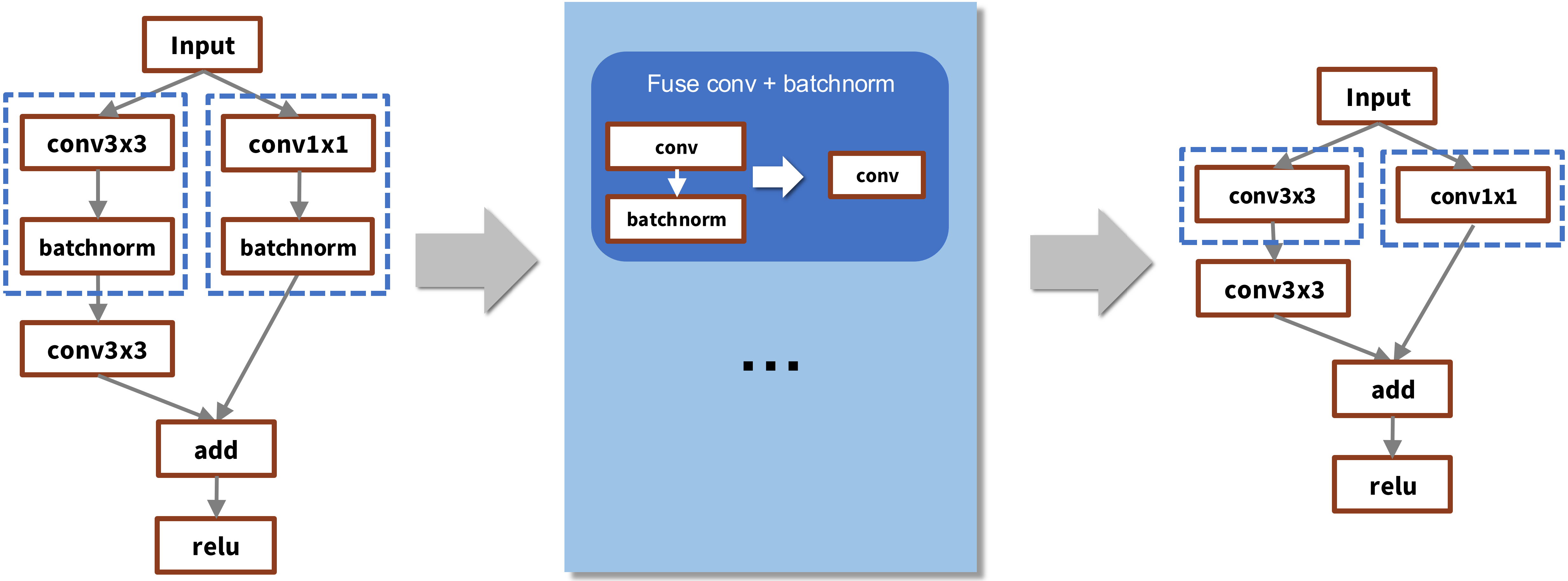
15-779 Lecture 11: ML Superoptimization

Zhihao Jia

Carnegie Mellon University

Last Lecture: Graph-Level Optimizations



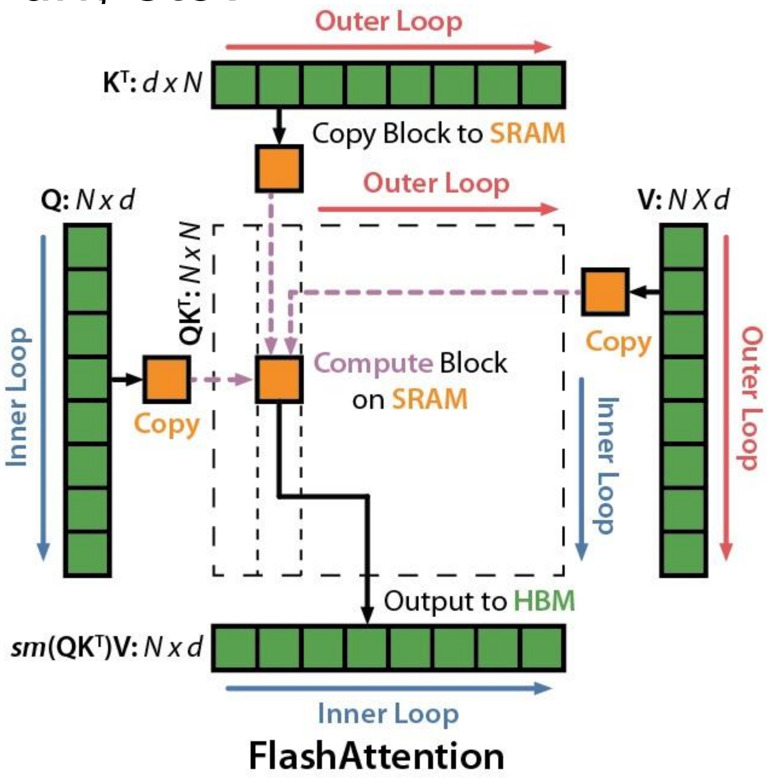
Input Computation Graph

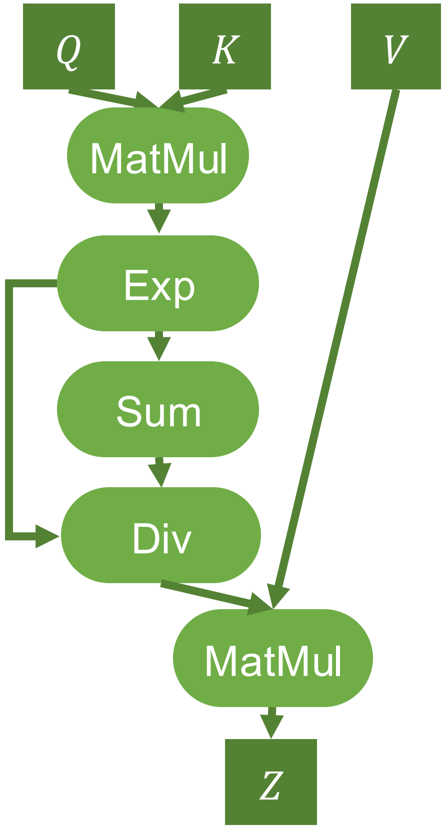
Potential graph transformations

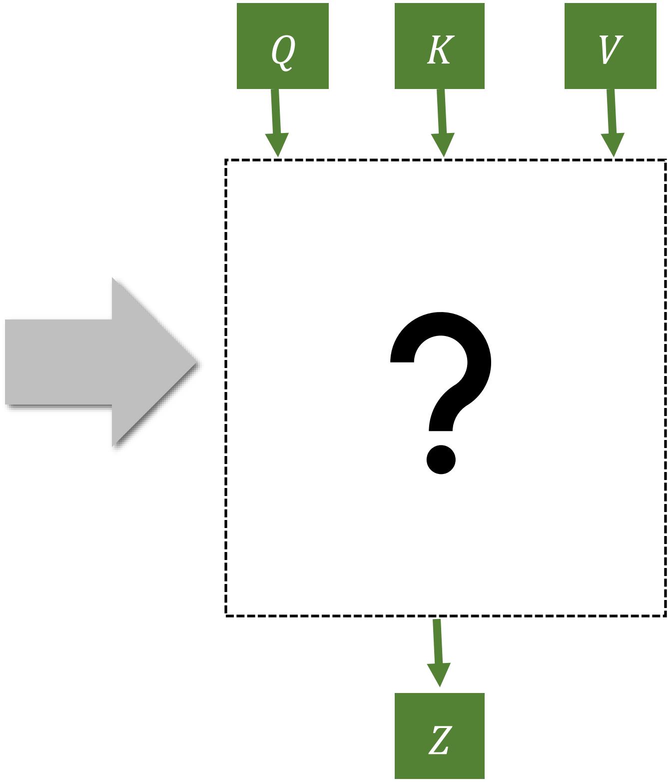
Optimized Computation Graph

Can We Represent FlashAttention as a Graph Optimization?

Is it possible to implement FlashAttention as a combination of matmul, exp, add, mul, div, etc?







No. Kernels load inputs from and write outputs to device memory

FlashAttention requires algebraic transformations + customized kernels + schedule transformations

This Lecture: ML Superoptimization

Optimize ML performance by simultaneously considering

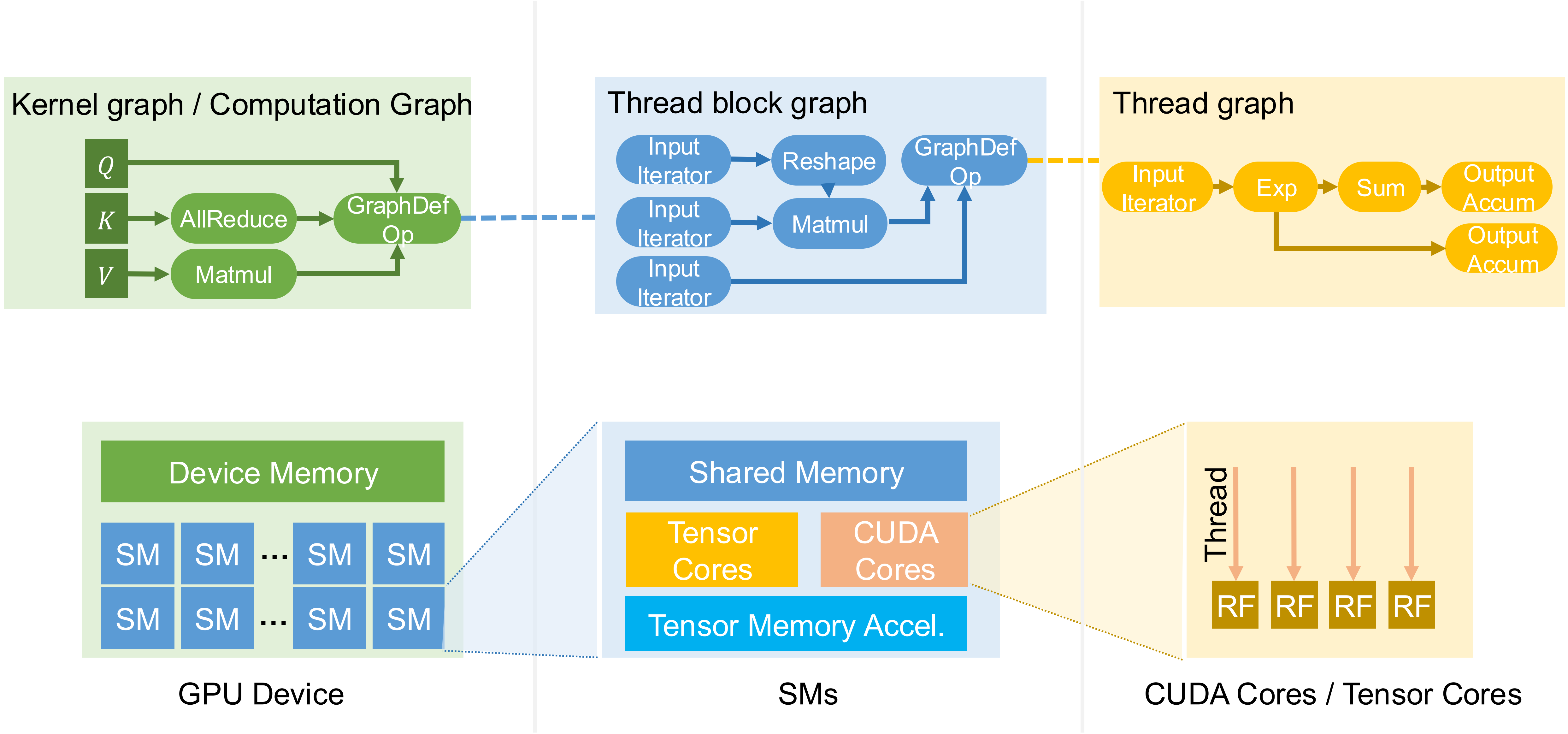
- Algebraic transformations
- Custom GPU kernels
- Schedule transformations

