MICHIGAN ENGINEERING
UNIVERSITY OF MICHIGAN

Triton: an intermediate language and compiler for tiled neural network computations

Paper by: P. Tillet, H. T. Kung, D. Cox

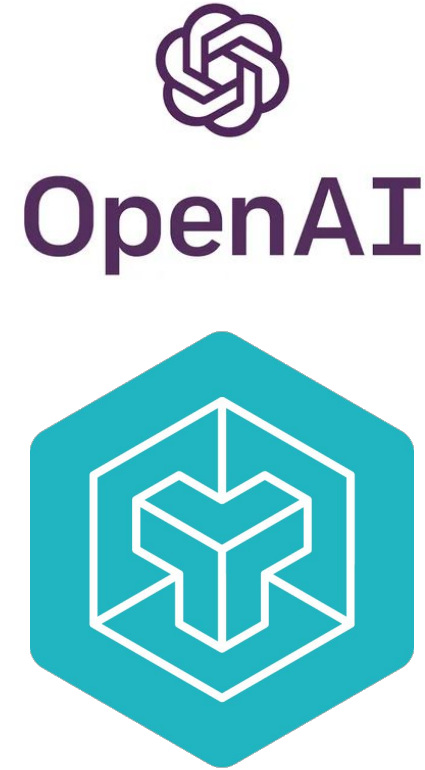
Group 9: Alan Zhang, Andrew Wang, Ben Schattinger, Josh Brodsky, Lani Quach

Introduction

- Deep learning computation and compiler optimization for GPUs
- Limited tensor operation support with current vendor libraries
- Current solution requires hand-written GPU kernels (expensive and timely)
    - Operator fusion
- Current DSLs struggle with performance and tile-level optimizations
- In 2019 Triton was released

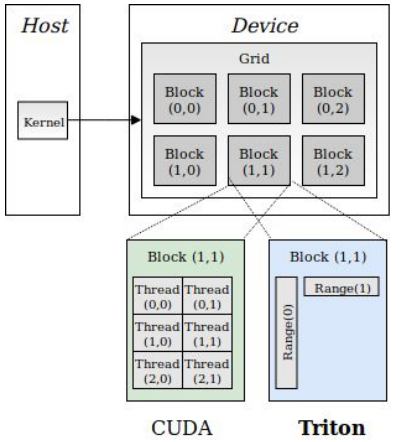
Triton

- Write custom, optimized GPU kernels
- Language + Compiler
- **Triton-C:** C-like language for writing GPU tensor kernels
- **Triton-IR:** LLVM-based layer
- **Triton-JIT:** JIT compiler with auto-tuning



Triton C Language

- Triton's modern DSL is in **Python**
- Tiles are the main abstraction
  - small blocks of a tensor with tunable shapes (compile time constants)
- Numpy-style semantics like "+", "*", "dot" defined over whole tiles
- Kernels follow an SPMD model instead of manual thread/block management (CUDA)




```
int a[16], b[32, 16], c[16, 1];
// a is first reshaped to [1, 16]
// and then broadcast to [32, 16]
int x_1[32, 16] = a[newaxis, :] + b;
// Same as above but implicitly
int x_2[32, 16] = a + b;
// a is first reshaped to [1, 16]
// a is broadcast to [16, 16]
// c is broadcast to [16, 16]
int y[16, 16] = a + c;
```

Triton IR

High-Level IR generated directly from Trition's DSL
Triton DSL -> Triton IR -> LLVM

```
define kernel void @relu(float* %A, i32 %M, i32 %N) {
prologue:
  %rm = call i32<8> get_global_range(0);
  %rn = call i32<8> get_global_range(1);
  ; broadcast shapes
  %1 = reshape i32<8, 8> %M;
  %M0 = broadcast i32<8, 8> %1;
  %2 = reshape i32<8, 8> %N;
  %N0 = broadcast i32<8, 8> %2;
```

How does Triton IR's introduction of tiles help support:

- *Data-Flow Analysis*
  - Challenge: ML workloads require locality and parallelism
  - Solution: Tiles abstraction enables blocked computation and optimizes memory layout
- *Control-Flow Analysis*
  - Challenge: Elements in a tile may diverge in branches
  - Solution: Predicated-SSA (PSSA) and ψ-functions to merge branches

