Mirage

A Multi-Level Superoptimizer for Tensor Programs

Authors: Wu et al.
Presenter: John Berberian, Jr.

GPUs are Extremely Useful

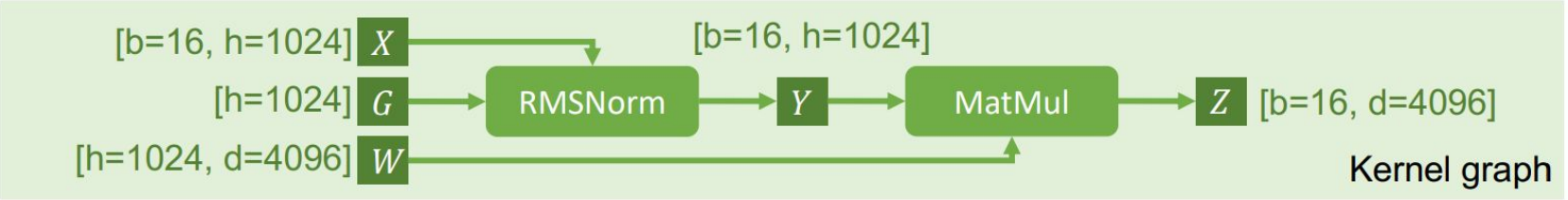
GPUs: just rendering? No more!

- CUDA (2007)
- OpenCL (2009)

Ever since CUDA, GPUs are used everywhere:

- AI/ML, computer vision
- Large-scale data processing
- Simulation, rendering

Tensor Programs



Operate on multidimensional arrays

Possible representation: DAG

- Edges are tensors
- Nodes are operations

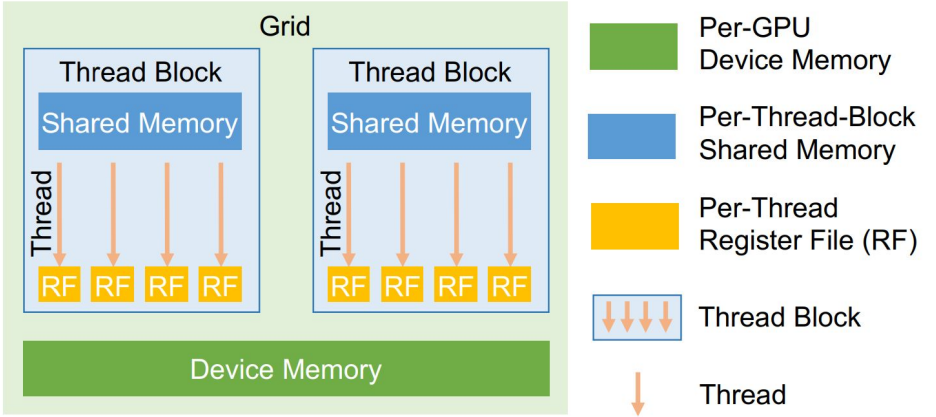
Example: RMSNorm and matrix multiply

$$Y_{ij} = \frac{X_{ij} G_j}{\mathrm{RMS}(X_i)}, \mathrm{RMS}(X_i) = \sqrt{\frac{1}{d} \sum_{j=1}^{d} X_{ij}^2},$$

Problem: Writing Efficient Code is Hard

- Complex GPU memory hierarchy
- Multiple parallel thread blocks:
  - Mapping problem across blocks
  - Efficiently sharing memory
  - Synchronization challenges
- Cache structure

Writing CUDA well is very challenging!




```
__global__ void staticReverse(int *d, int n)
{
  __shared__ int s[64];
  int t = threadIdx.x;
  int tr = n-t-1;
  s[t] = d[t];
  __syncthreads();
  d[t] = s[tr];
}
```

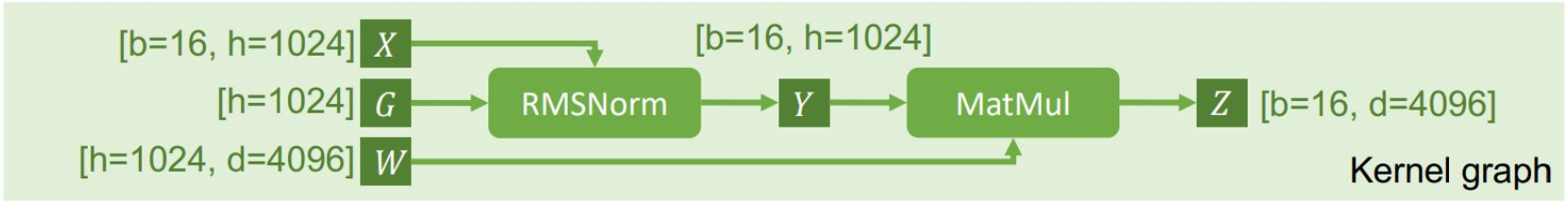
Old Solution: Hand-Optimized Kernels

- Expert-written GPU kernels
- Manually-designed rules for mapping

Much easier to use! But:

- Large up-front engineering cost
- Missed optimization opportunities
- Hardware changes → maintenance cost

PyTorch



Previous Work: Schedule Optimization

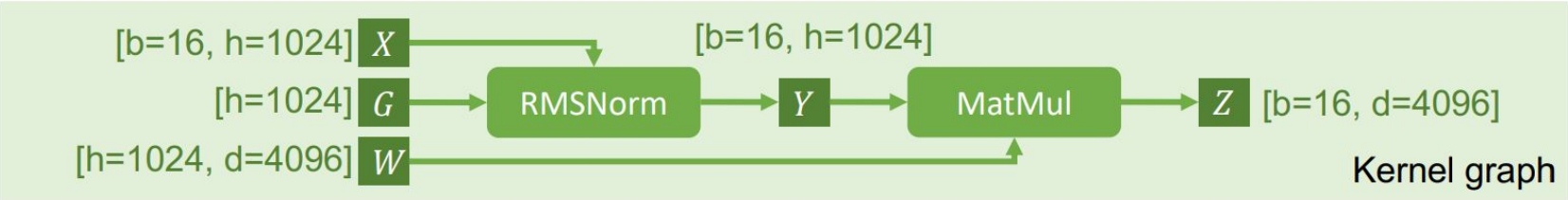
For a given algorithm, optimize how it is mapped to hardware.

Key ideas:

- Separation of algorithm and schedule
- Optimize execution strategies for target hardware

Ex: Halide, TVM, Ansor

Problem: misses algorithmic optimizations



Previous Work: Algorithmic Transformations

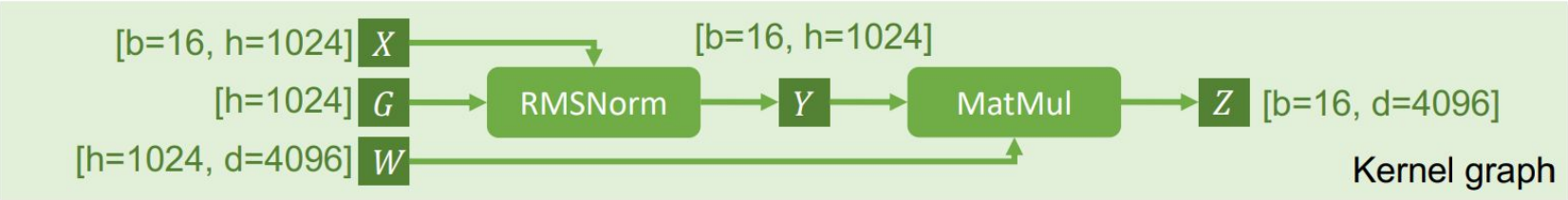
Perform algebraic transformations to simplify algorithms.

Key ideas:

- Operator fusion can simplify computation
- Faster algorithms can be mathematically equivalent

Ex: TASO, Grappler, Tensat, PET.