Mirage: A SuperOptimizer for ML

Key idea: automatically generate highly-optimized GPU kernels for DNNs

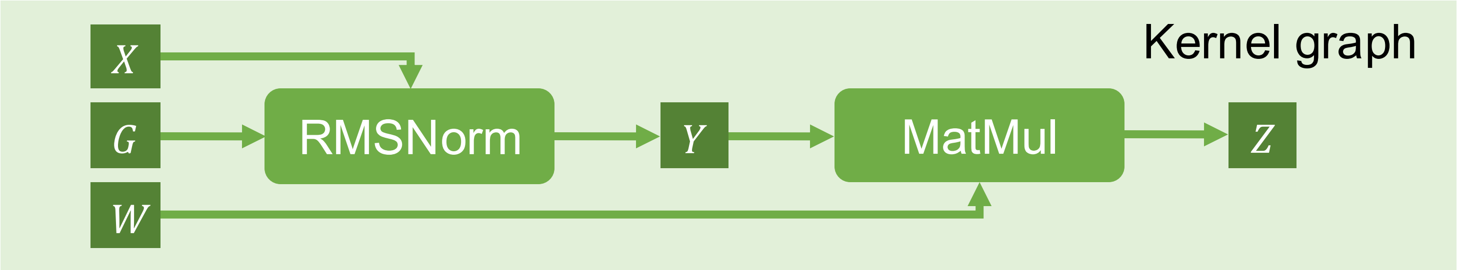
- **Less engineering effort**: thousands lines of CUDA code → a few lines of Python code in Mirage
- **Better performance**: outperform existing systems by 1.1-3.5x
- **Faster adaptation:** day-0 support for new models; no manual effort

μGraphs: Hierarchical Graph Representation



Example: RMSNorm & MatMul in LLMs

Existing systems launch two kernels since Y does not fit in shared memory

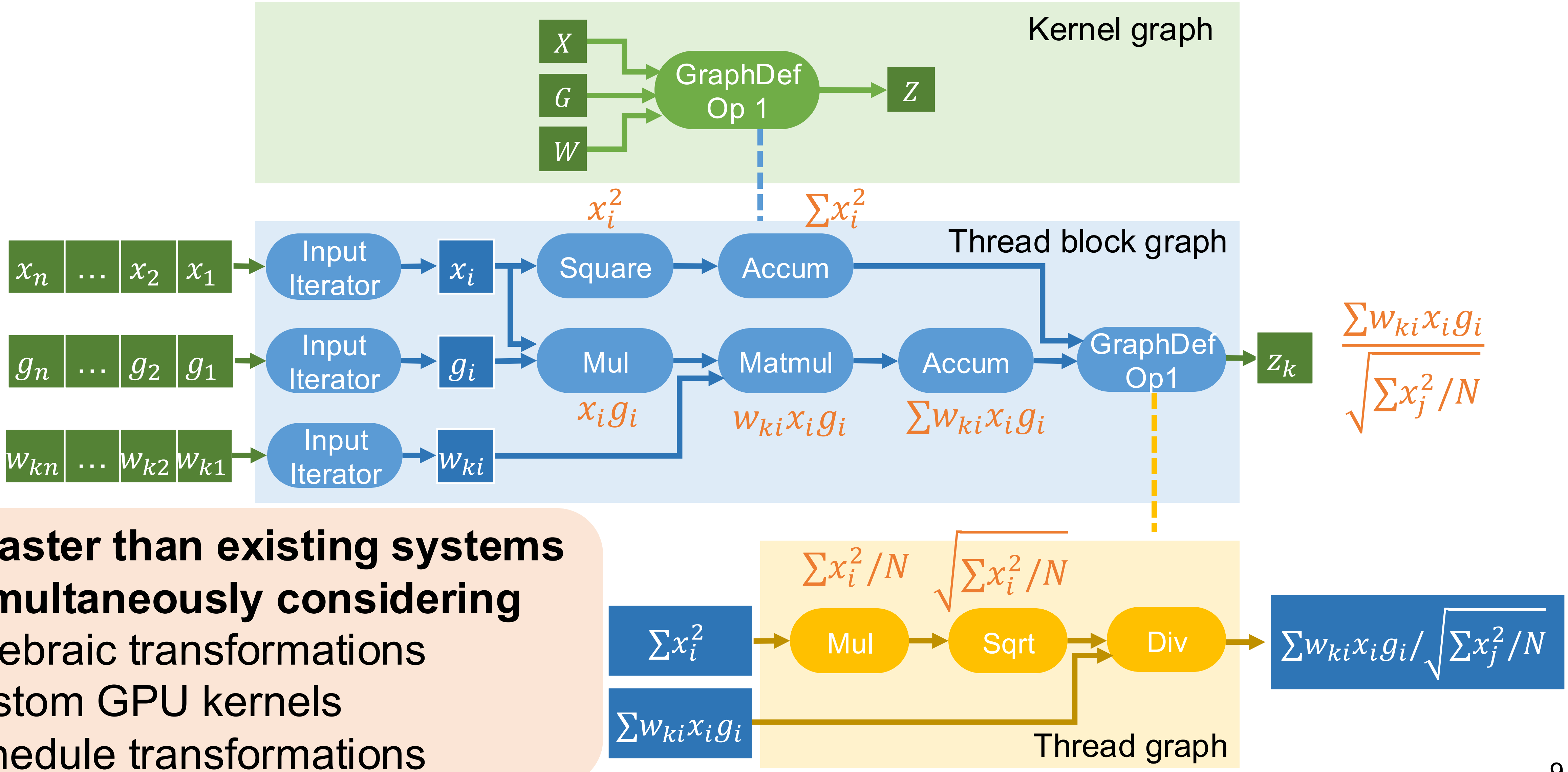


$$y_i = \frac{x_i g_i}{\sqrt{\frac{1}{N}\sum_j x_j^2}} \qquad z_i = \sum_k w_{ik} y_k$$

Performance issues:

1. No shared memory reuse
2. Kernel launch overhead

Best Discovered μGraph for RMSNorm & MatMul

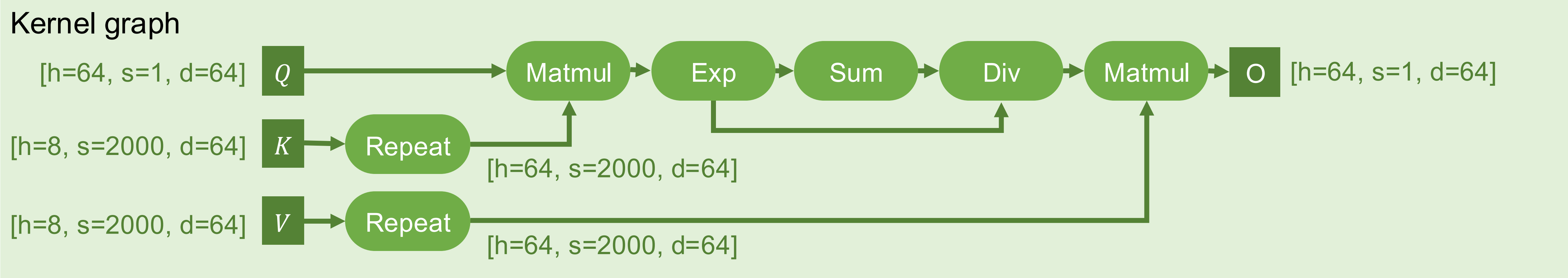


2.2x faster than existing systems by simultaneously considering

- Algebraic transformations
- Custom GPU kernels
- Schedule transformations

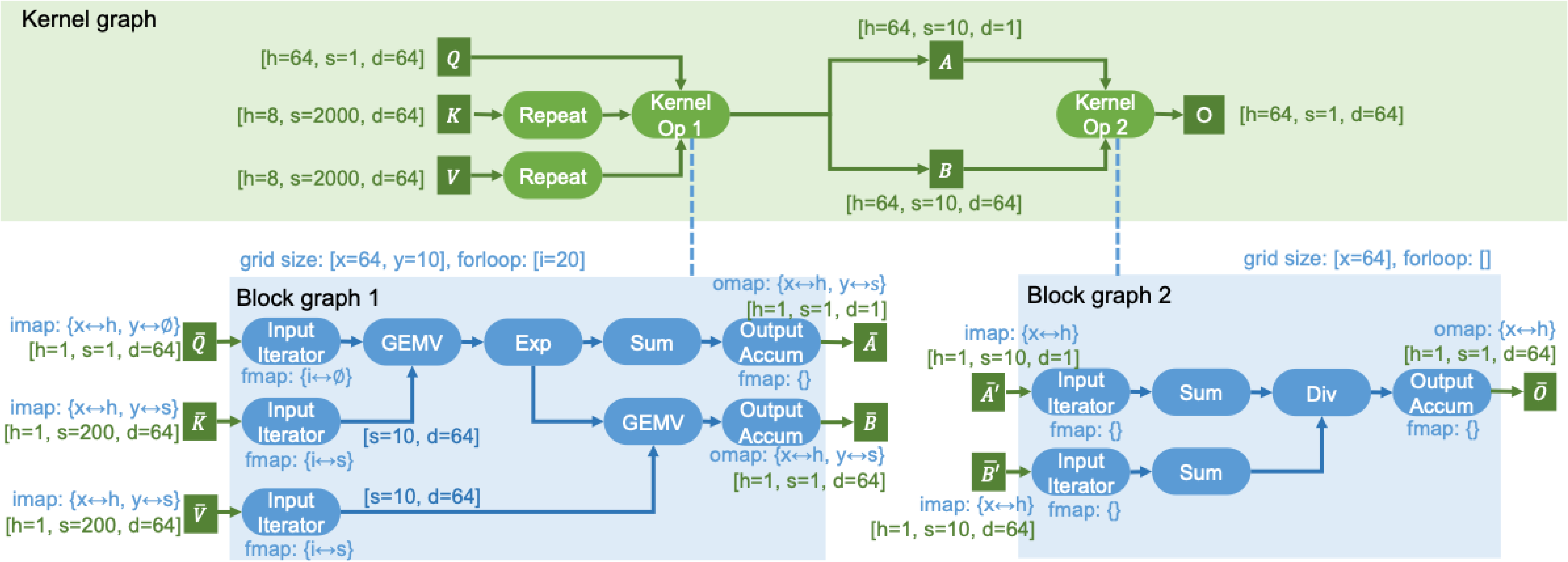
Example: Group-Query Attention in PyTorch

```
A = torch.matmul(Q, K.repeat(8))
S = torch.softmax(A)
O = torch.matmul(S, V.repeat(8))
```



```
A = torch.matmul(Q, K.repeat(8))
S = torch.softmax(A)
O = torch.matmul(S, V.repeat(8))
```

