Pushing Tensor Accelerators Beyond MatMul in a User-Schedulable Language

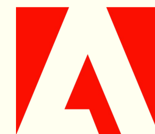
Yihong Zhang
Univ. of Washington

Derek Gerstmann
Adobe

Andrew Adams
Adobe

Maaz Ahmad
Adobe

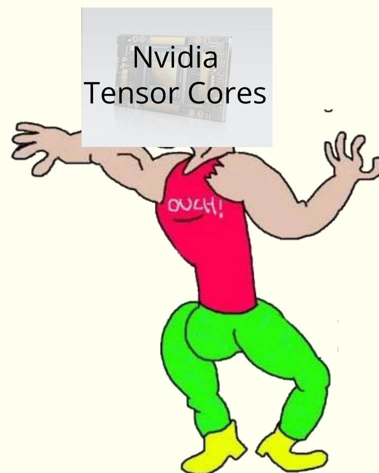
UNIVERSITY of WASHINGTON



Adobe

Filtering, resampling, domain transforms...

Performance engineer working on graphics kernels

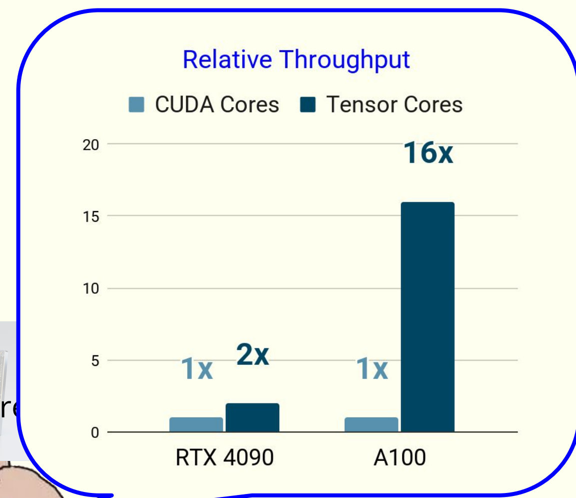


Colleague at the ML department

My kernels are also MatMuls 🤔

Filtering, resampling, domain transforms...

Performance engineer working on graphics kernels



Colleague at the ML department

My kernels are also MatMuls 🤔

Filtering, resampling, domain transforms…

How should I implement my kernels using accelerators?

Performance engineer working on graphics kernels

Colleague at the ML department

My kernels are also MatMuls 🤔

Filtering, resampling, domain transforms…

Performance engineer working on graphics kernels

How should I implement my kernels using accelerators?

- Programming the hardware directly seems hard.

Colleague at the ML department

My kernels are also MatMuls 🤔

Filtering, resampling, domain transforms…

How should I implement my kernels using accelerators?

- Programming the hardware directly seems hard.
- Vendor libraries have rigid interfaces.

Performance engineer working on graphics kernels

Colleague at the ML department

My kernels are also MatMuls 🤔

Filtering, resampling, domain transforms…

Performance engineer working on graphics kernels

How should I implement my kernels using accelerators?

- Programming the hardware directly seems hard.
- Vendor libraries have rigid interfaces.
  - Need to massage the workload around the interfaces.

Colleague at the ML department

My kernels are also MatMuls 🤔

Filtering, resampling, domain transforms...

How should I implement my kernels using accelerators?

- Programming the hardware directly seems hard.
- Vendor libraries have rigid interfaces.
  - Need to massage the workload around the interfaces.
  - Not intuitive to use.
  - Non-optimal performance.

Performance engineer working on graphics kernels

Colleague at the ML department

My kernels are also MatMuls 🤔

Filtering, resampling, domain transforms...

Performance engineer working on graphics kernels

How should I implement my kernels using accelerators?

- Programming the hardware directly seems hard.
- Vendor libraries have rigid interfaces.
  - Need to massage the workload around the interfaces.
  - Not intuitive to use.
  - Non-optimal performance.

We showed: huge missed opportunities here!

- Nonrecursive filtering 3x⇧
- Recursive filtering 1.2x⇧
- Denoising 4x⇧
- Resizing 1.5x⇧

Algorithm

```
mm(y, x) = 0.f;
mm(y, x) += A(r, x) * B(y, r);
```

Hardware-specific Schedule

"tile this loop"
"unroll that computation"
"store that on the stack"

Halide

```
for x in range(0, N, 16):
 for y in range(0, M, 2):
  float mm0[8][2] # local 8x2 buffer
  mm0[0:8, 0] = 0;  mm0[0:8, 1] = 0
  for r in range(0, K):
   mm0[0:8, 0] += A[r, x]   * B[y:y+8, r]
   mm0[0:8, 1] += A[r, x+1] * B[y:y+8, r]
  mm[y:y+8, x]   = mm0[0:8, 0]
  mm[y:y+8, x+1] = mm0[0:8, 1]
```

LLVM

x86

CUDA

ARM

RISC-V

...

Algorithm

```
mm(y, x) = 0.f;
mm(y, x) += A(r, x) * B(y, r);
```

Hardware-specific Schedule

"tile this loop"
"unroll that computation"
"store that on the stack"
"run this with the accelerator"? 🤔

Halide

```
for x in range(0, N, 16):
 for y in range(0, M, 2):
  float mm0[8][2] # local 8x2 buffer
  mm0[0:8, 0] = 0;  mm0[0:8, 1] = 0
  for r in range(0, K):
   mm0[0:8, 0] += A[r, x]   * B[y:y+8, r]
   mm0[0:8, 1] += A[r, x+1] * B[y:y+8, r]
  mm[y:y+8, x]   = mm0[0:8, 0]
  mm[y:y+8, x+1] = mm0[0:8, 1]
```

LLVM

x86 | CUDA | ARM | RISC-V | ...

Wish I can use Halide for tensor accelerators...

Research Problem

Research Problem

To what extent can tensor accelerators be useful for signal and image kernels?

Contributions

HardBoiled	Instruction selector for tensor accelerators in Halide.
Case studies	Significant speedups for kernels including resizing, recursive filtering, denoising.

