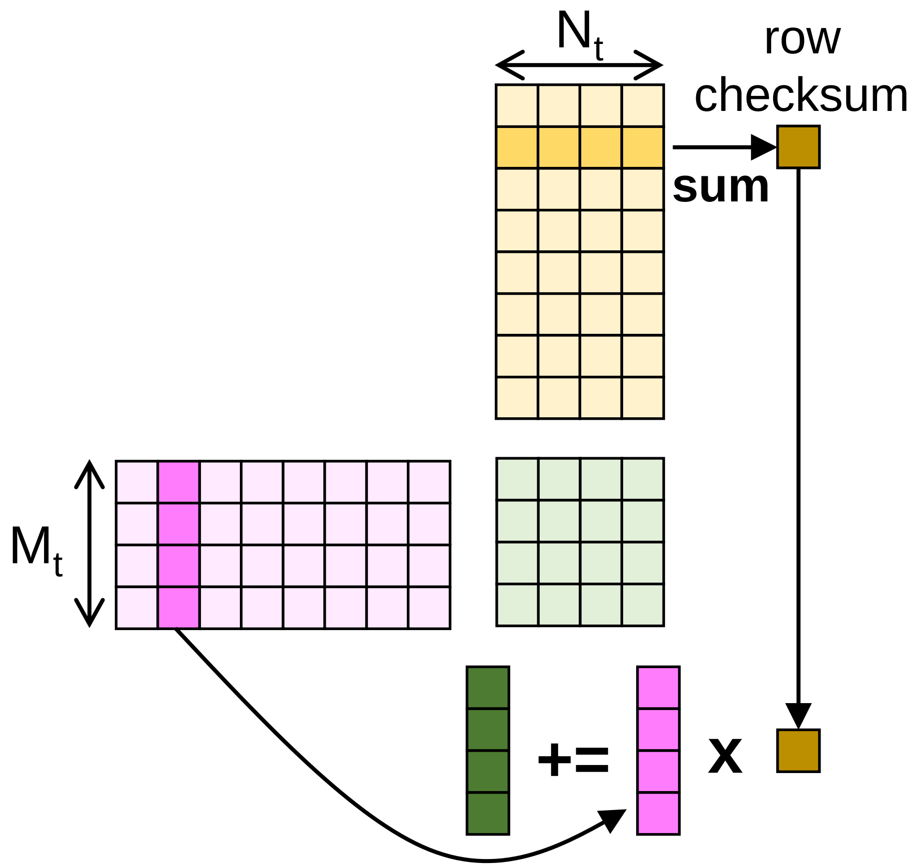
- *Thread-level ABFT:*
  - Each GPU thread performs thread-local ABFT alongside original mat. mult.
  - Avoids additional loads/stores
  - Adds more redundant operations, but exploits compute stalls in mat. mult.

Further exploiting underutilized computational bandwidth

Two-Sided Thread-Level ABFT



One-Sided Thread-Level ABFT



Arithmetic-intensity-guided ABFT

Key idea: adapt the type of fault detection used depending on bottleneck of layer

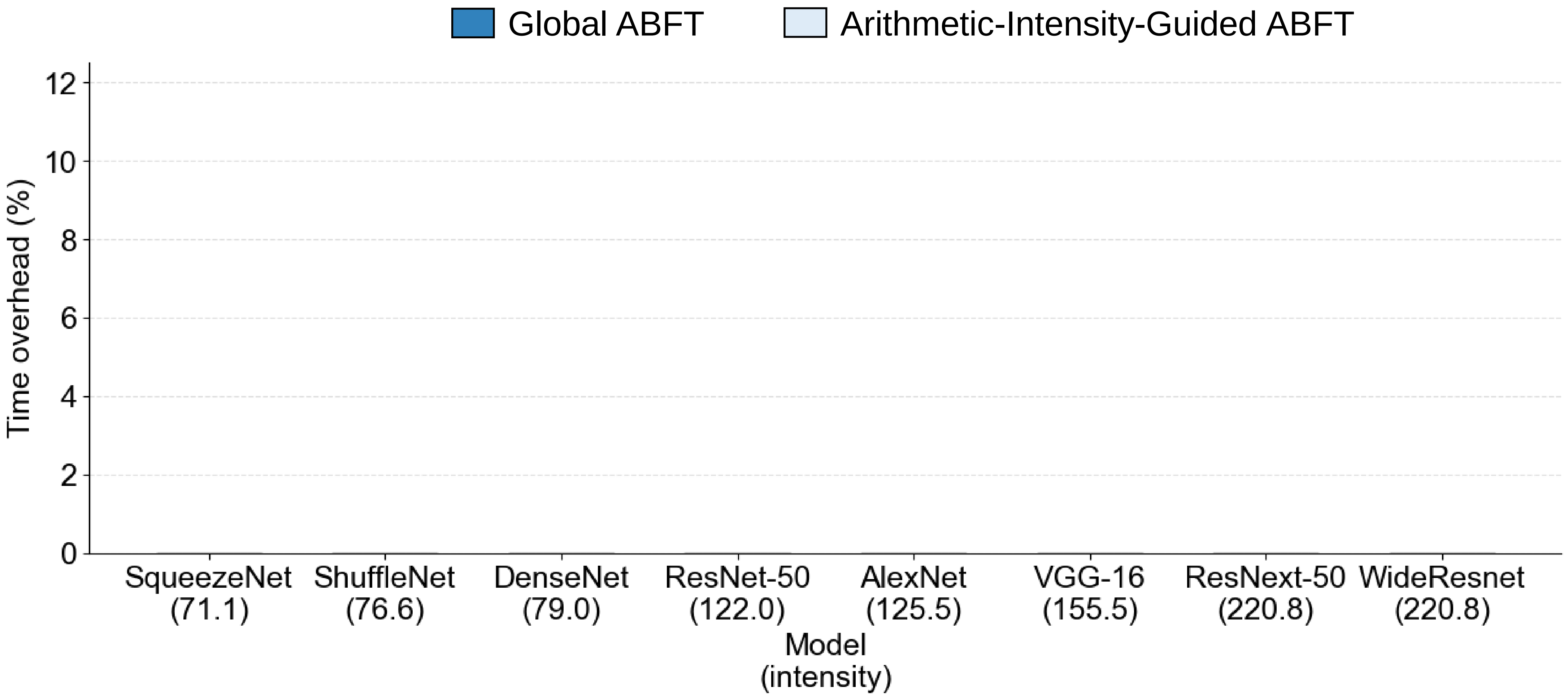
- Compute-bound layers: global ABFT
- Bandwidth-bound layers: thread-level ABFT