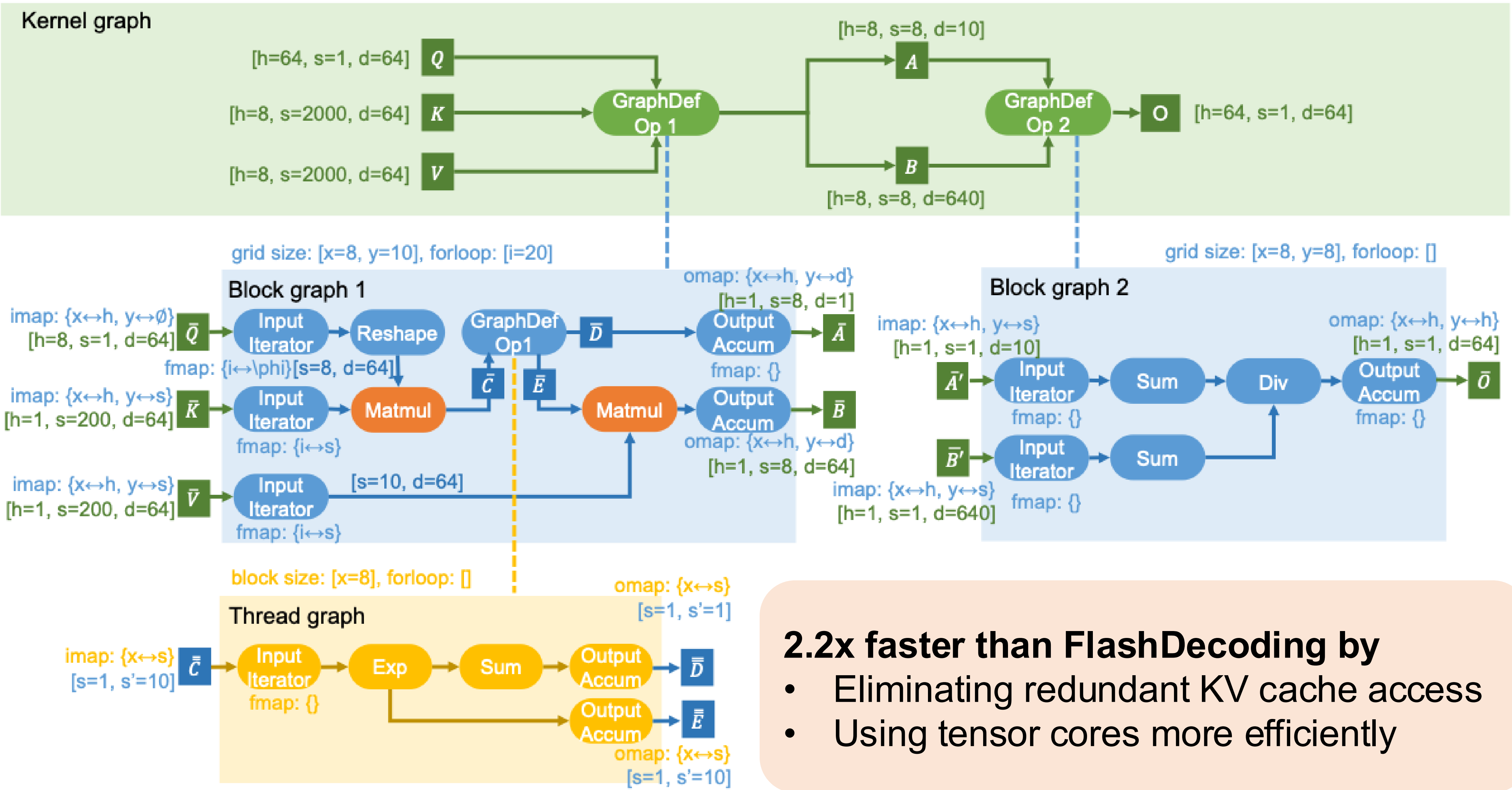
Group-Query Attention using FlashDecoding



Perform attention in two kernels by decomposing softmax into exp, 3 sums, and a div

Best μGraph for GQA

```
A = torch.matmul(Q, K.repeat(8))
S = torch.softmax(A)
O = torch.matmul(S, V.repeat(8))
```



2.2x faster than FlashDecoding by

- Eliminating redundant KV cache access
- Using tensor cores more efficiently

Key Challenges to Discover High-Performance μGraphs

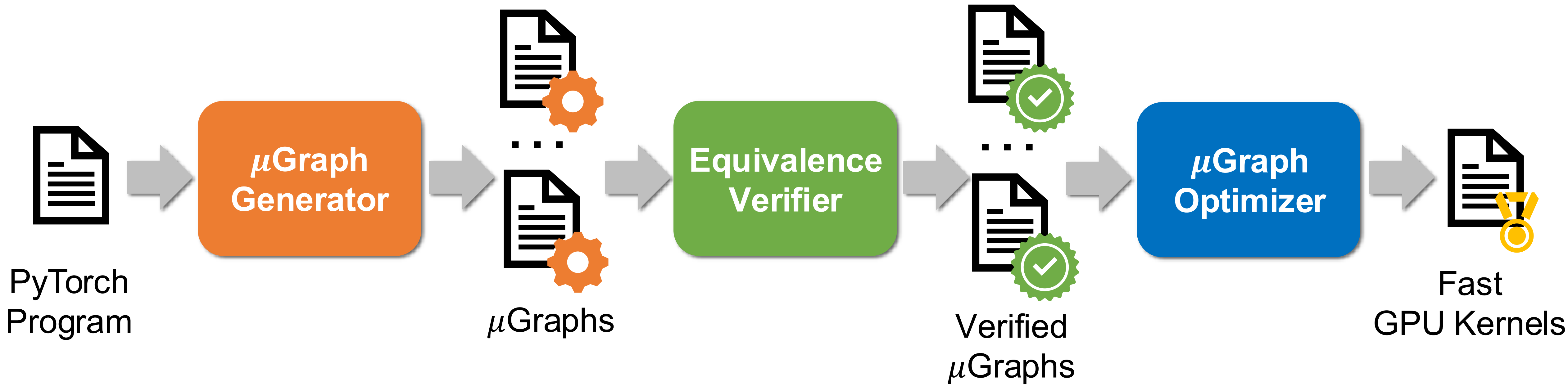
1. How to generate potential μGraphs?