Writing a MatMul for Intel AMX

```
mm(y, x) = 0.f;
mm(y, x) += A(r, x) * B(y, r);
```

```
for x in ...
  for y in ...
    mm[x][y] = 0
    for r in ...
      mm[x][y] += A[r][x] * B[y][r]
```

Writing a MatMul for Intel AMX

```
mm.tile(y, x, yi, xi, 16, 16)
  .compute_at(mm.in(), xi);
```

```
for x in ...
  for y in ...
    mm[x][y] = 0
    for r in ...
      mm[x][y] += A[r][x] * B[y][r]
```

Writing a MatMul for Intel AMX

```
mm.tile(y, x, yi, xi, 16, 16)
  .compute_at(mm.in(), xi);
```

```
for x in ...
  for y in ...
    float mm0[16][16]
    for xi in range(0, 16):
      for yi in range(0, 16):
        mm0[xi][yi] = 0
    for xi in range(0, 16):
      for yi in range(0, 16):
        for r in ...:
          mm0[xi][yi] += A[..] * B[..]
    for xi in range(0, 16):
      for yi in range(0, 16):
        mm[x*16+xi][y*16+yi] = mm0[xi][yi]
```

local buffer

initialize

accumulate

update the global matrix

Writing a MatMul for Intel AMX

```
mm.tile(y, x, yi, xi, 16, 16)
  .compute_at(mm.in(), xi);
```

```
for x in ...
  for y in ...
    float mm0[16][16]
    for xi in range(0, 16):
      for yi in range(0, 16):
        mm0[xi][yi] = 0
    for xi in range(0, 16):
      for yi in range(0, 16):
        for r in ...:
          mm0[xi][yi] += A[..] * B[..]
    for xi in range(0, 16):
      for yi in range(0, 16):
        mm[x*16+xi][y*16+yi] = mm0[xi][yi]
```

Writing a MatMul for Intel AMX

```
mm.tile(y, x, yi, xi, 16, 16)
  .compute_at(mm.in(), xi)
  .store_in(AMXTile);
```

```
for x in ...
  for y in ...
    float mm0[16][16] in AMX tile register
    for xi in range(0, 16):
      for yi in range(0, 16):
        mm0[xi][yi] = 0
    for xi in range(0, 16):
      for yi in range(0, 16):
        for r in ...:
          mm0[xi][yi] += A[..] * B[..]
    for xi in range(0, 16):
      for yi in range(0, 16):
        mm[x*16+xi][y*16+yi] = mm0[xi][yi]
```

Writing a MatMul for Intel AMX

```
mm.tile(y, x, yi, xi, 16, 16)
  .compute_at(mm.in(), xi)
  .store_in(AMXTile);
mm.vectorize(x)
  .vectorize(y);
mm.update()
  .vectorize(x)
  .vectorize(y)
  .vectorize(r, 32);
mm.in()
  .vectorize(x)
  .vectorize(y);
```

Turns the inner loops into vectorized expressions

```
for x in ...
  for y in ...
    float mm0[16][16] in AMX tile register
    for xi in range(0, 16):
      for yi in range(0, 16):
        mm0[xi][yi] = 0
    for xi in range(0, 16):
      for yi in range(0, 16):
        for r in ...:
          mm0[xi][yi] += A[..] * B[..]
    for xi in range(0, 16):
      for yi in range(0, 16):
        mm[x*16+xi][y*16+yi] = mm0[xi][yi]
```

Writing a MatMul for Intel AMX

```
mm.tile(y, x, yi, xi, 16, 16)
  .compute_at(mm.in(), xi)
  .store_in(AMXTile);
```

```
mm.vectorize(x)
  .vectorize(y);
mm.update()
```

Turns the inner loops into vectorized expressions

Instruction selector only needs to deal with expressions

```
for x in ...
  for y in ...
    float mm0[16][16] in AMX tile register

    mm0[0:16][0:16] = 0

    for r in ...
      mm0[0:16][0:16] += sum(A[..] * B[..])

    mm[x*16:16][y*16:16] = mm0[0:16][0:16]
```

Writing a MatMul for Intel AMX

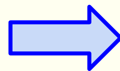
```
mm.tile(y, x, yi, xi, 16, 16)
  .compute_at(mm.in(), xi)
  .store_in(AMXTile);
mm.vectorize(x)
  .vectorize(y);
mm.update()
  .vectorize(x)
  .vectorize(y)
  .vectorize(r, 32);
mm.in()
  .vectorize(x)
  .vectorize(y);
```

Turns the inner loops into vectorized expressions

```
for x in ...
  for y in ...
    float mm0[16][16] in AMX tile register

    tile_zero(mm0)

    for r in …
      tdpbf16ps(mm0, tile_load(A, ..),
                     tile_load(B, ..))

    tile_store(mm, mm0, ..)
```

Tensor Instruction Selection in Halide is Challenging

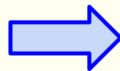
```
mm0[0:16][0:16] = 0.f
mm0[0:16][0:16] += sum(A[r:32][0:16] *
                       B[0:16][r:32])
mm[x*16:16][y*16:16] = mm0[0:16][0:16]
```



```
tile_zero(mm0)
tdpbf16ps(mm0, tile_load(A, ..),
               tile_load(B, ..))
tile_store(mm, mm0, ..)
```

Tensor Instruction Selection in Halide is Challenging

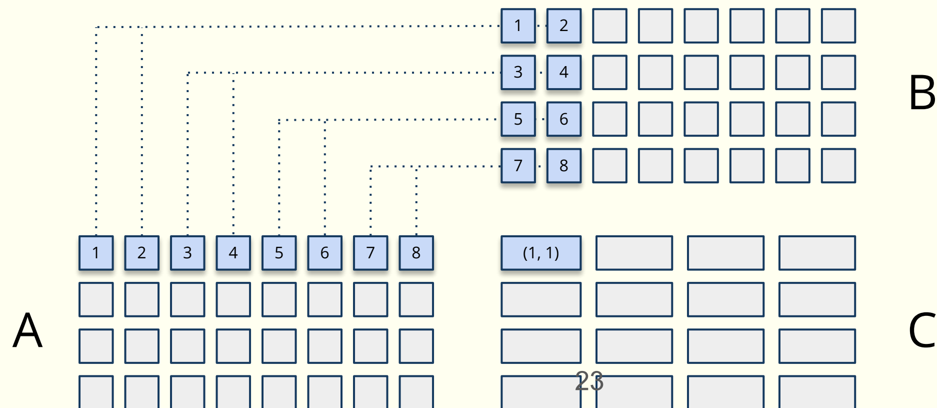
```
mm0[0:16][0:16] = 0.f
mm0[0:16][0:16] += sum(A[r:32][0:16] *
                       B[0:16][r:32])
mm[x*16:16][y*16:16] = mm0[0:16][0:16]
```



```
tile_zero(mm0)
tdpbf16ps(mm0, tile_load(A, ..),
               tile_load(B, ..))
tile_store(mm, mm0, ..)
```

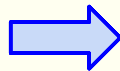
It’s not as easy as it seems!