- Before deployment, for each linear layer:
  - Select fastest among global ABFT and thread-level ABFT
  - Choice typically aligns with intensity of layer and CMR of GPU
- More design decisions in the paper

Evaluation setup

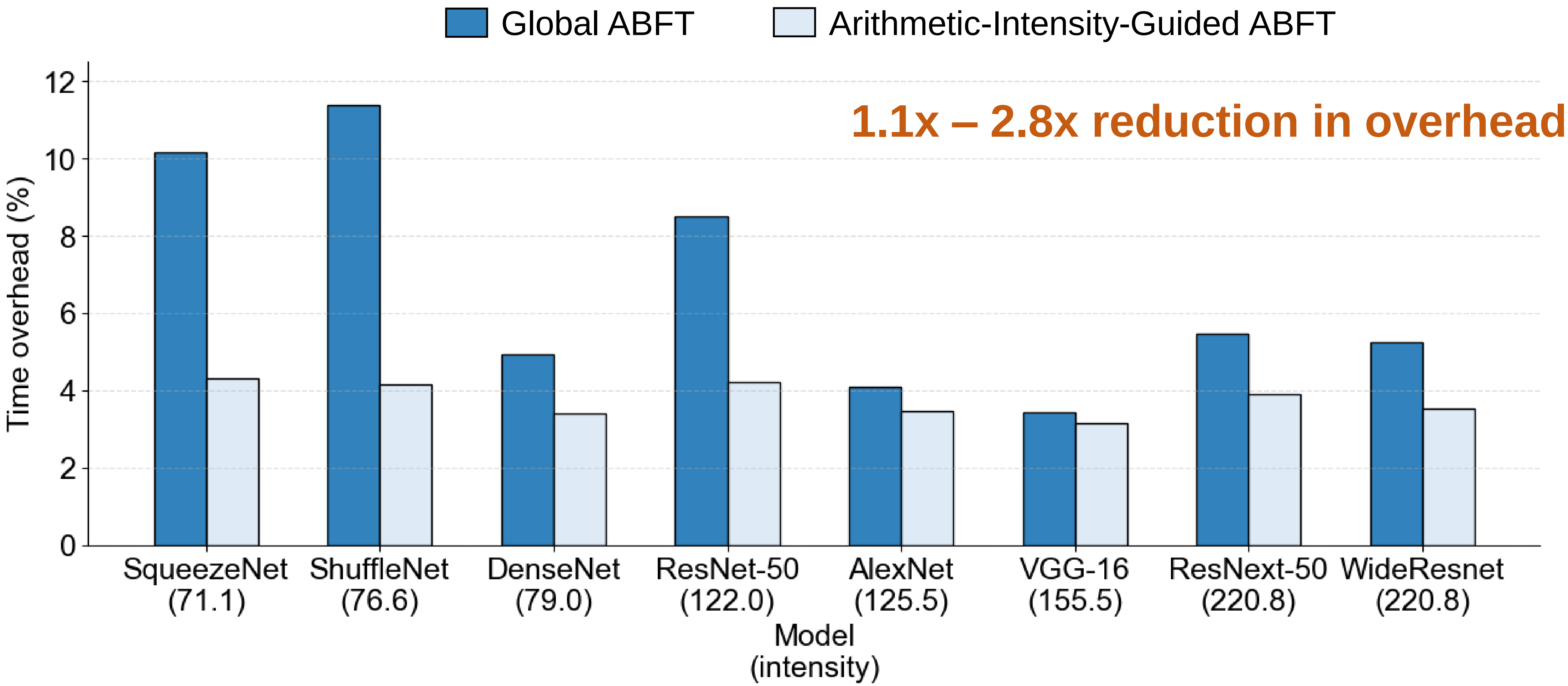
- Implemented in NVIDIA CUTLASS linear algebra library
- Run on T4 GPU, using Tensor Cores (FP16)
- Variety of neural network workloads
  - Popular CNNs
  - CNNs developed through model specialization
  - NNs in recommendation models (DLRMs)
- Detailed evaluation in paper:
  - Various batch sizes
  - Various image resolutions
  - Evaluation of design decisions in thread-level ABFT