Expression-guided search

2. How to verify their correctness?

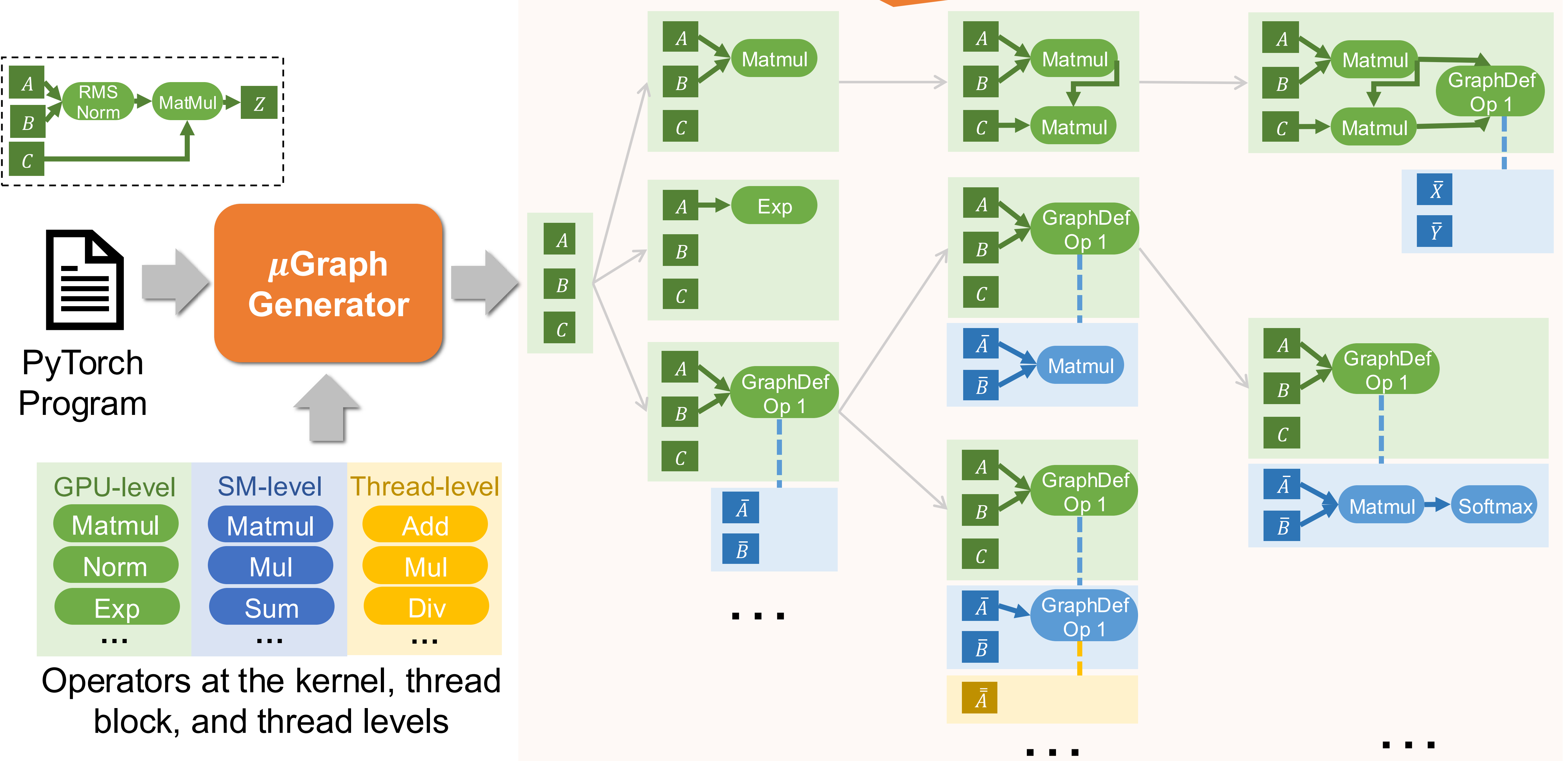
Probabilistic verification

Mirage Overview

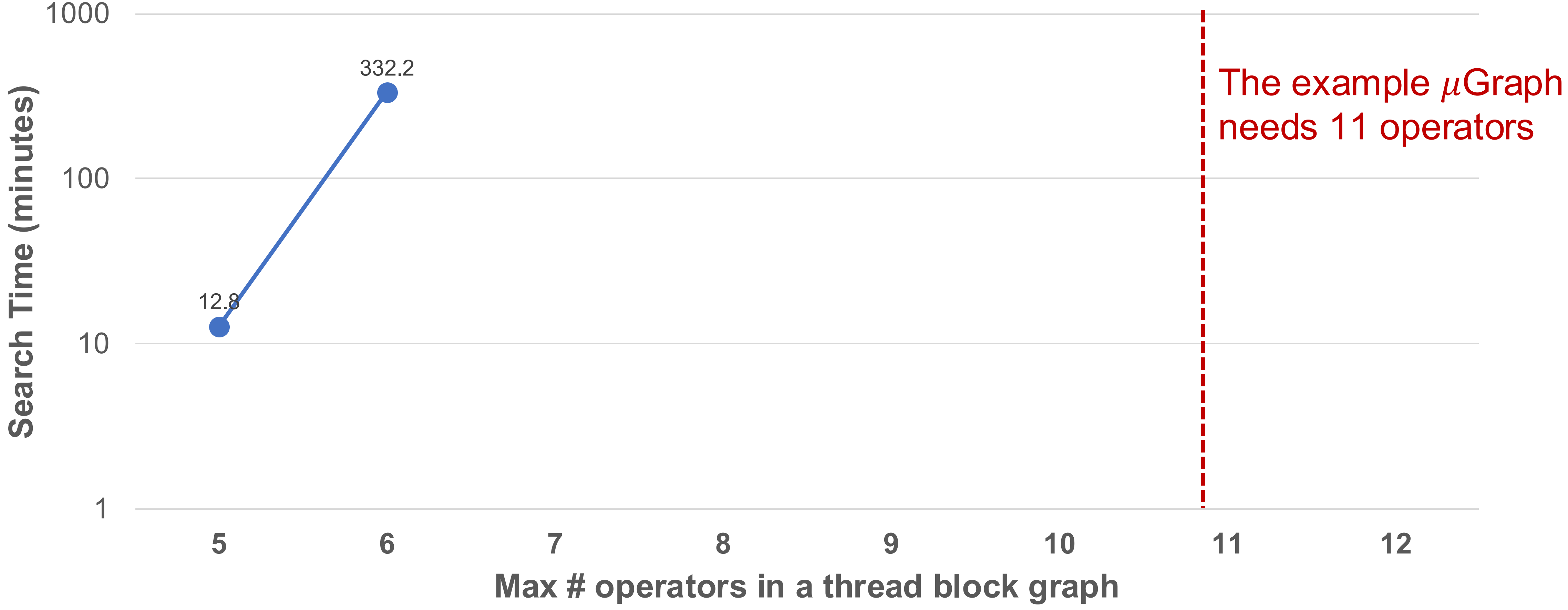


μGraph Generator

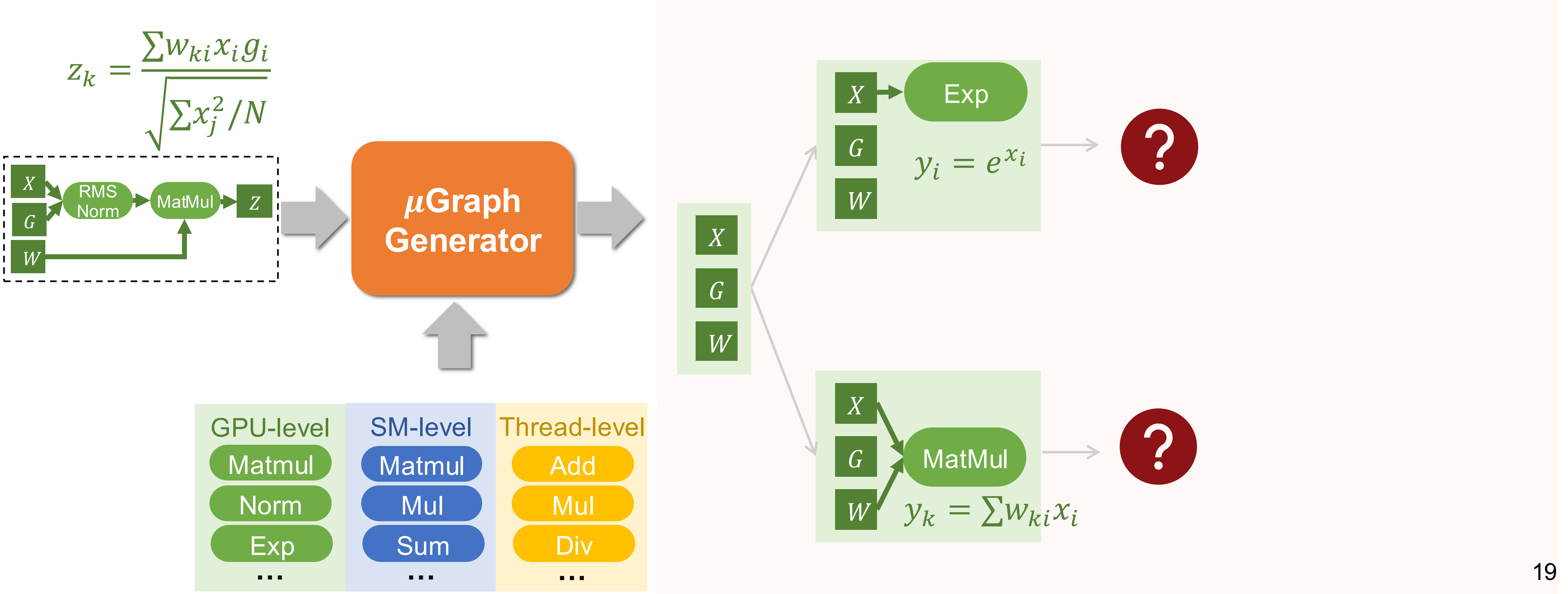
Consider all possible μGraphs using available operators



Challenge: Extremely Large Search Space

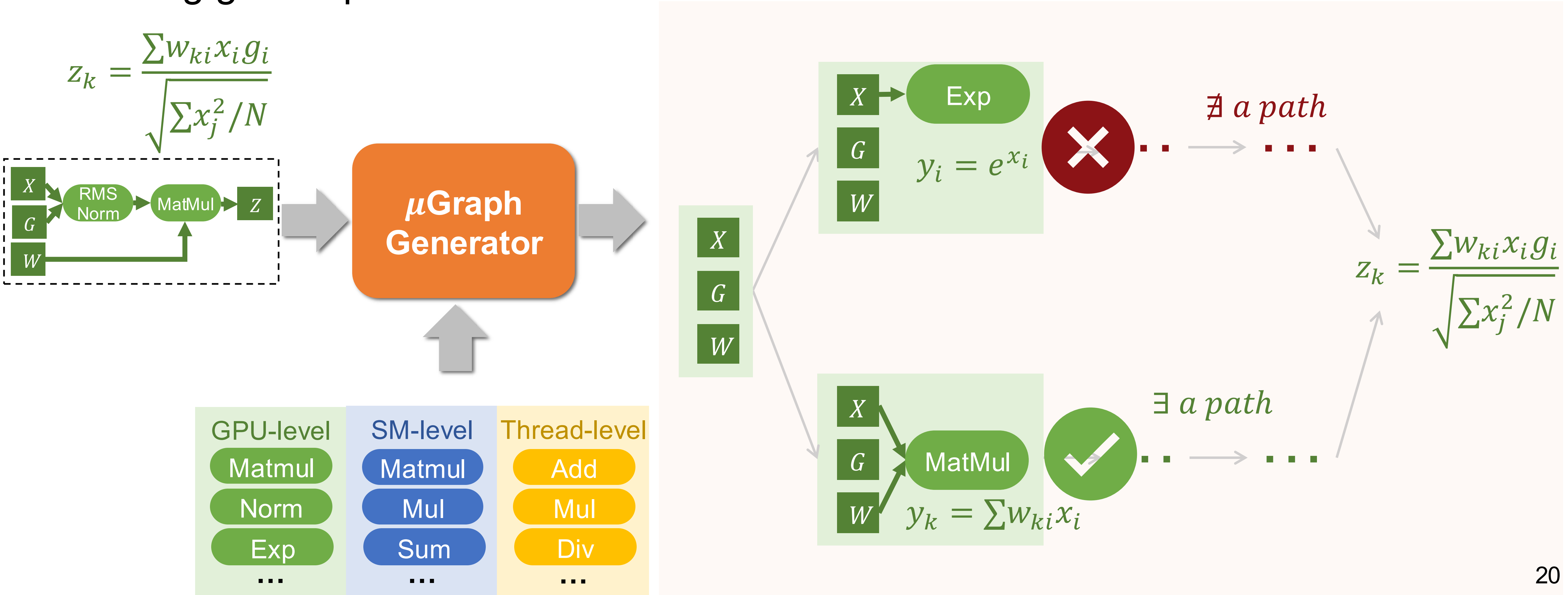


Can Algebraic Properties Guide Search?



Can Algebraic Properties Guide Search?

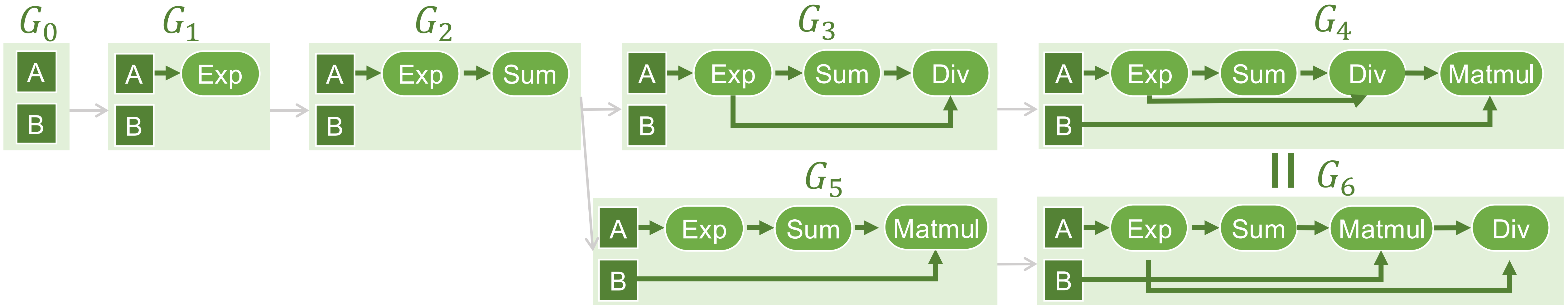
Key idea: prune candidates that do not have a path to desired computation using given operators



Abstract Expression

Goal #1. capture subgraph relations: $E(G_0) \preccurlyeq E(G_1) \preccurlyeq E(G_2) \preccurlyeq E(G_3) \preccurlyeq E(G_4)$

Goal #2. capture graph equivalence: $E(G_4) = E(G_6)$



$\exists$ a path from G to a μGraph G' equivalent to the input G_0?

$$G \longrightarrow \cdots \longrightarrow \cdots \longrightarrow G' = G_0$$

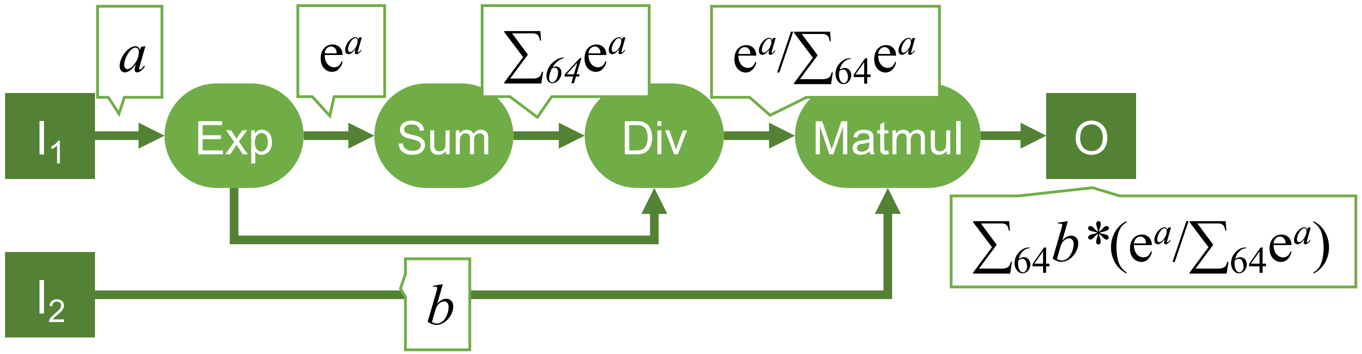
$E(G) \preccurlyeq E(G_0)$?