1. AMX expects a different matrix layout than shown above.



Tensor Instruction Selection in Halide is Challenging

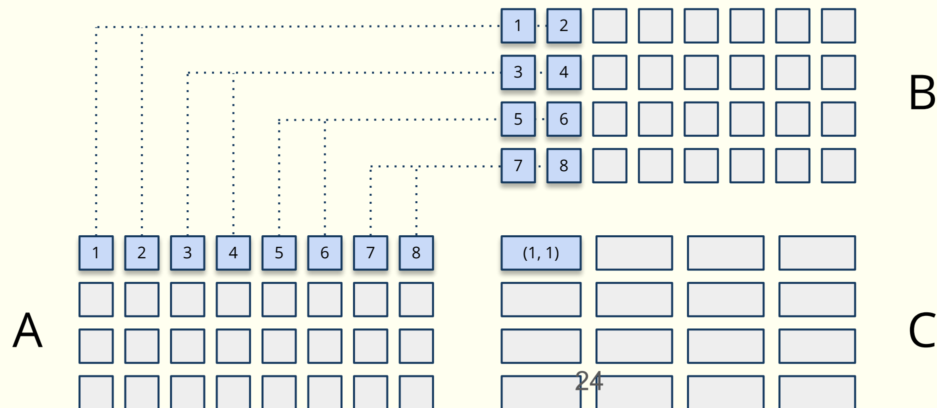
```
mm0[0:16][0:16] = 0.f
mm0[0:16][0:16] += sum(A[r:32][0:16] *
                       B[0:16][r:32])
mm[x*16:16][y*16:16] = mm0[0:16][0:16]
```



```
tile_zero(mm0)
tdpbf16ps(mm0, tile_load(A, ..),
               tile_load(B, ..))
tile_store(mm, mm0, ..)
```

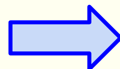
It's not as easy as it seems!

1. AMX expects a different matrix layout than shown above.
2. Halide does not track dimensions of the expression.



Tensor Instruction Selection in Halide is Challenging

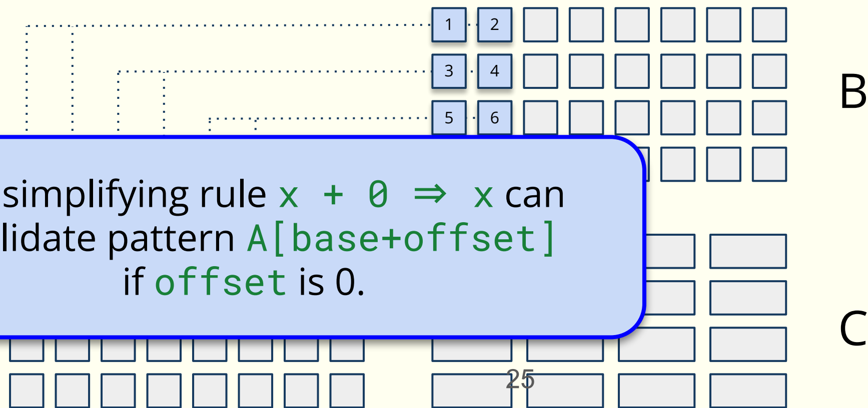
```
mm0[0:16][0:16] = 0.f
mm0[0:16][0:16] += sum(A[r:32][0:16] *
                       B[0:16][r:32])
mm[x*16:16][y*16:16] = mm0[0:16][0:16]
```



```
tile_zero(mm0)
tdpbf16ps(mm0, tile_load(A, ..),
               tile_load(B, ..))
tile_store(mm, mm0, ..)
```

It's not as easy as it seems!

1. AMX expects a different matrix layout than shown above.
2. Halide does not track dimensions of the expression.
3. Simplification leads to phase-ordering problems.



E.g., simplifying rule `x + 0 ⇒ x` can invalidate pattern `A[base+offset]` if `offset` is 0.

Tensor Instruction Selection in Halide is Challenging

<u>Actual pattern to match…</u>

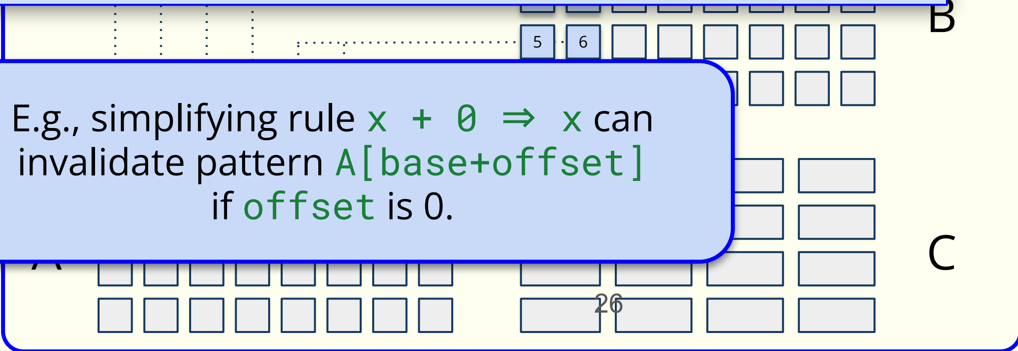
```
mm0[ramp(0, 1, 512)] = x512(0.f)
mm0[ramp(0, 1, 512)] =
  (float32x512)vector_reduce_add(
    (cast<float32x8192>(
      A[ramp(x512(0), x512(32), 16) + x256(ramp(0, 1, 32))]
    ) * x16(
      cast<float32x256>(B[ramp(ramp(ramp(0, 1, 2), x2(16), 16), x32(2), 16)])
    ))) + mm0[ramp(0, 1, 512)]
mm[ramp(0, 1, 512)] = mm0[ramp(0, 1, 512)]
```

layout than shown above.

2. Halide does not track dimensio of the expression.

3. Simplification leads to phase-ordering problems.

E.g., simplifying rule `x + 0 ⇒ x` can invalidate pattern `A[base+offset]` if `offset` is 0.



Previous AMX Support

Implementation

- Manually enumerate every possible syntactic pattern a priori.
- Took **seven** months to develop initial support (#5818).
- Took another **six** months to generalize (#6582).

Previous AMX Support

Implementation

- Manually enumerate every possible syntactic pattern a priori.
- Took **seven** months to develop initial support (#5818).
- Took another **six** months to generalize (#6582).