Problem: cannot perform coordinated algorithm-schedule transformations



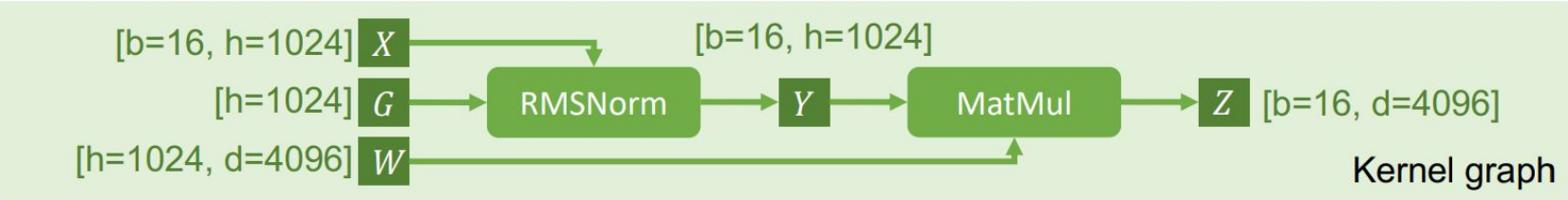
Mirage: A *Multi-Level* Superoptimizer

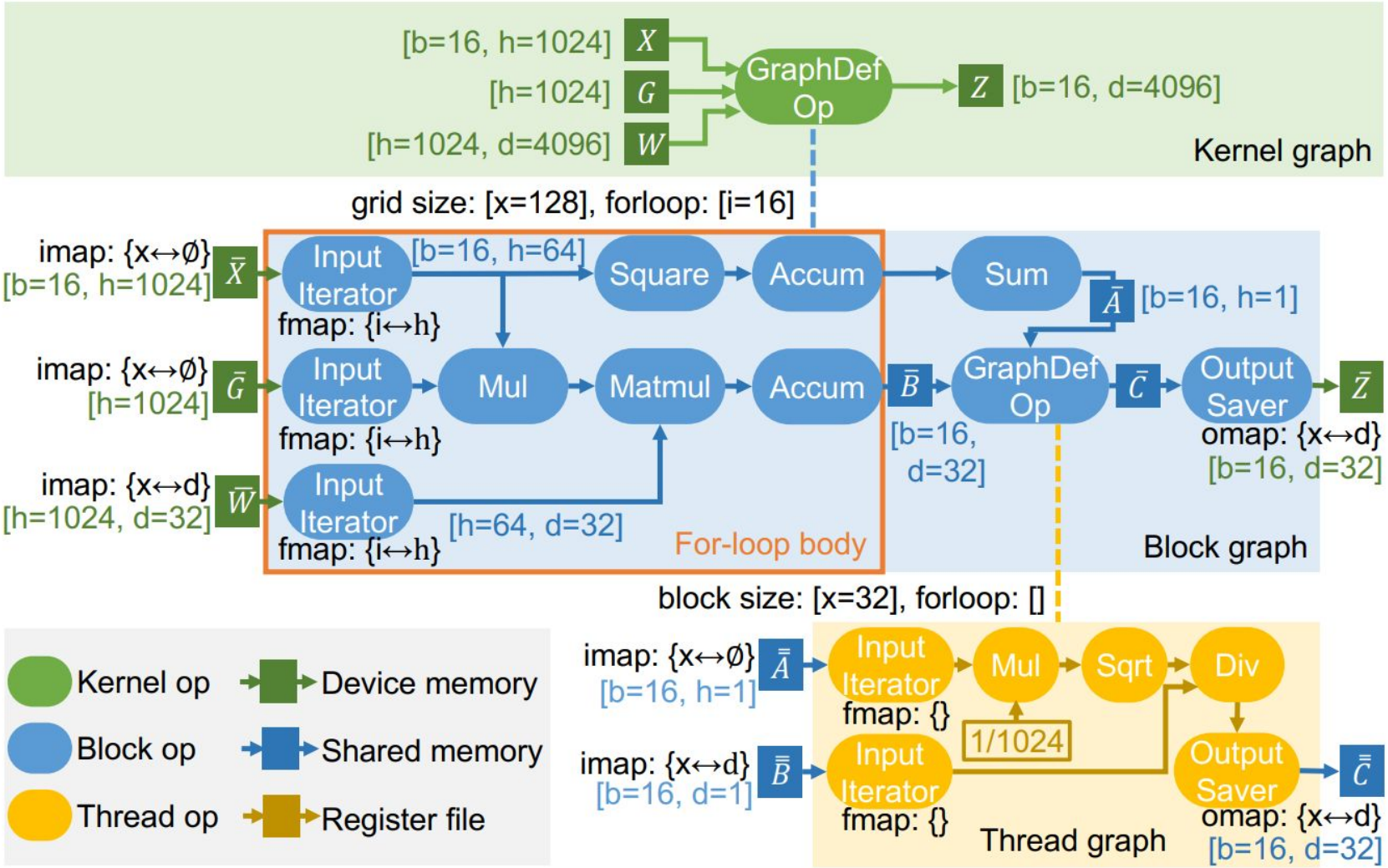
Mirage combines the two optimization methods.

Techniques:

- μGraphs: hierarchical graph representation of tensor program
- Automatic discovery and verification of joint schedule/algorithm optimizations
- Exhaustive search of kernel space can yield optimized custom kernels