Machine Independent Passes

- Prefetching for tile-level memory operations in loops
  - Hide memory latency by pipelining

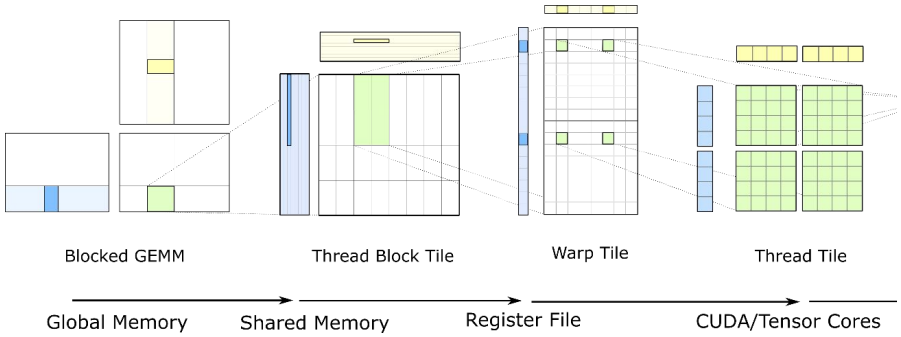
```
B0:
 %p0 = getelementptr %1, %2
B1:
 %p = phi [%p0,B0], [%p1,B1]
 %x = load %p
 ; increment pointer
 %p1 = getelementptr %p, %3
```

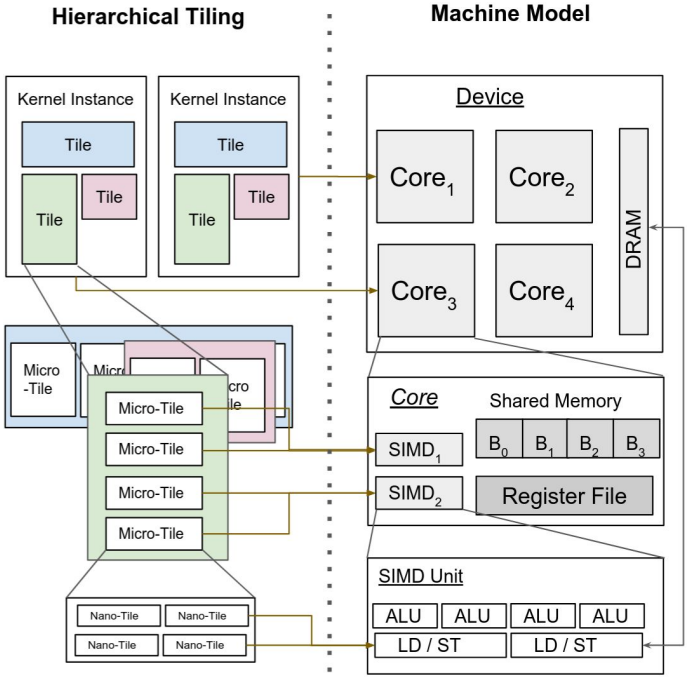
```
B0:
 %p0 = getelementptr %1, %2
 %x0 = load %p0
B1:
 %p = phi [%p0,B0], [%p1,B1]
 %x = phi [%x0,B0], [%x1,B1]
 ; increment pointer
 %p1 = getelementptr %p, %3
 ; prefetching
 %x1 = load %p
```

- Tile level operations in Triton IR allows for peephole optimizers
  - i.e $(X^T)^T = X$
  - Other algebraic properties able to be exploited