Limitations

- Still has a number of bugs (#8350).
- Cannot support signal and image kernels we wanted to develop.
- Hard to extend support to new patterns.

Equality Saturation to the Rescue

Input program

Tries all possible orders of rewrites to find the best program.

Equational rules

$x + y \Leftrightarrow y + x$

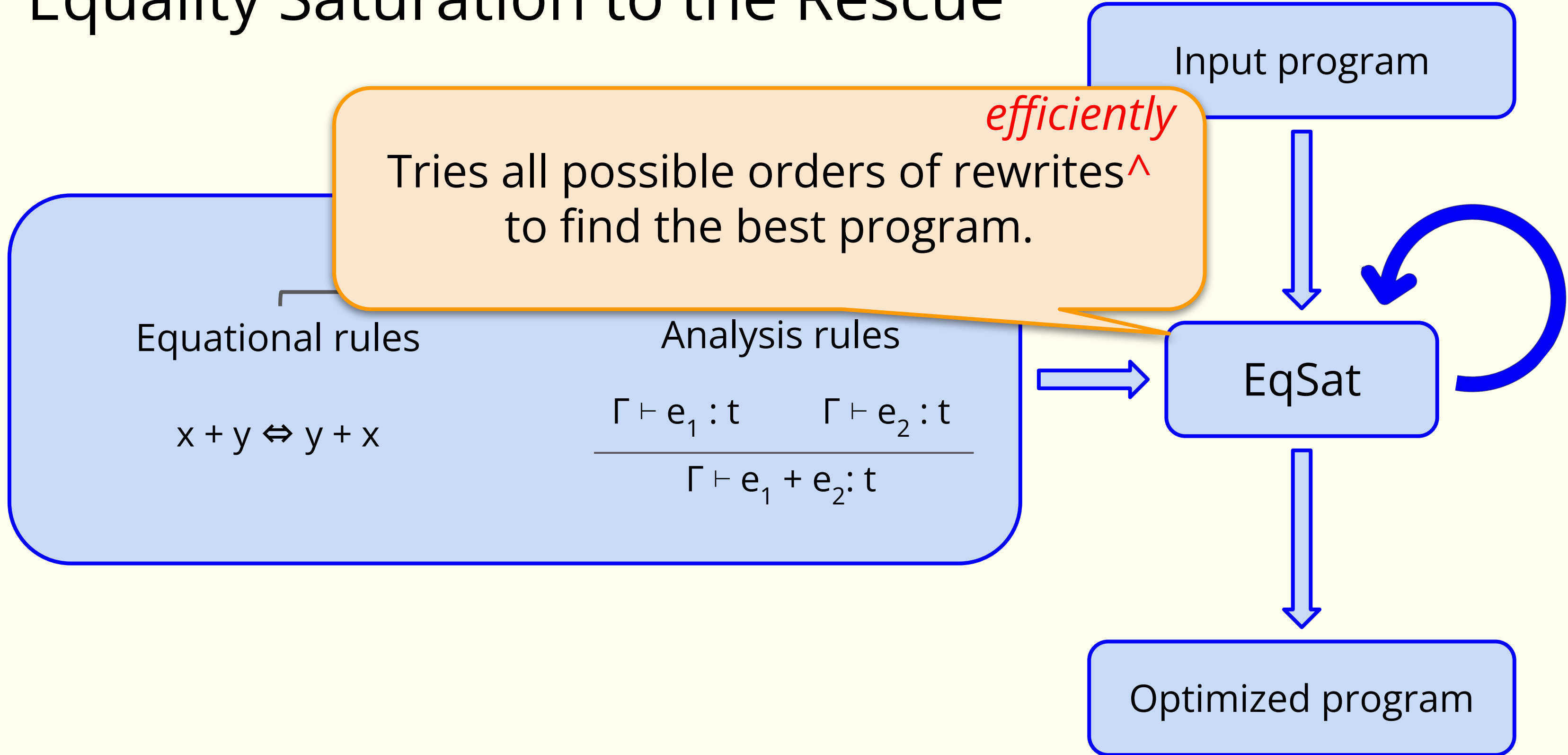
Analysis rules

$$\frac{\Gamma \vdash e_1 : t \qquad \Gamma \vdash e_2 : t}{\Gamma \vdash e_1 + e_2 : t}$$