Abstract Expression: An Implementation

Represent a tensor's computation by abstracting away index details

- E.g., $\mathrm{C} = A \times B$: $c_{ij} = \sum_{k=1}^{64} a_{ik}\, b_{kj}$ ⇒ abstract expression is $c = \sum_{64} ab$

Recursively compute abstract expressions



Abstract Expression

Mirage uses first-order logic to reason about two relations

Goal 1: subexpression

Subexpression Axioms A_{sub}

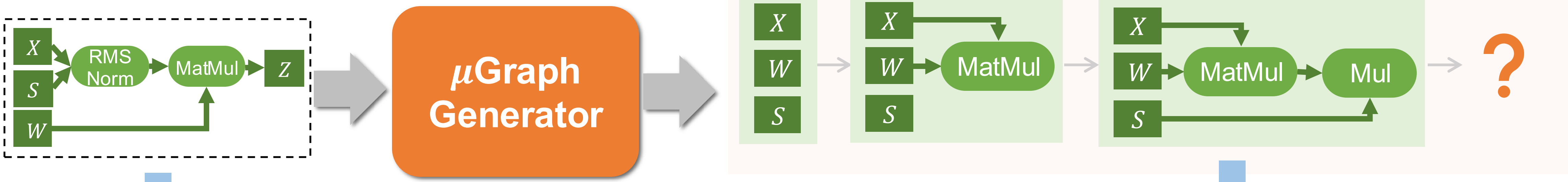
$\forall x, y.\ \mathsf{subexpr}(x, \mathsf{add}(x, y))$	
$\forall x, y.\ \mathsf{subexpr}(x, \mathsf{mul}(x, y))$	
$\forall x, y.\ \mathsf{subexpr}(x, \mathsf{div}(x, y))$	
$\forall x, y.\ \mathsf{subexpr}(y, \mathsf{div}(x, y))$	
$\forall x.\ \mathsf{subexpr}(x, \mathsf{exp}(x))$	
$\forall x, i.\ \mathsf{subexpr}(x, \mathsf{sum}(i, x))$	
$\forall x.\ \mathsf{subexpr}(x, x)$	reflexivity
$\forall x, y, z.\ \mathsf{subexpr}(x, y) \land \mathsf{subexpr}(y, z) \rightarrow \mathsf{subexpr}(x, z)$	transitivity

Goal 2: equivalence

Equivalence Axioms A_{eq}

$\forall x, y.\ \mathsf{add}(x, y) = \mathsf{add}(y, x)$	commutativity
$\forall x, y.\ \mathsf{mul}(x, y) = \mathsf{mul}(y, x)$	commutativity
$\forall x, y, z.\ \mathsf{add}(x, \mathsf{add}(y, z)) = \mathsf{add}(\mathsf{add}(x, y), z)$	associativity
$\forall x, y, z.\ \mathsf{mul}(x, \mathsf{mul}(y, z)) = \mathsf{mul}(\mathsf{mul}(x, y), z)$	associativity
$\forall x, y, z.\ \mathsf{add}(\mathsf{mul}(x, z), \mathsf{mul}(y, z)) = \mathsf{mul}(\mathsf{add}(x, y), z)$	distributivity
$\forall x, y, z.\ \mathsf{add}(\mathsf{div}(x, z), \mathsf{div}(y, z)) = \mathsf{div}(\mathsf{add}(x, y), z)$	associativity
$\forall x, y, z.\ \mathsf{mul}(x, \mathsf{div}(y, z)) = \mathsf{div}(\mathsf{mul}(x, y), z)$	associativity
$\forall x, y, z.\ \mathsf{div}(\mathsf{div}(x, y), z) = \mathsf{div}(x, \mathsf{mul}(y, z))$	associativity
$\forall x.\ x = \mathsf{sum}(1, x)$	identity reduction
$\forall x, i, j.\ \mathsf{sum}(i, \mathsf{sum}(j, x)) = \mathsf{sum}(i * j, x)$	associativity
$\forall x, y, i.\ \mathsf{sum}(i, \mathsf{add}(x, y)) = \mathsf{add}(\mathsf{sum}(i, x), \mathsf{sum}(i, y))$	associativity
$\forall x, y, i.\ \mathsf{sum}(i, \mathsf{mul}(x, y)) = \mathsf{mul}(\mathsf{sum}(i, x), y)$	distributivity
$\forall x, y, i.\ \mathsf{sum}(i, \mathsf{div}(x, y)) = \mathsf{div}(\mathsf{sum}(i, x), y)$	distributivity

Abstract Expression-Guided Search



$$E(G_0) = sum\Big(mul\Big(div\Big(mul(x,s), sqrt\big(sum(x)\big)\Big), w\Big)\Big)$$

$$E(G) = mul\big(sum\big(mul(x,w)\big), s\big)$$

A1. $\forall x, y.\, x \preccurlyeq mul(x,y)$
A2. $\forall x, i.\, x \preccurlyeq sum(i,x)$
A3. …

Subexpression axioms

E1. $\forall x, y, z.$
$mul\big(x, div(y,z)\big) = div(mul(x,y), z)$
E2. …

Equivalence axioms

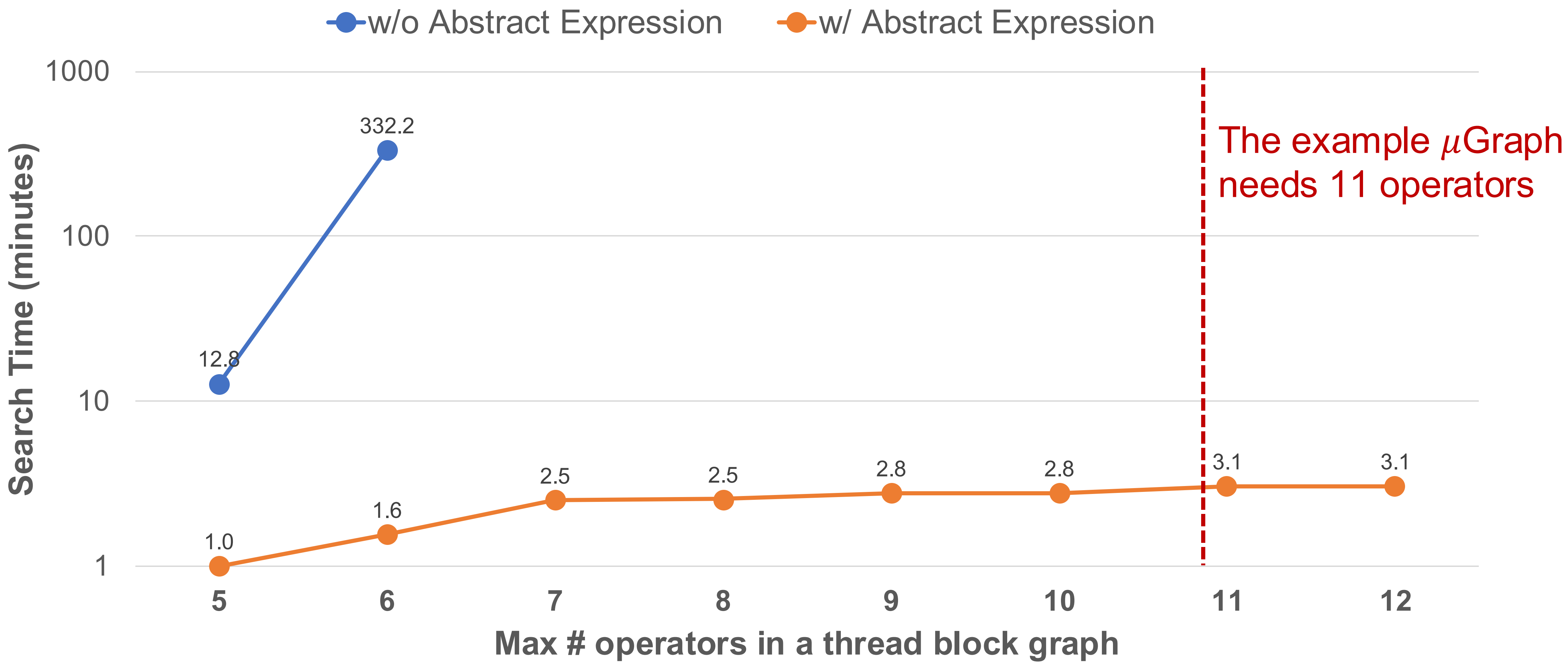
$E(G) \preccurlyeq E(G_0)$?

Automated Theorem Prover

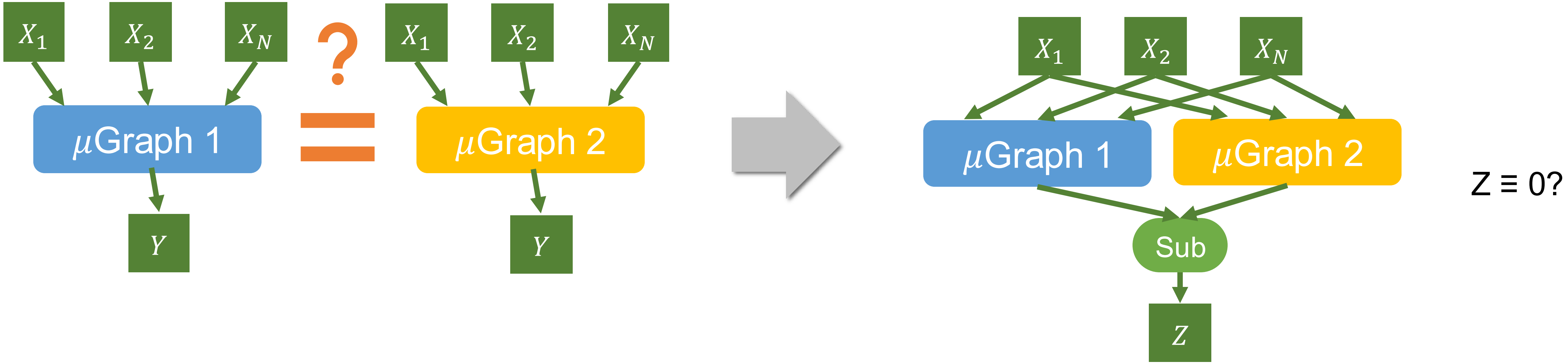
continue the search

prune candidate μGraph

Abstract Expression Significantly Improves Scalability



How to Verify Equivalence between μGraphs?

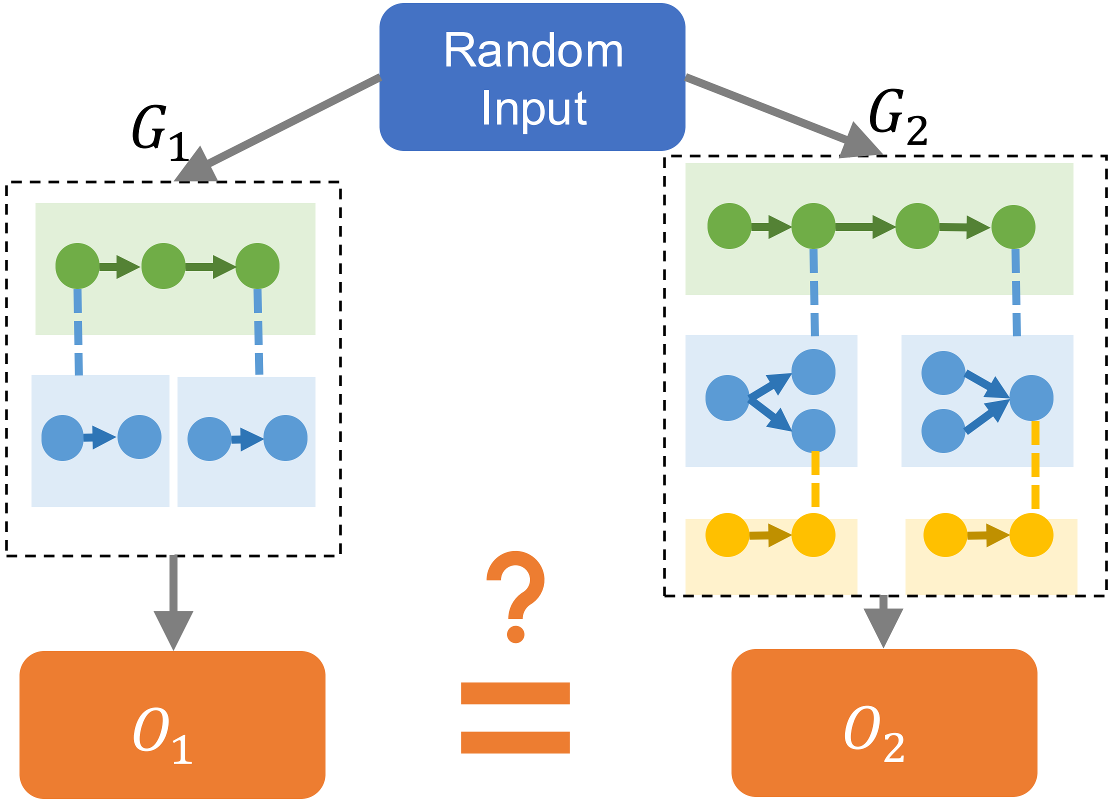


Schwartz–Zippel lemma: Let $P \in F[X_1, X_2, .., X_n]$ be a non-zero polynomial of total degree d over a finite field F. Let $r_1, r_2, .., r_n$ be selected randomly and uniformly from F. Then