Quicker development, higher performance, easier hardware migration

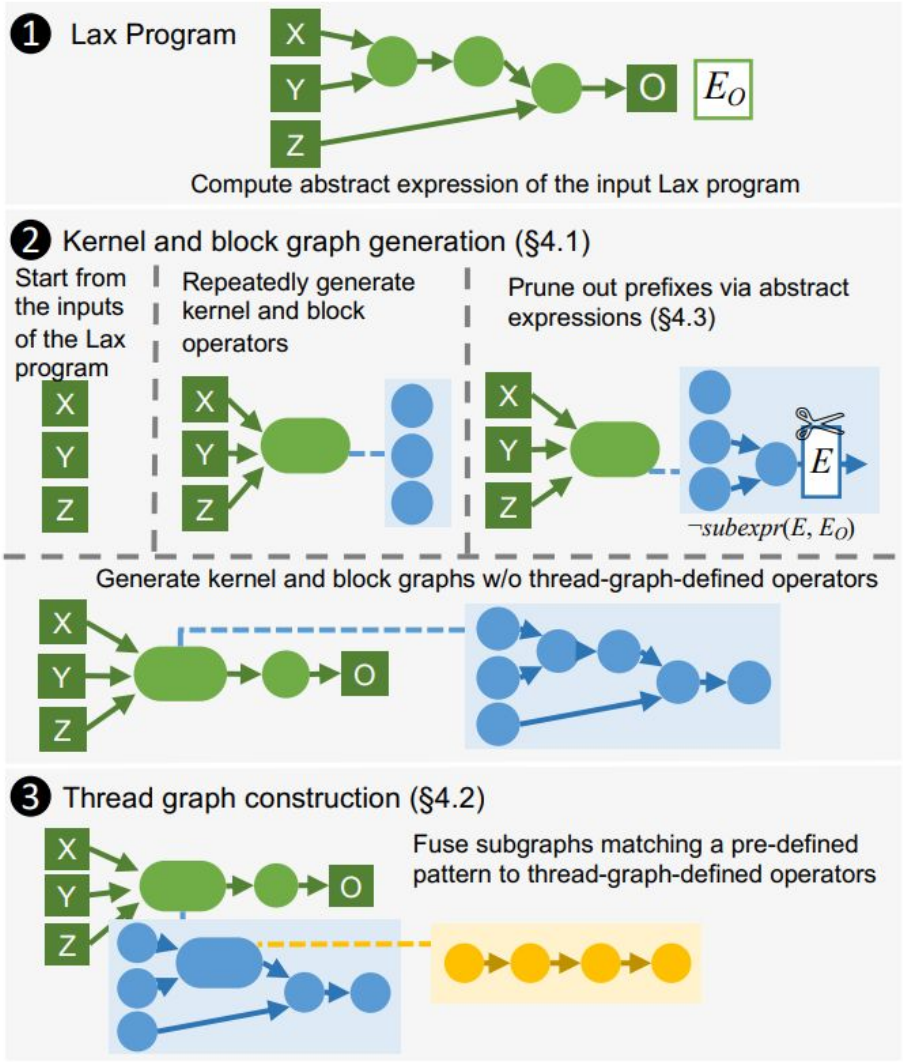



[b=16, h=1024]
X
[h=1024]
G
[h=1024, d=4096]
W
GraphDef Op
Z
[b=16, d=4096]
Kernel graph
grid size: [x=128], forloop: [i=16]
imap: {x↔∅}
[b=16, h=1024]
Input Iterator
[b=16, h=64]
fmap: {i↔h}
Square
Accum
Sum
[b=16, h=1]
imap: {x↔∅}
[h=1024]
Input Iterator
fmap: {i↔h}
Mul
Matmul
Accum
GraphDef Op
Output Saver
omap: {x↔d}
[b=16, d=32]
imap: {x↔d}
[h=1024, d=32]
Input Iterator
fmap: {i↔h}
[h=64, d=32]
For-loop body
[b=16, d=32]
Block graph
block size: [x=32], forloop: []
imap: {x↔∅}
[b=16, h=1]
Input Iterator
fmap: {}
Mul
Sqrt
Div
1/1024
imap: {x↔d}
[b=16, d=1]
Input Iterator
fmap: {}
Output Saver
omap: {x↔d}
[b=16, d=32]
Thread graph
Kernel op
Device memory
Block op
Shared memory
Thread op
Register file

How to explore search space?

More optimizations → larger search space

- Partition program into limited LAX subprograms
- Prioritize kernel- and block-level optimizations → restrict exhaustive search to high-level
- Prune exhaustive search using novel *abstract expressions* technique

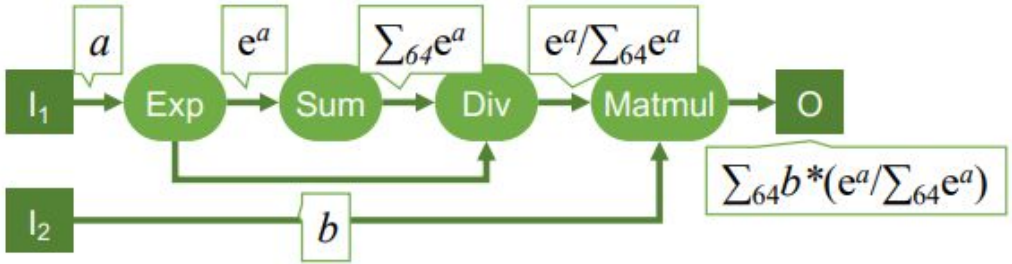


Abstract Expression Pruning

Simplified mathematical representation of μGraph, abstracting indexing details.