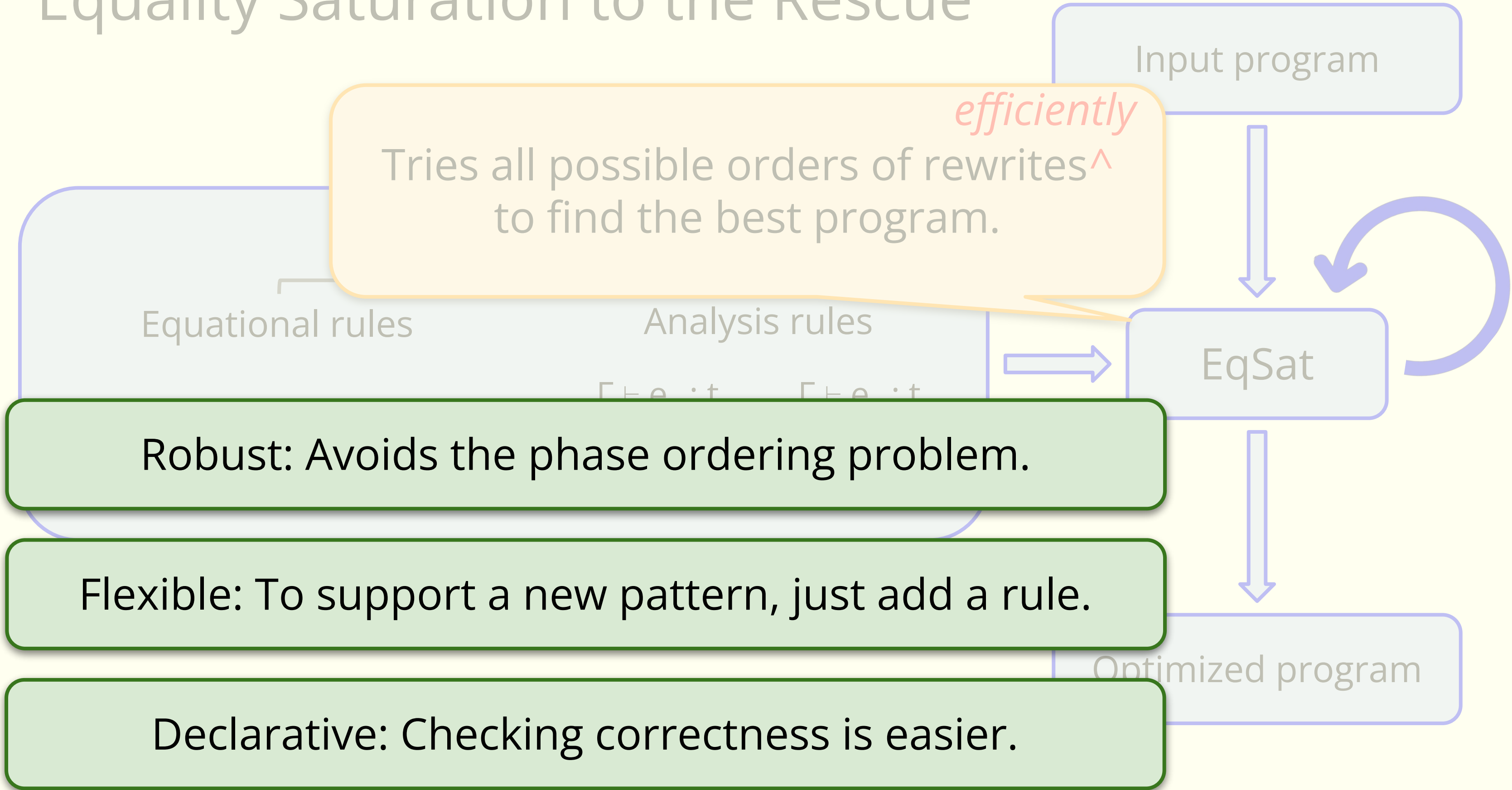
EqSat

Optimized program

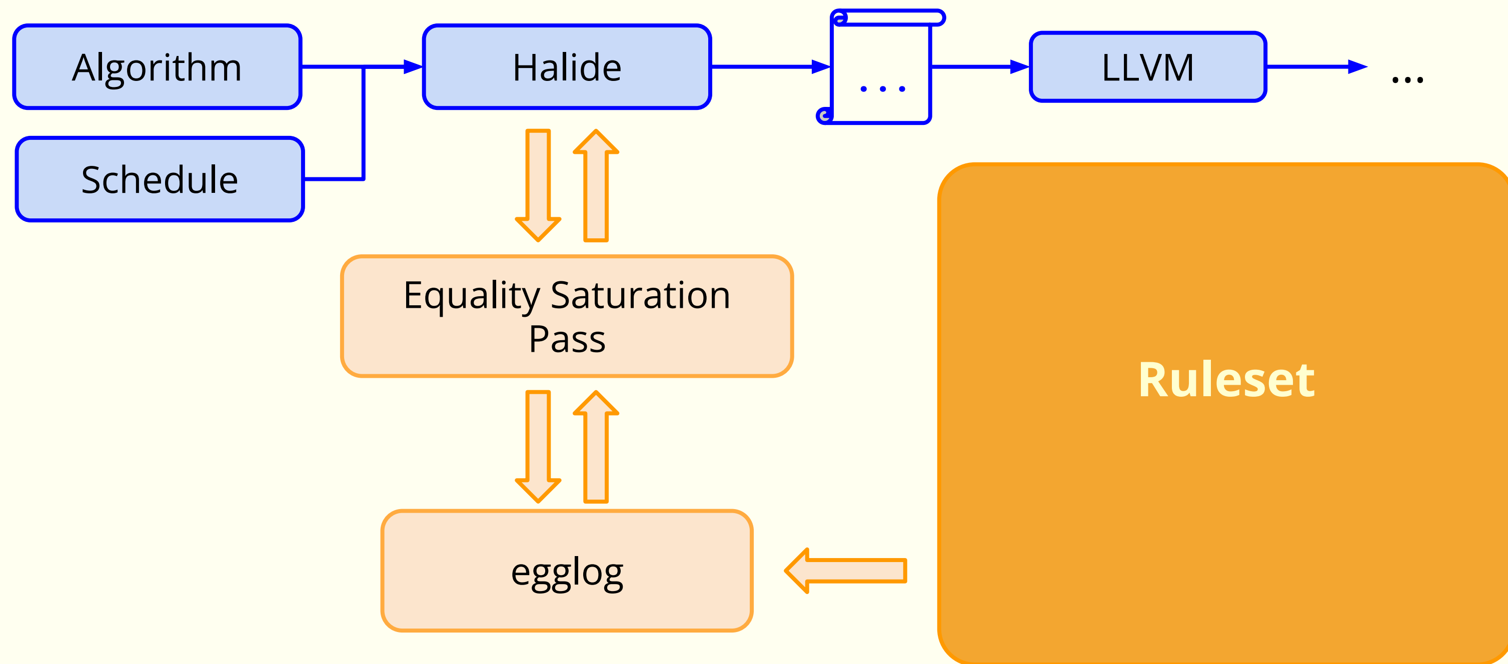
Equality Saturation to the Rescue



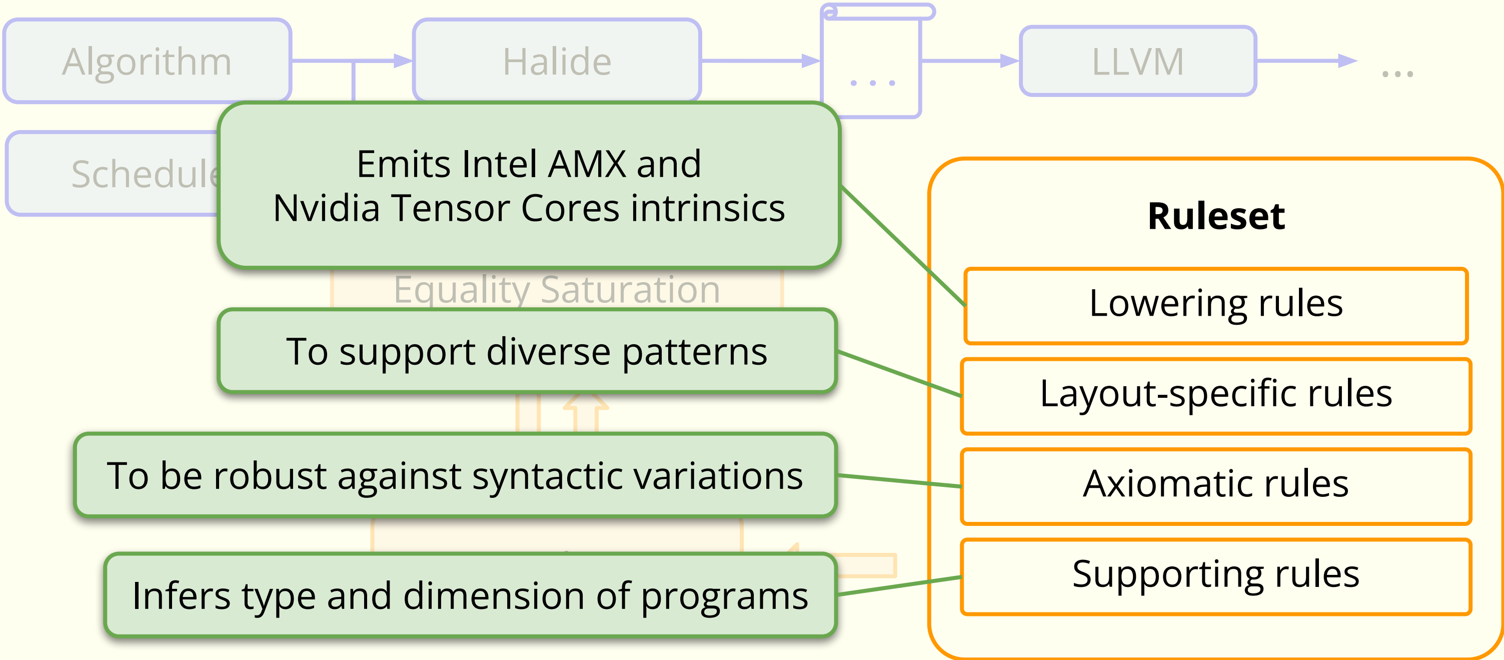
Equality Saturation to the Rescue



Implementation



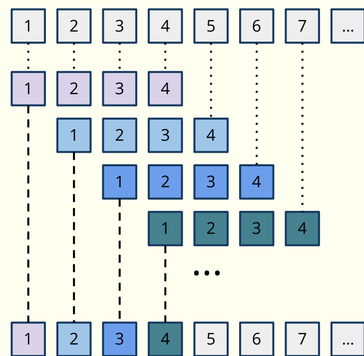
Implementation

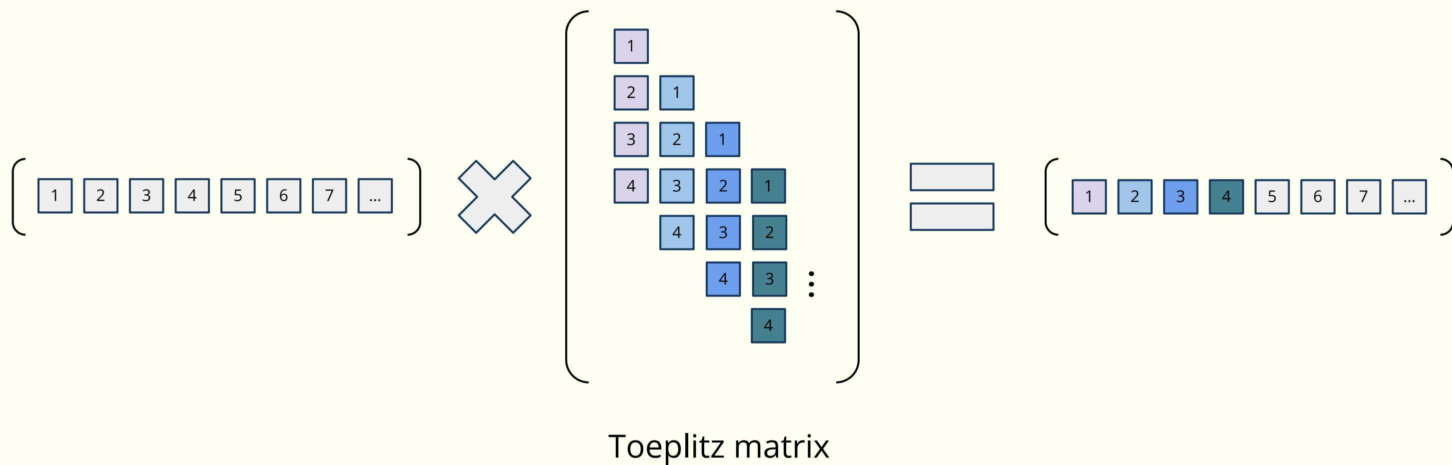


Supporting patterns beyond MatMul

Convolution as MatMul

```
conv(x) = 0
conv(x) += img(x + r) * k(r)
```





Toeplitz matrix

Convolution

```
conv(x) = 0
conv(x) += img(
```

Downsampling by 2x:

Upsampling by 2x:

Toeplitz matrix

✅ Supported in HardBoiled

Microbenchmarks on Nvidia Tensor Cores

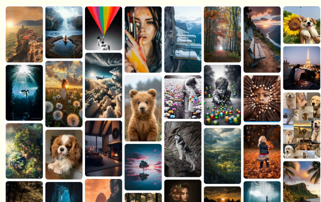