Results: high-intensity neural networks



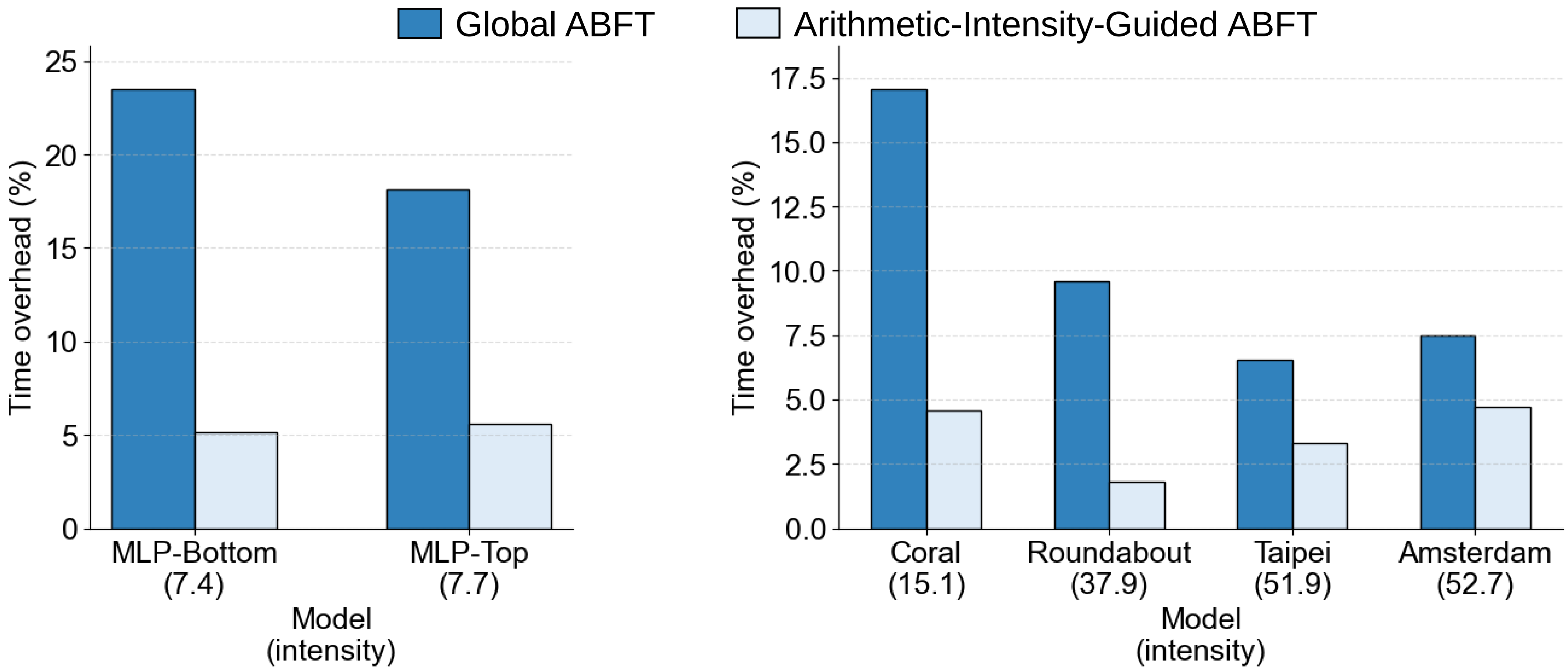
increasing intensity →

Results: high-intensity neural networks



increasing intensity →

Results: low-intensity neural networks



Recommendation models

3.2x – 4.6x speedup

Specialized CNNs

1.6x – 5.3x speedup

Summary of arithmetic-intensity-guided ABFT

- Analyze trends in neural network design and GPU hardware
- Made case for prevalence of bandwidth-bound linear layers
  - Prior approaches to ABFT are not well suited for these
- Propose *arithmetic-intensity-guided ABFT:*
  - Investigate approaches to ABFT for bandwidth-bound layers
  - Tailor the ABFT scheme used to the intensity of the layer, CMR of GPU
  - Enables 1.1x – 5.3x reduction in execution-time overhead

Code: github.com/thesys-lab/arithmetic-intensity-guided-abft

Contact: jkosaian@cs.cmu.edu, rvinayak@cs.cmu.edu