- All equivalent μGraphs will have abstract expressions that are substrings, up to some algebraic transformations, of the original.
- First-order logic is used for transforming these expressions.

Explanation of the mechanics was rather unclear.



Probabilistic μGraph Verification

Equivalence of abstract expressions does not guarantee correctness.

To verify the generated μGraphs:

- Perform tests over randomized finite fields
- Repeat testing until chance of error falls below threshold

Authors claim specific error bound for LAX μGraphs; no proof provided.

Paper briefly mentions full (non-probabilistic) verifier for non-LAX programs.

µGraph Optimization

The verified µGraphs might not use optimal memory layout or scheduling.

This stage optimizes:

- Tensor memory layout
- Operator scheduling and synchronization
- Memory access and storage planning

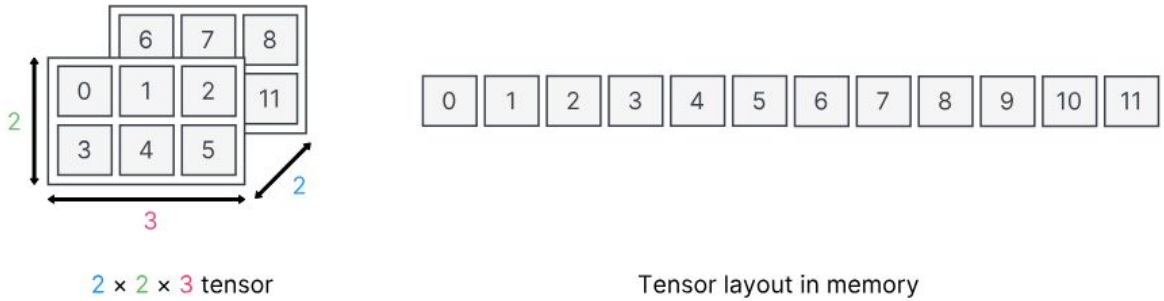


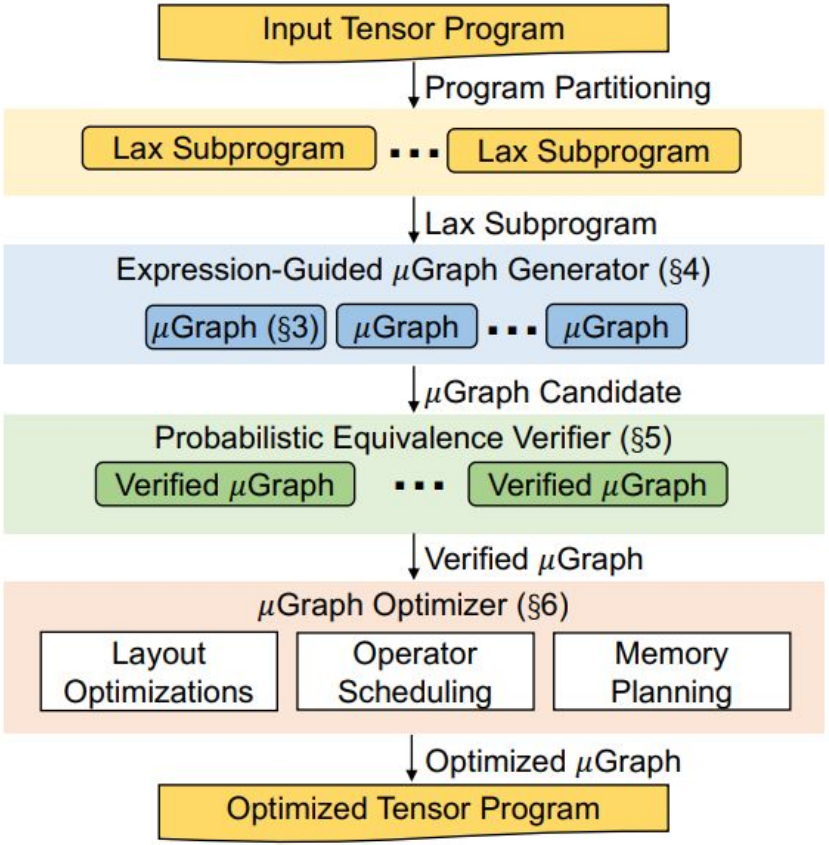
Image credit: Unity Sentis Documentation

Full Mirage Architecture

Put it all together!