	k=16	k=32
Convolution	3x ⇧	2.4x ⇧
Downsampling	5x ⇧	6x ⇧
Upsampling	1.4x ⇧	3x ⇧

Accelerating end-to-end applications

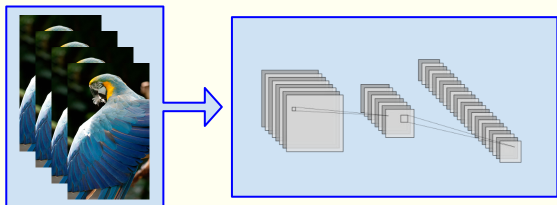
Case Study I: Resizing Images to a Fixed Resolution

Use cases

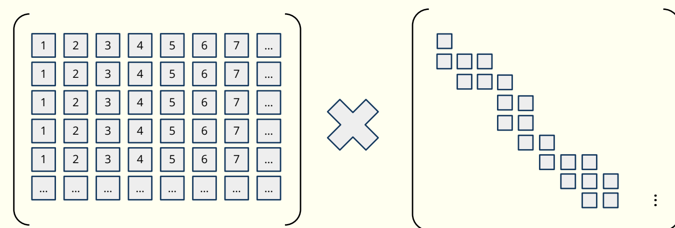
- Thumbnail generation



- Preprocessing images for training



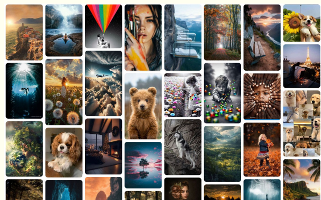
Algorithm



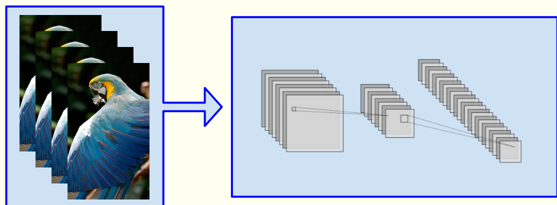
Case Study I: Resizing Images to a Fixed Resolution

Use cases

- Thumbnail generation



- Preprocessing images for training



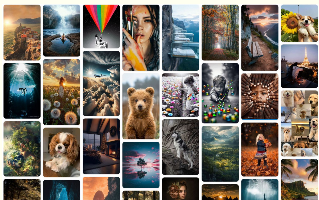
Algorithm

- Reformulated as block-sparse matrix multiplication.