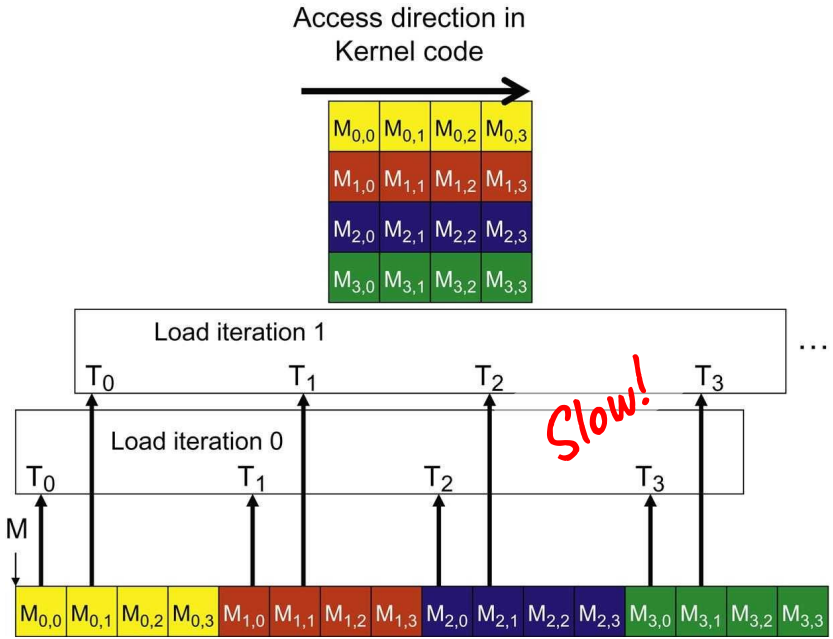
Machine Dependent Passes: Hierarchical Tiling

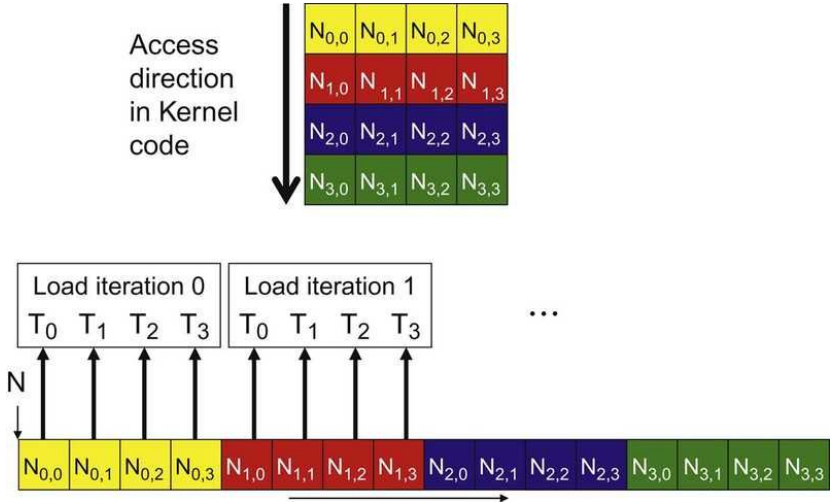
- Exploit GPU cores with tiles
- Exploit ALU with instruction-level parallelism