$$\Pr[P(r_1, r_2, \ldots, r_n) = 0] \leq \frac{d}{|F|}$$

Probabilistic Equivalence Verifier

Idea: use random inputs in ***finite fields*** to examine μGraph equivalence



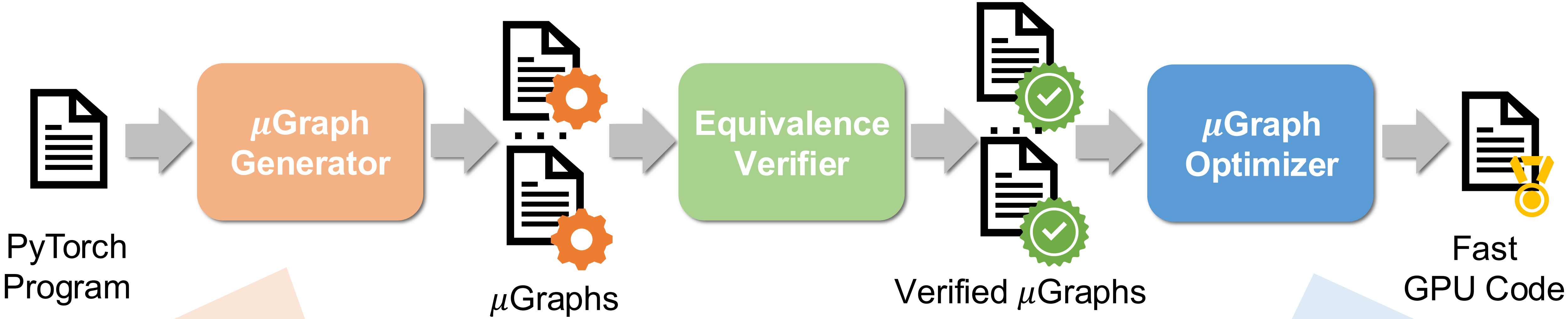
Theorem 1: if G_1 is equivalent to G_2, then $O_1 = O_2$

Theorem 2: if G_1 is not equivalent to G_2, then $O_1 \neq O_2$ with a certain probability p^*

Run t random tests, non-equivalent μGraphs pass all random tests with probability $(1-p)^t$

* $p \geq \frac{1}{T}$, where T is the size of intermediate tensors

μGraph Optimizer



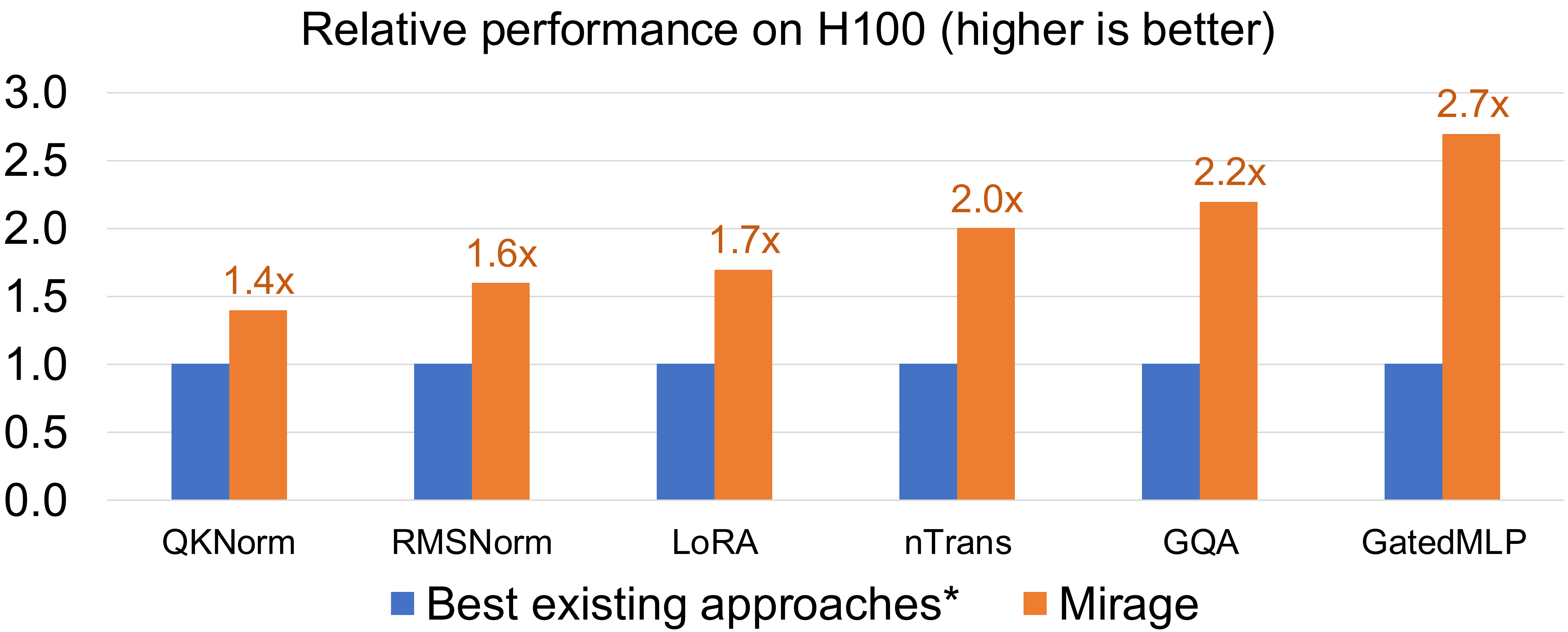
Only consider output-alternating optimizations:
- Algebraic transformations
- Customized kernels
- Compute organization

Reduce generator's search space

Other optimizations are deferred to μGraph Optimizer:
- Tensor layouts
- Memory planning
- Operator scheduling

Solve these tasks optimally

Mirage Outperforms Existing Approaches

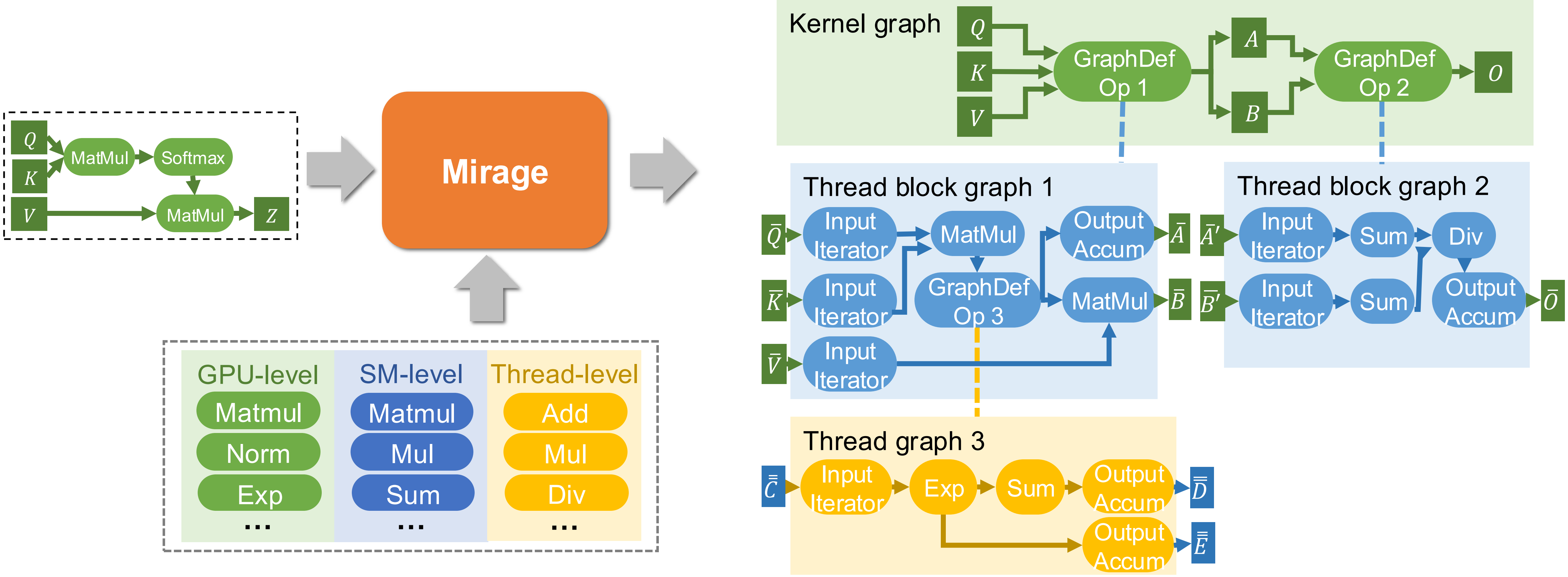


Vender-provided Expert-written Compiler-generated

*Best existing approaches are the best of cuDNN/cuBLAS, FlashAttention, PyTorch, TensorRT, Triton, and TVM.

Mirage Discovers Hardware-Customized μGraphs

Find μGraphs similar to expert-written implementations for attention on A100

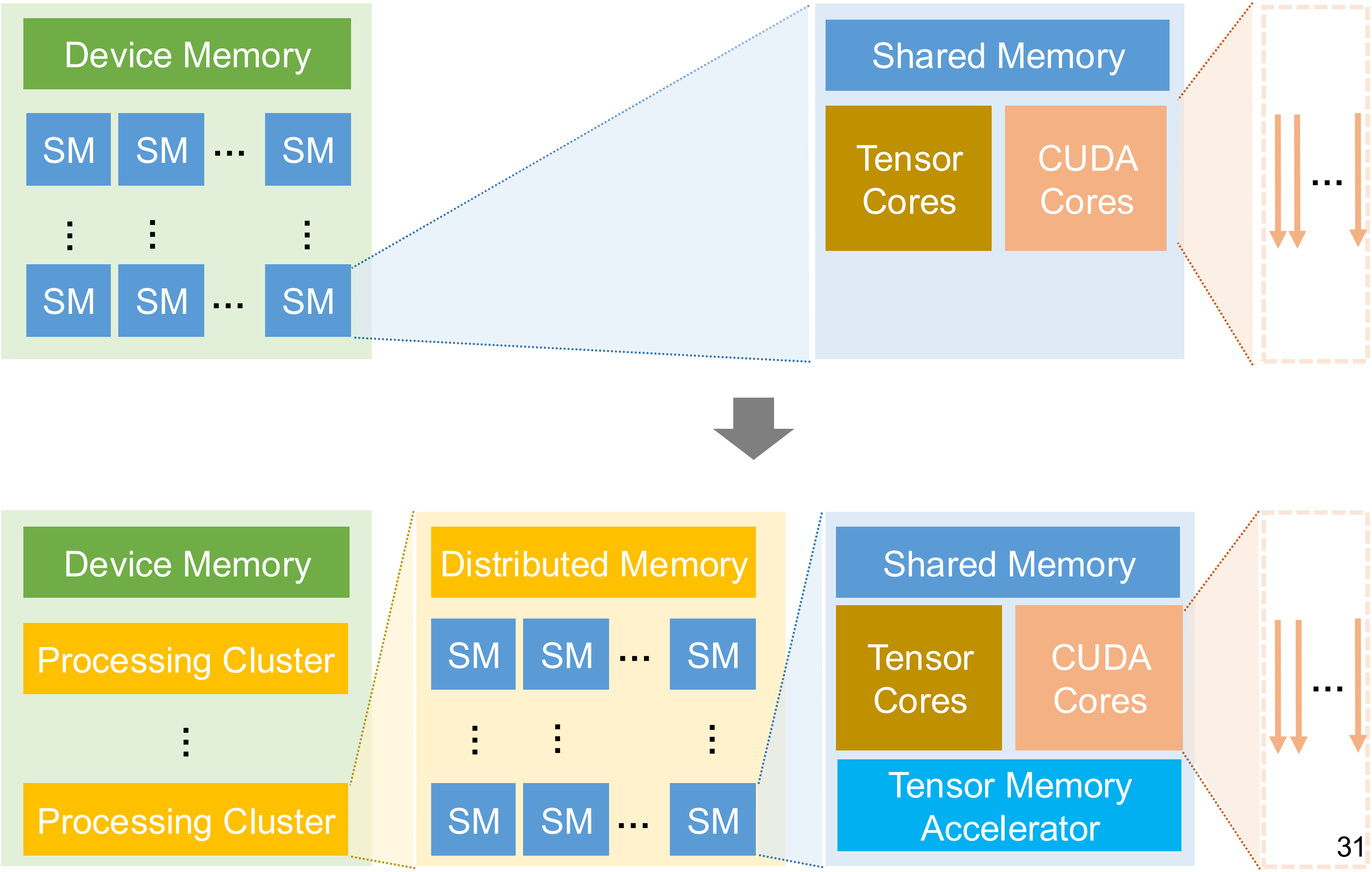


* FlashDecoding++: Faster LLM Inference with Asynchronization, Flat GEMM Optimization, and Heuristics. MLSys'24