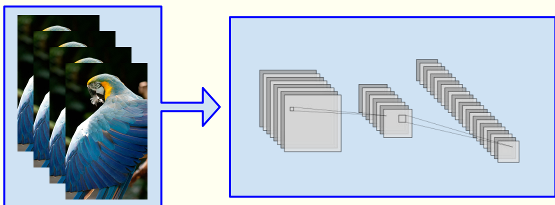
Case Study I: Resizing Images to a Fixed Resolution

Use cases

- Thumbnail generation



- Preprocessing images for training



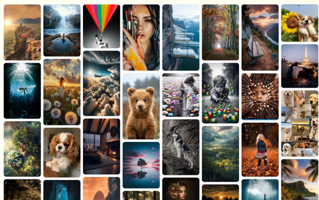
Algorithm

- Reformulated as block-sparse matrix multiplication.
  - **3x** ⇧ without Tensor Cores
  - High compute/bandwidth utilization.

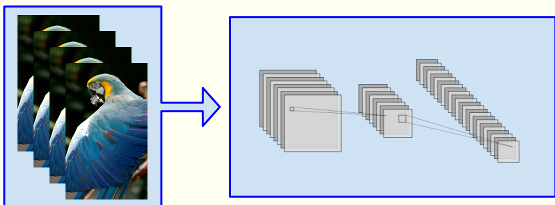
Case Study I: Resizing Images to a Fixed Resolution

Use cases

- Thumbnail generation



- Preprocessing images for training



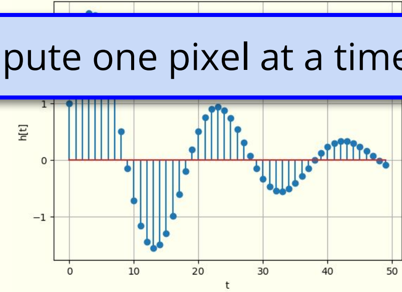
Algorithm

- Reformulated as block-sparse matrix multiplication.
  - **3x** ⇧ without Tensor Cores
  - High compute/bandwidth utilization.
- Changed to use a Tensor Cores schedule.
  - Additional **1.5x** ⇧

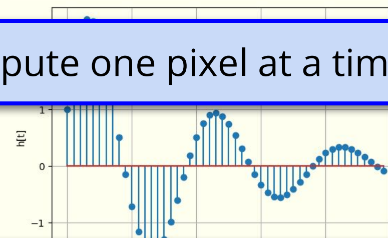
Case Study II: Recursive Filtering

Can only compute one pixel at a time.

$$Y[t] = X[t] + \alpha Y[t-1] + \beta Y[t-2]$$



Case Study II: Recursive Filtering



Can only compute one pixel at a time.

$$Y[t] = X[t] + \alpha Y[t-1] + \beta Y[t-2]$$

Reformulate using Scattered-Lookahead Interpolation

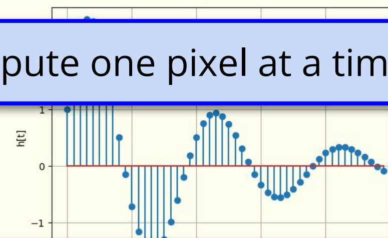
Can compute d pixels in parallel.

$$Y[t] = X'[t] + \alpha' Y[t-d] + \beta' Y[t-2d]$$

Case Study II: Recursive Filtering

Can only compute one pixel at a time.

$$Y[t] = X[t] + \alpha Y[t-1] + \beta Y[t-2]$$



Reformulate using Scattered-Lookahead Interpolation

Can compute d pixels in parallel.

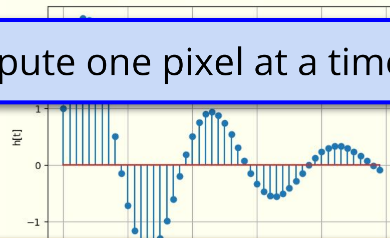
$$Y[t] = X'[t] + \alpha' Y[t-d] + \beta' Y[t-2d]$$

where X’ is the convolution of X with a kernel of size 2d-1.

Case Study II: Recursive Filtering

Can only compute one pixel at a time.

$$Y[t] = X[t] + \alpha Y[t-1] + \beta Y[t-2]$$



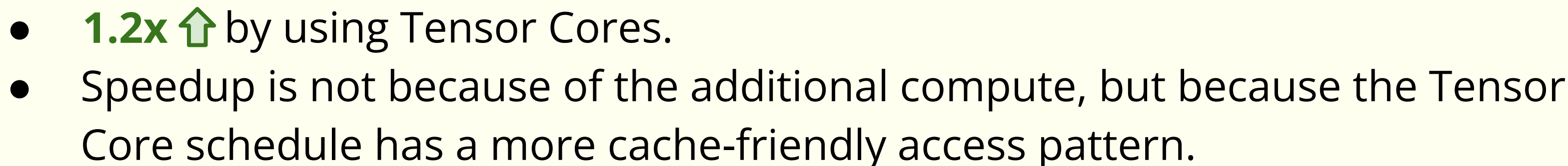
Reformulate using Scattered-Lookahead Interpolation

Can compute d pixels in parallel.

$$Y[t] = X'[t] + \alpha' Y[t-d] + \beta' Y[t-2d]$$

where X' is the convolution of X with a kernel of size 2d-1.