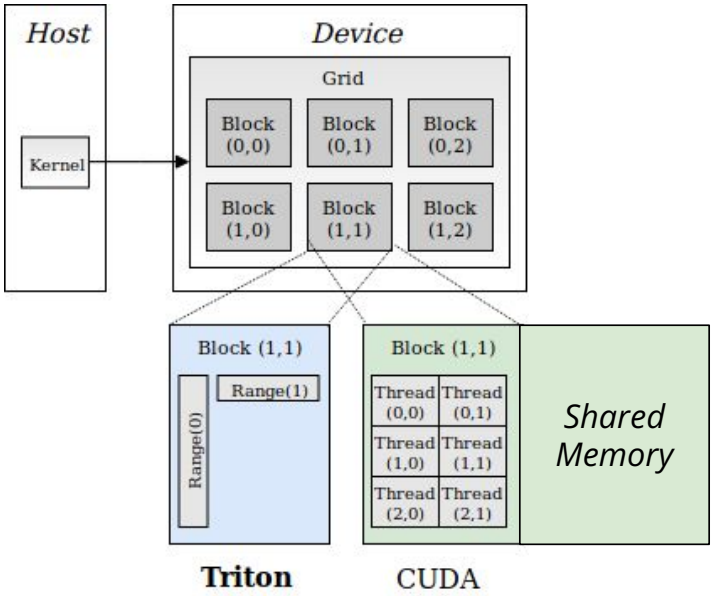


Machine Dependent Passes: Memory Coalescing

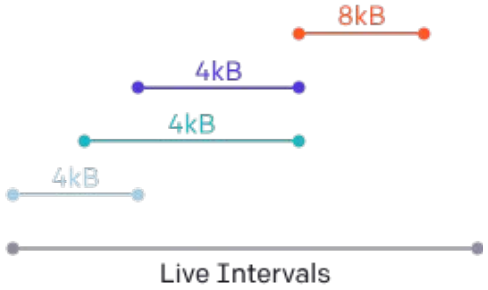


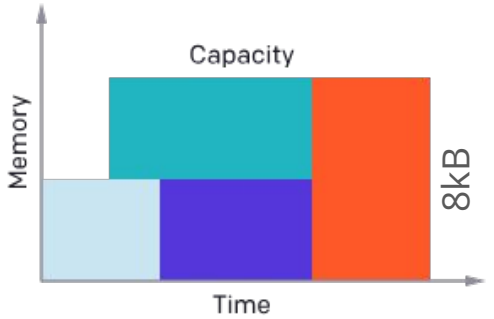


Machine Dependent Passes: Shared Memory Allocation

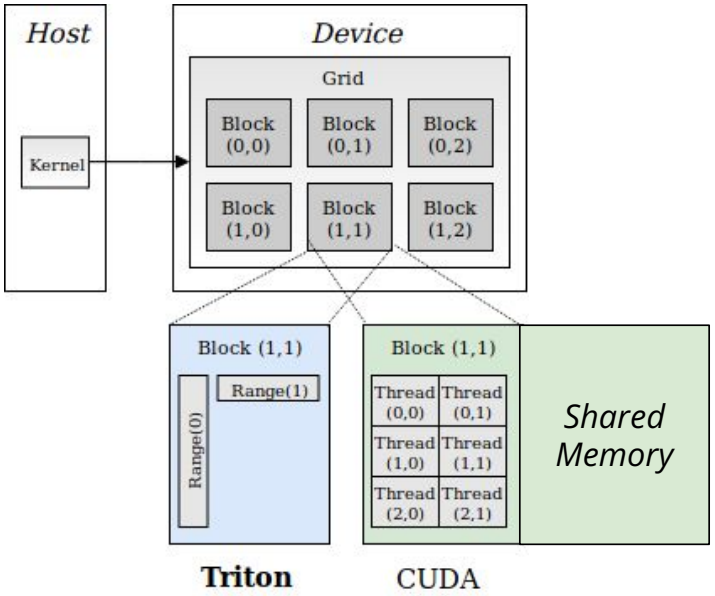



```
C = dot(A, B)

D = C + E

F = mul( D , G )

X = conv( Y , Z)
```

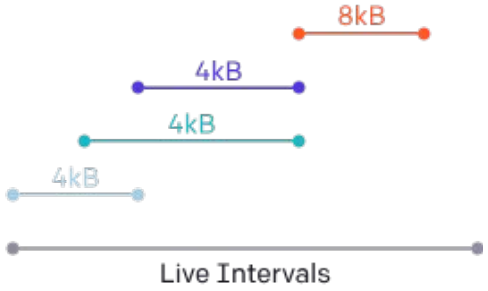


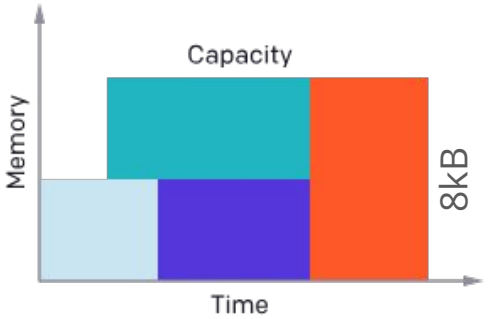


Machine Dependent Passes: Shared Memory Synchronization




```
C = dot(A, B)
synchronize()
D = C + E
synchronize()
F = mul( D , G )
synchronize()
X = conv( Y , Z)
```





Auto-tuner

- Hand-written parameterized code templates → optimization spaces extracted directly from Triton-IR programs
- Hierarchical tiling pass
  - Tile size (32-128 dim)
  - Micro-tile size (8-32 dim)
  - Nano-tile size (1-4 dim)

```
// Tile shapes are parametric and can be optimized
// by compilation backends
const tunable int TM  = {16, 32, 64, 128}
const tunable int TN  = {16, 32, 64, 128}
const tunable int TK  = {8, 16}
```