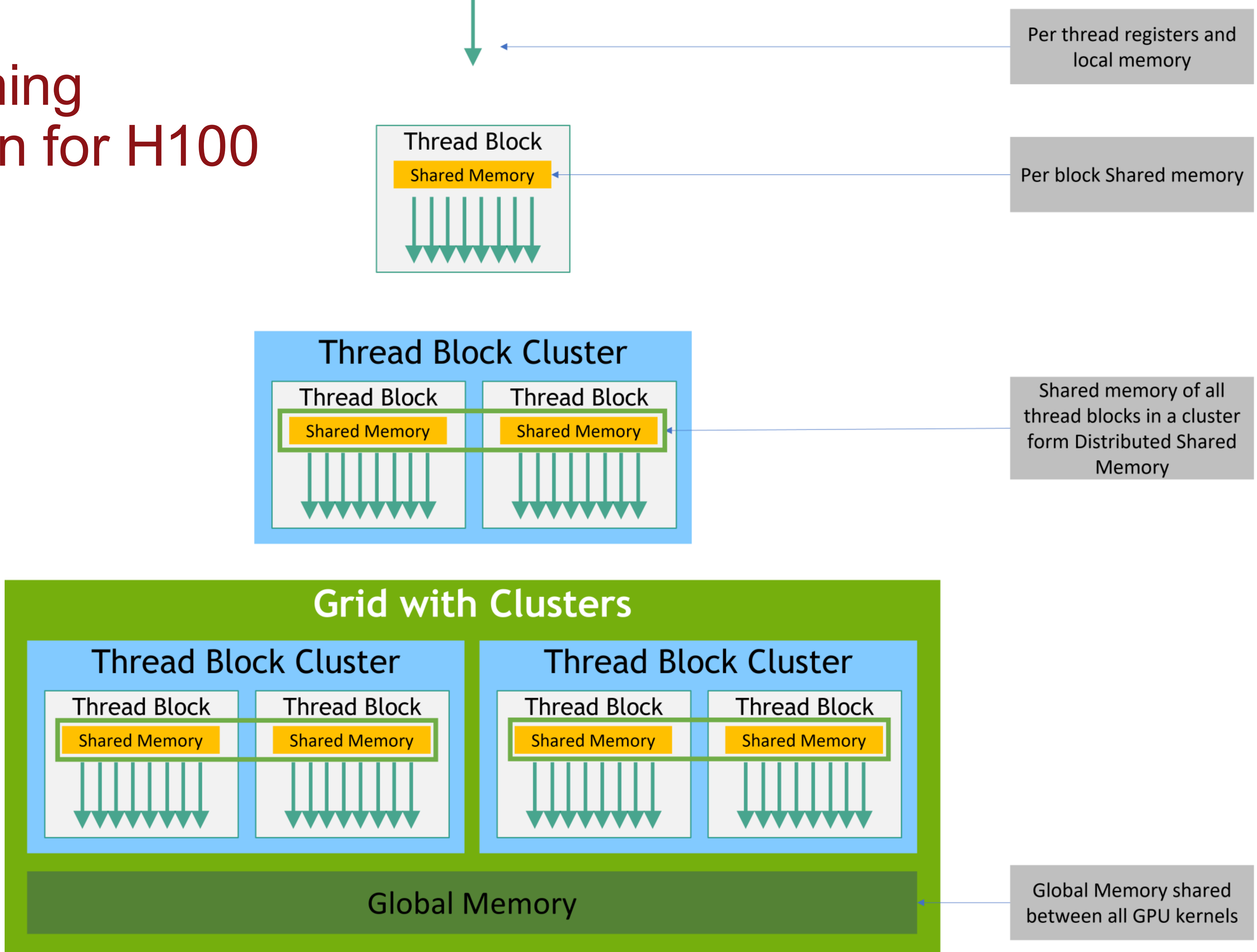
H100: GPU Processing Clusters

NVIDIA A100 GPU (2020)

NVIDIA H100 GPU (2022)



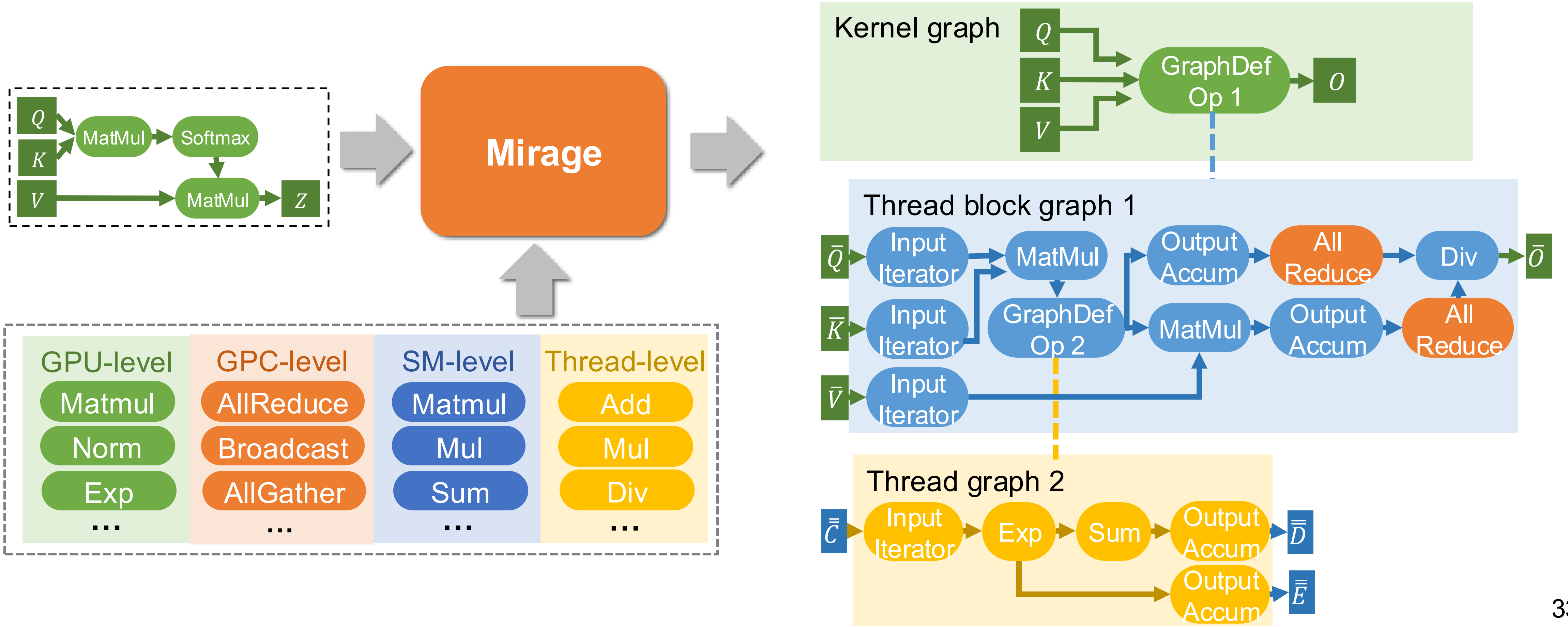
Programming Abstraction for H100



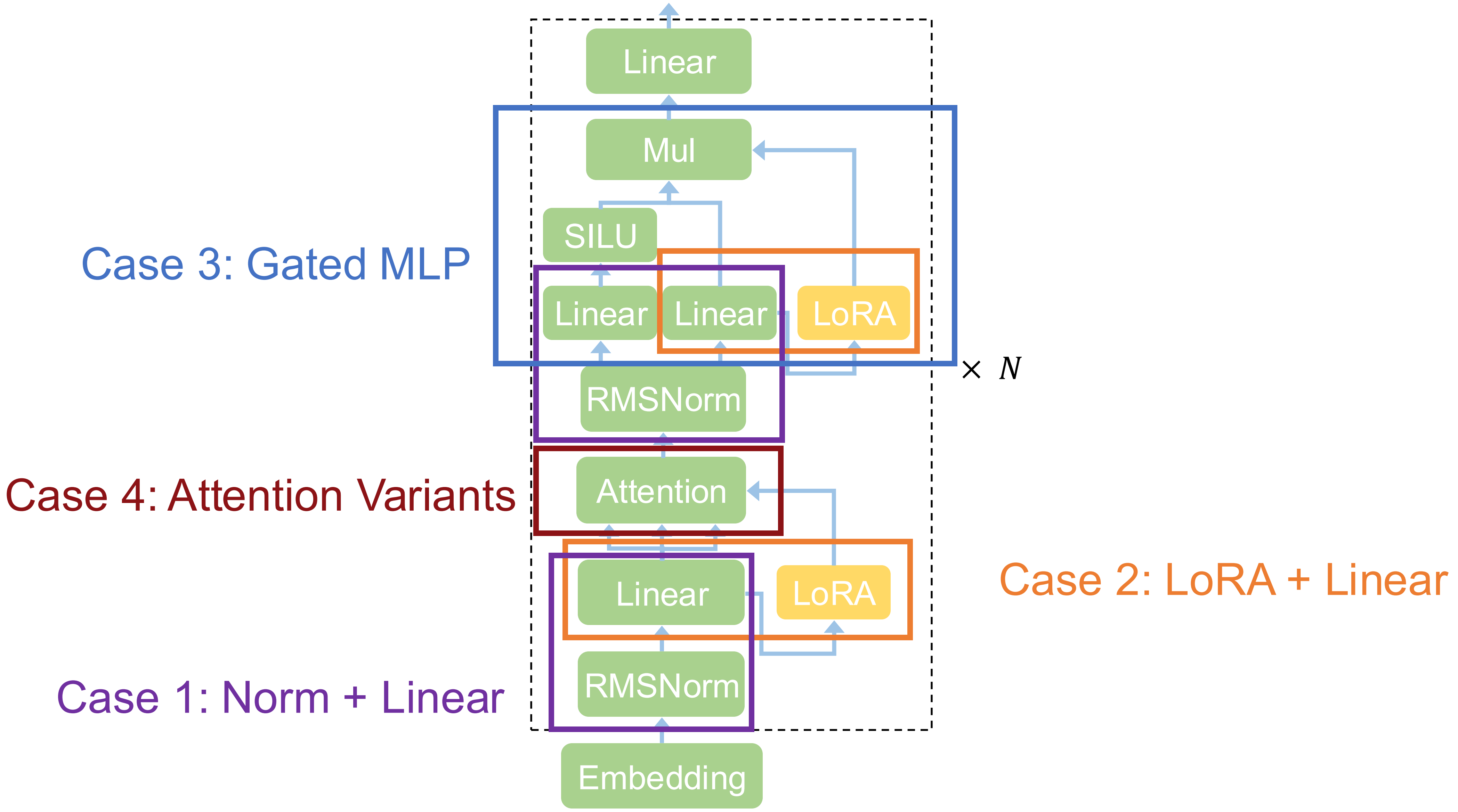
Mirage Discovers Hardware-Customized μGraphs