Unclear:

- How the most optimal μGraph is selected
- When the LAX subprograms are combined
- How the caching hierarchy plays into optimization choices



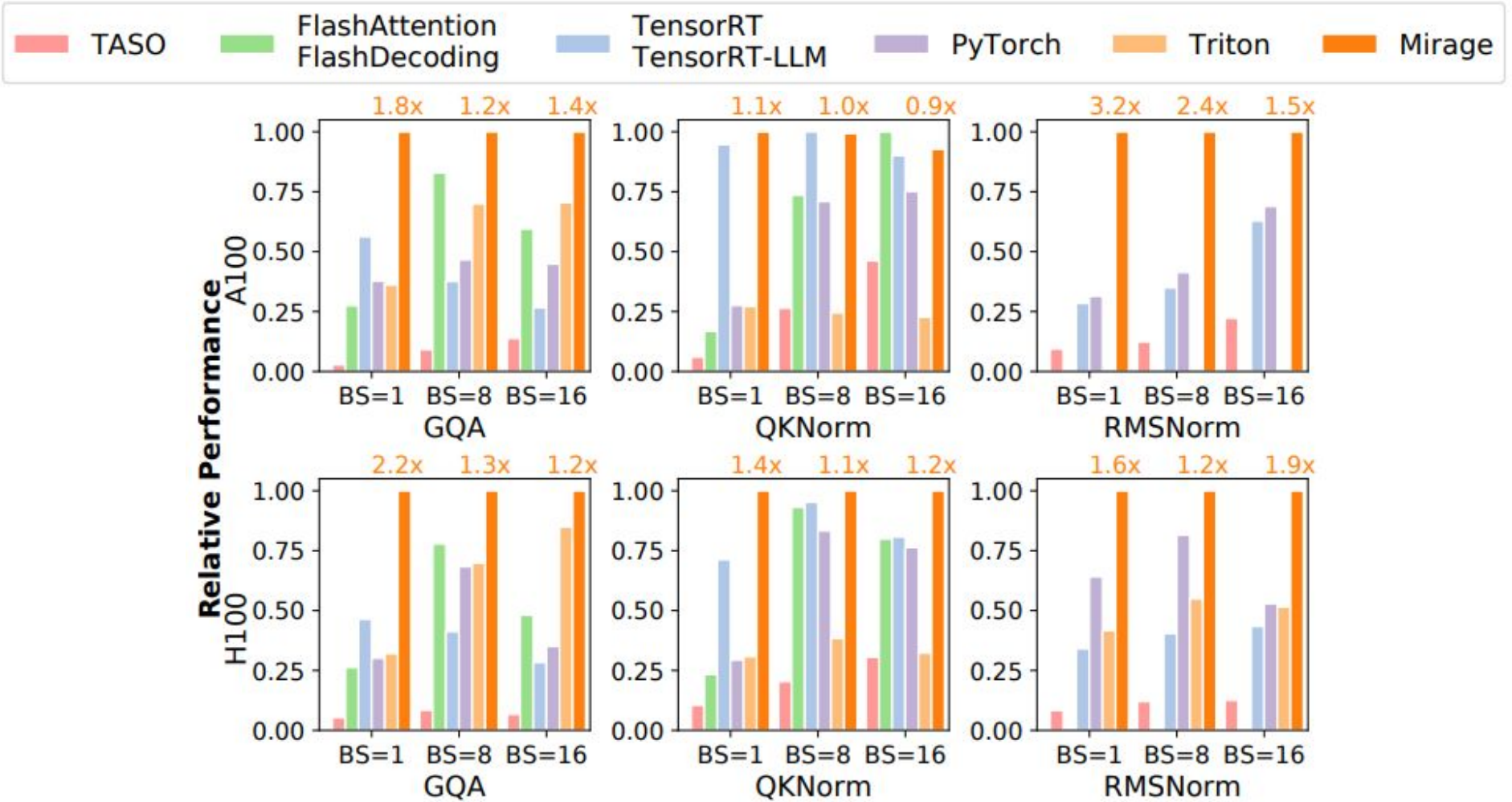
Evaluation