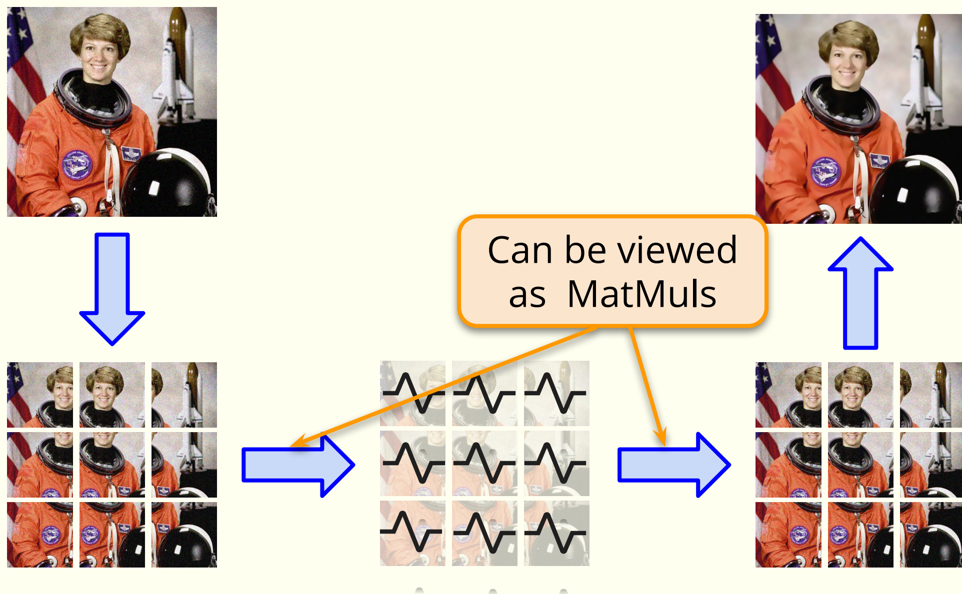
HardBoiled recognizes the convolution and generates Tensor Core intrinsics.



- **1.2x** ⇧ by using Tensor Cores.
- Speedup is not because of the additional compute, but because the Tensor Core schedule has a more cache-friendly access pattern.

Case Study III: Denoising

Denoise images using discrete cosine transform (DCT).



Tensor Core schedule

- 4x speedup over CUDA baseline.
- 10% faster than fast DCT, despite doing 3.6x more flops.

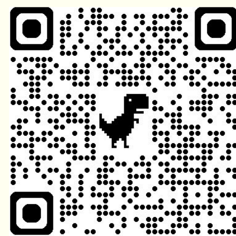
Summary

Huge potentials in using tensor accelerators for non-conventional workloads.

User-schedulable languages like Halide can unlock tensor accelerators for new workloads

Equality saturation makes supporting tensor accelerators easy.

Implementation & Case Studies

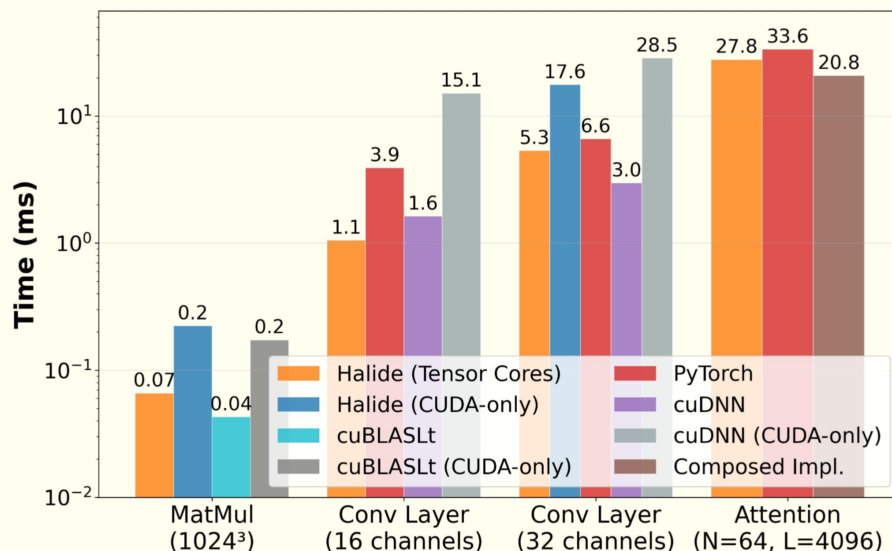


Paper

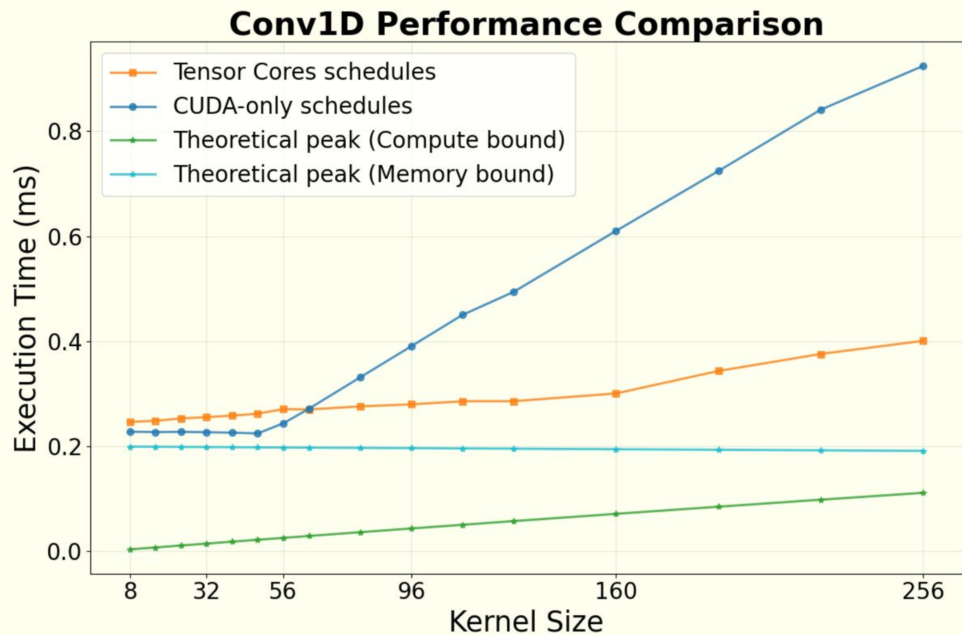
Sanity Check on Machine Learning Workloads

Implementation	VNNI	Standard
Reference impl.	✅	✅
Loop reordering	✅	✅
Preloading matrix A	✅	✅
Preloading matrix B	✅	
Software pipelining		

Intel's Optimization Manual



Scaling



Compilation time