Leverage GPC-level AllReduce to accelerate attention on H100

- 2.2x faster than best existing kernels

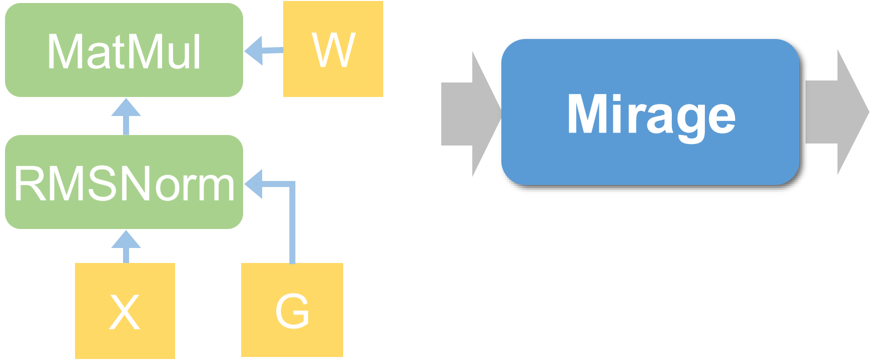


Ubiquitous Superoptimization Opportunities in DNNs

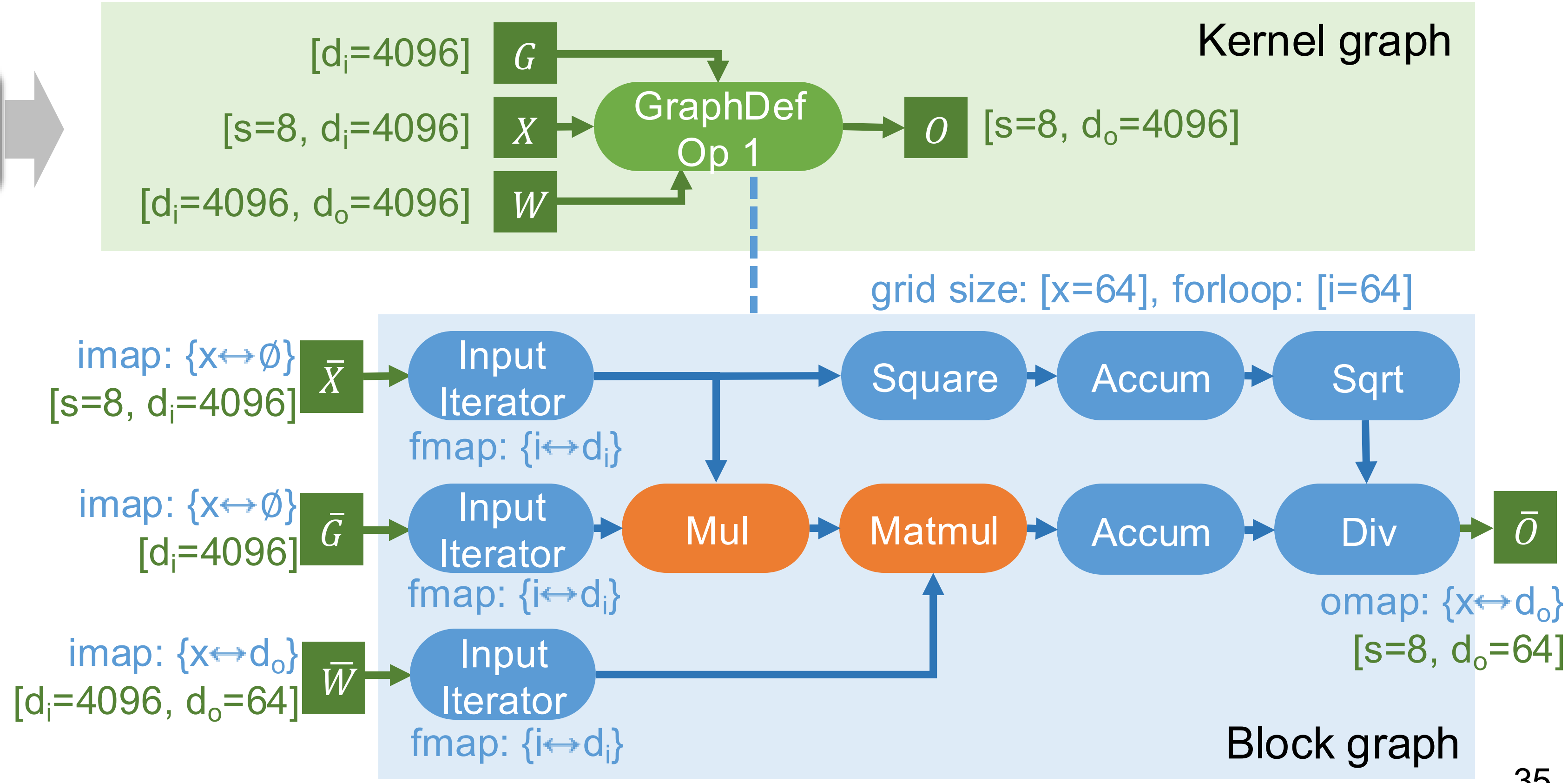


Case 1: RMSNorm + Linear

- Key idea: move the denominator of RMSNorm after matrix multiplication to fuse kernels and reuse shared memory

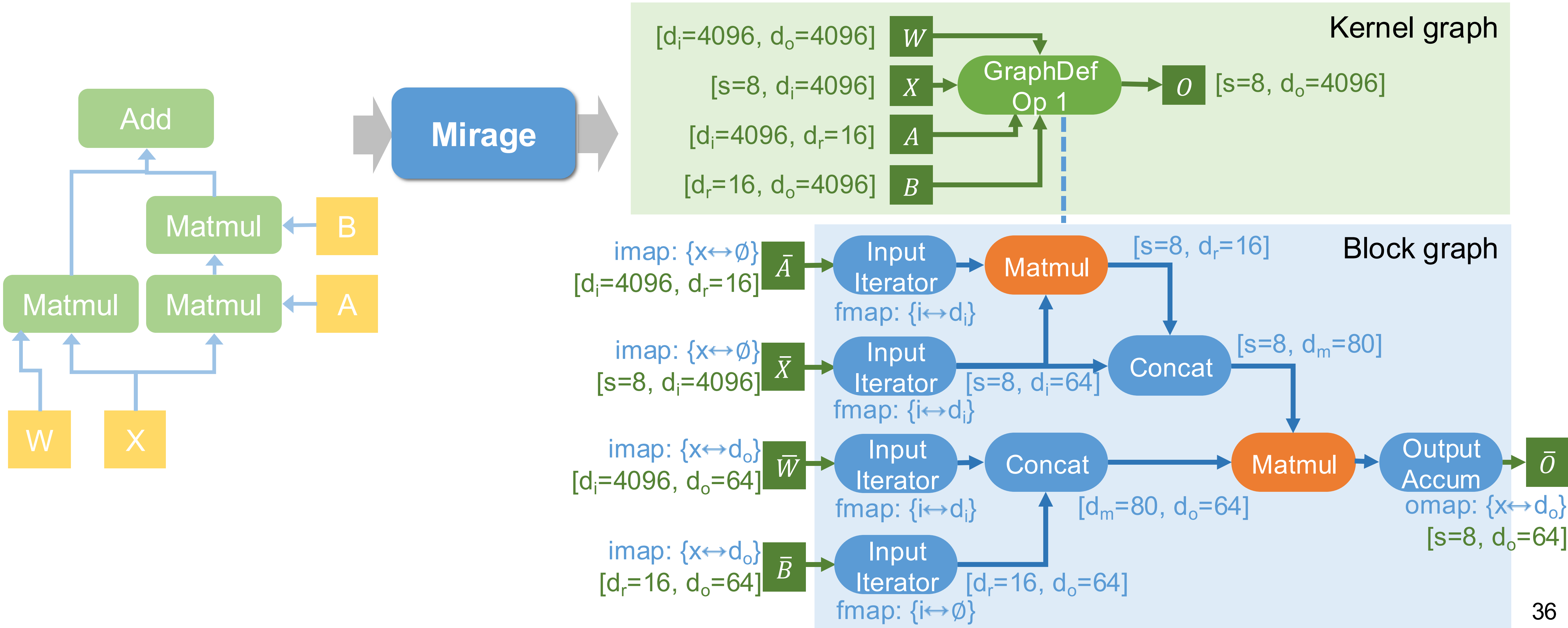


$$RMSNorm(x_i) = \frac{x_i * g_i}{\sqrt{\frac{1}{N}\sum_j x_j^2}}$$

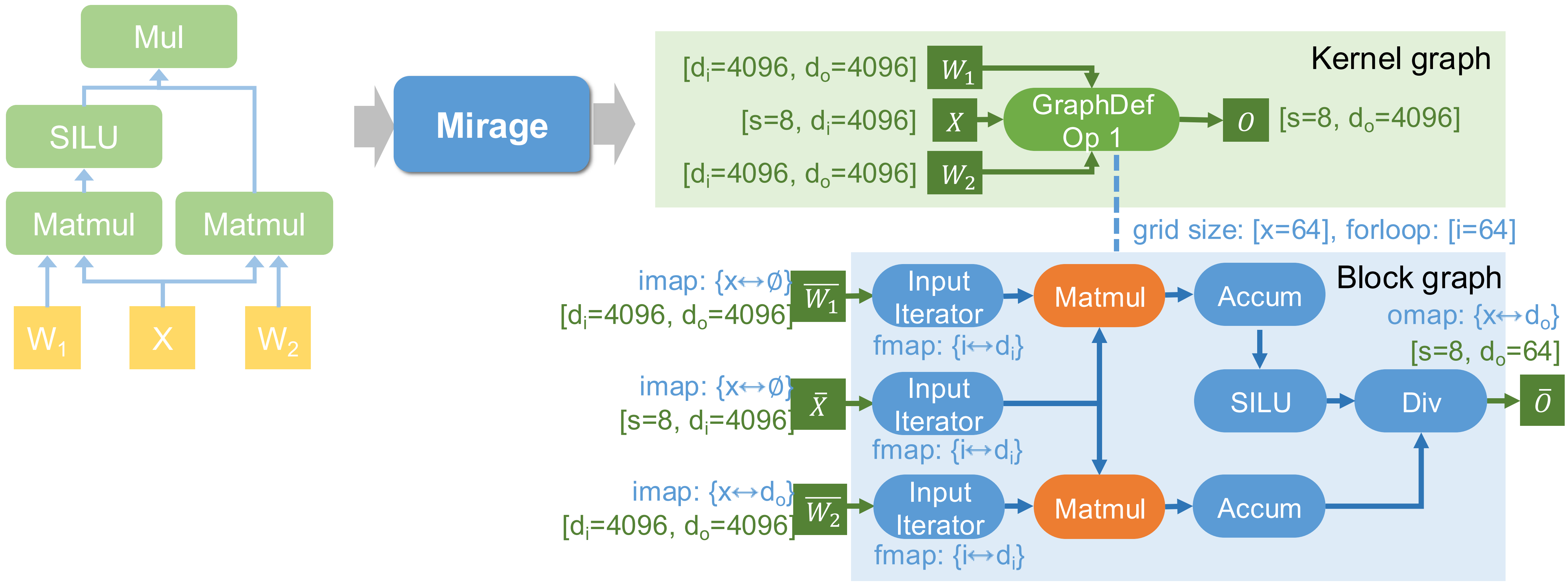


Case 2: Optimized μGraph for LoRA (2.3x faster than Triton)

- Fuse three matmuls and an addition into a single kernel



Case 3: Gated MLP



Questions to Discuss

What are the key differences between Triton, TVM, TASO, Mirage?