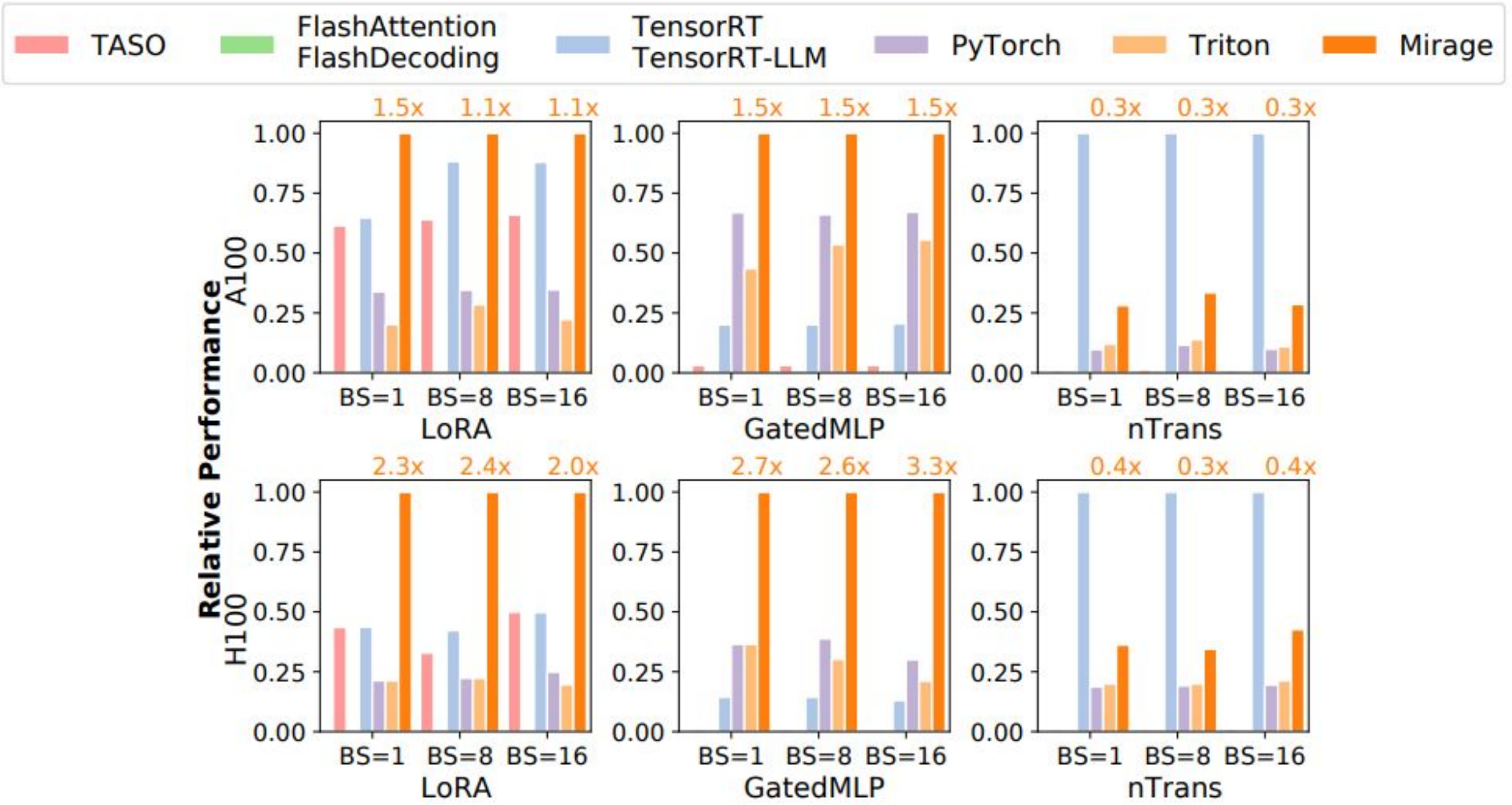
Primarily on DNN benchmarks:

- Group-Query Attention
- Query-Key Normalization
- Low-Rank Adaptation
- RMS Normalization
- Gated Multi-Layer Perceptron
- Normalized Transformer