Results

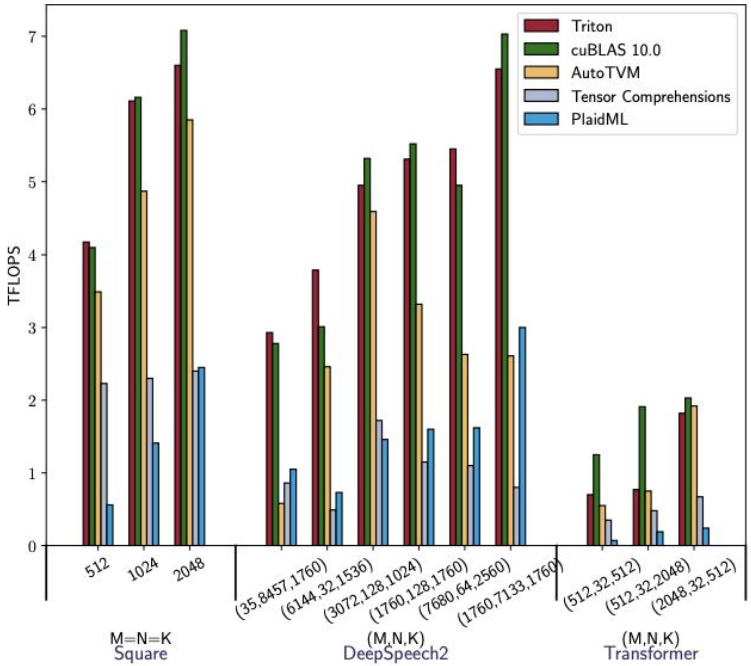


Figure 8. Performance of matrix multiplication

Results

Successfully re-implements cuDNN's general matrix multiplication (IMPLICIT_GEMM)

	H	W	C	B	K	R	S	Application
Task 1	112	112	64	4	128	3	3	ResNet [19]
Task 2	56	56	128	4	256	3	3	ResNet
Task 3	28	28	256	4	512	3	3	ResNet
Task 4	14	14	512	4	512	3	3	ResNet
Task 5	7	7	512	4	512	3	3	ResNet
Task 6	161	700	1	8	64	5	5	DeepSpeech2
Task 7	79	341	32	8	32	5	10	DeepSpeech2

Table 1. Convolution tasks considered in this paper

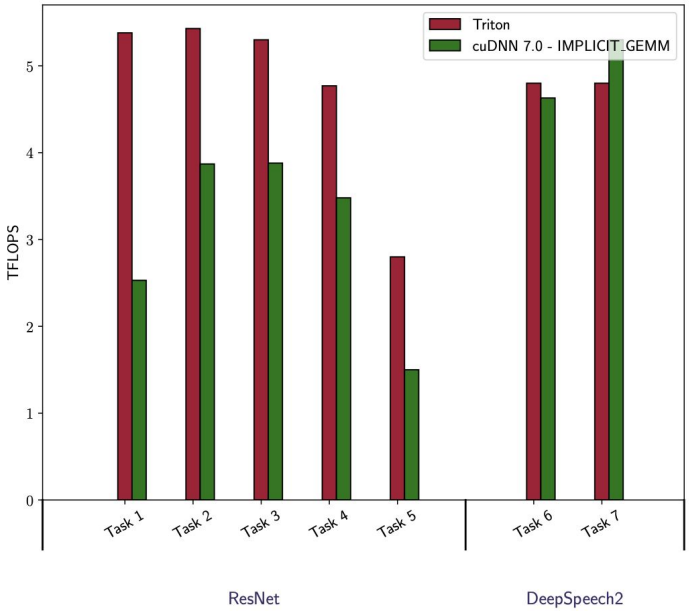


Figure 10. Performance of implicit matrix multiplication

Advantages

- Introduces a tile-centric programming model that mirrors how high-performance kernels are actually optimized
- Simplifies GPU programming by hiding complex details
- Allows non-CUDA experts to write custom GPU kernels
- Offers an expressive IR that preserves tensor structures

Weaknesses and Future Work

- Strongly geared towards dense, regular tensor computations
  - Might struggle on sparse or irregular workloads
- Perhaps unavoidably still requires GPU knowledge

- **Next Steps**
  - Additional algebraic tile optimizations (diagonal, block, triangular)
  - Tools for debugging and profiling Triton kernels

MICHIGAN ENGINEERING
UNIVERSITY OF MICHIGAN

Thank you for listening!

Any Questions?