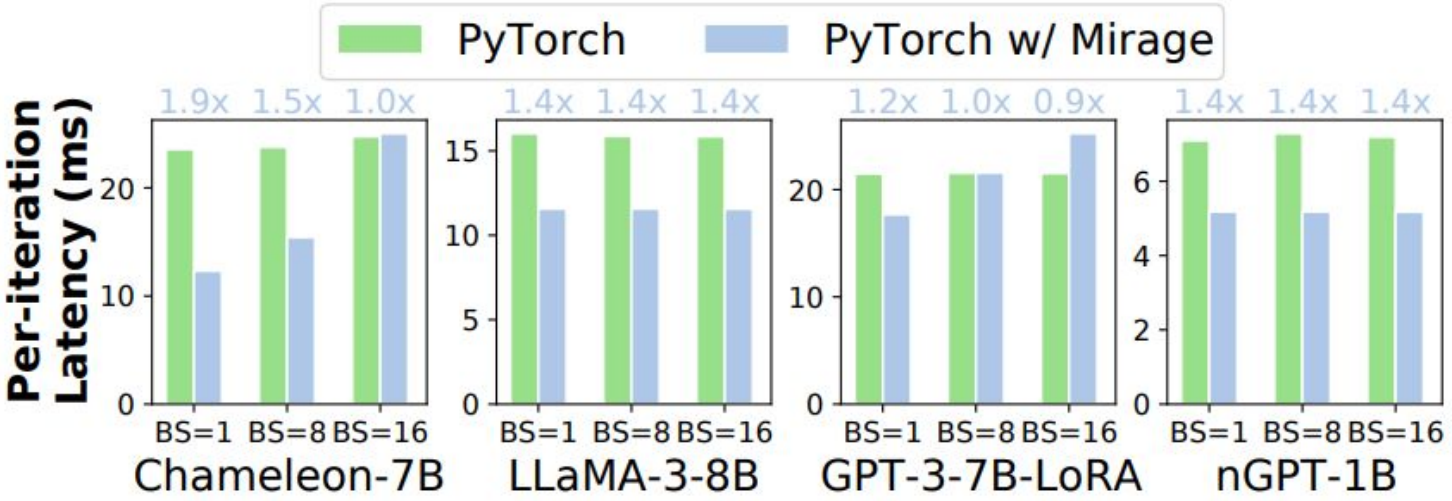
FP32, on A100 and H100, 40GB.

Batch size: 1, 8, 16.


TASO
FlashAttention
FlashDecoding
TensorRT
TensorRT-LLM
PyTorch
Triton
Mirage
Relative Performance
A100
H100
1.8x
1.2x
1.4x
1.1x
1.0x
0.9x
3.2x
2.4x
1.5x
2.2x
1.3x
1.2x
1.4x
1.1x
1.2x
1.6x
1.2x
1.9x
0.00
0.25
0.50
0.75
1.00
BS=1
BS=8
BS=16
GQA
QKNorm
RMSNorm


TASO
FlashAttention
FlashDecoding
TensorRT
TensorRT-LLM
PyTorch
Triton
Mirage
Relative Performance
A100
H100
1.5x
1.1x
1.1x
1.5x
1.5x
1.5x
0.3x
0.3x
0.3x
2.3x
2.4x
2.0x
2.7x
2.6x
3.3x
0.4x
0.3x
0.4x
1.00
0.75
0.50
0.25
0.00
BS=1
BS=8
BS=16
LoRA
GatedMLP
nTrans

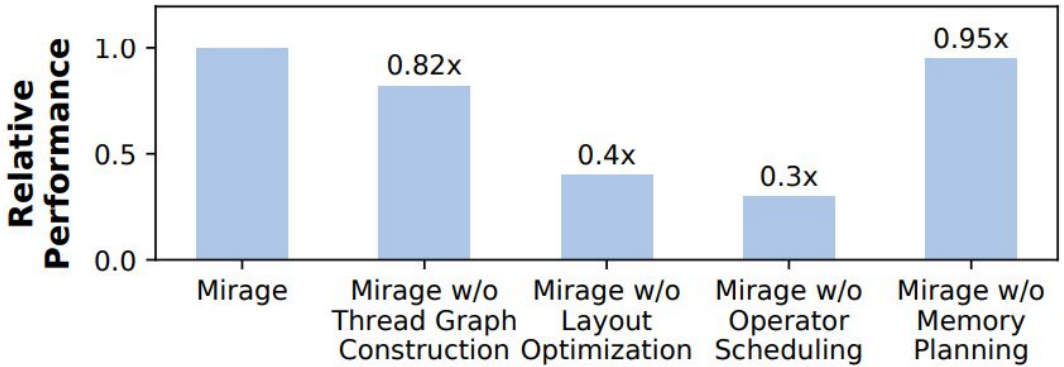
End-to-End Performance



Ablation Studies

What changes when we disable certain parts of the system?

- Kernel performance
- Compilation times



Max # Ops in a Block Graph	Mirage	Mirage w/o Multithreading	Mirage w/o Abstract Expression
5	11 sec	58 sec	768 sec
6	16 sec	93 sec	19934 sec
7	22 sec	150 sec	>10 h
8	24 sec	152 sec	>10 h
9	26 sec	166 sec	>10 h
10	26 sec	166 sec	>10 h
11	28 sec	183 sec	>10 h

Paper Review

Strengths:

- Impressive results
- Good explanation of μGraphs
- Excellent idea

Weaknesses:

- Explanation of mathematical details was unclear
- Output kernels are only *likely* correct
- Optimization is costly for large programs

Questions?