MLIR Vector Distribution

Kunwar Grover
Harsh Menon
Jakub Kuderski

Structured Codegen

```
%D = linalg.generic {
    indexing_maps = [affine_map<(d0, d1) -> (d1, d0)>,
                     affine_map<(d0, d1) -> (d0, d1)>,
                     affine_map<(d0, d1) -> (d0, d1)>],
    iterator_types = ["parallel", "parallel"]
  }

    ins(%A, %B : tensor<32x64xf32>, tensor<64x32xf32>)
    outs(%C: tensor<64x32xf32>) {

  ^bb0(%a: f32, %b: f32, %c: f32):
       ...

} -> tensor<64x32xf32>
```

Structured Codegen

```
for in range(64):
  for in range(32):
    add = A[j, i] + B[i, j]
    C[i, j] = special_op(add)
```

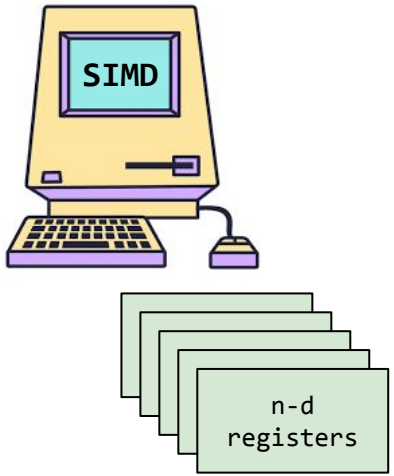
```
%D = linalg.generic {
    indexing_maps = [affine_map<(d0, d1) -> (d1, d0)>,
                     affine_map<(d0, d1) -> (d0, d1)>,
                     affine_map<(d0, d1) -> (d0, d1)>],
    iterator_types = ["parallel", "parallel"]
  }

    ins(%A, %B : tensor<32x64xf32>, tensor<64x32xf32>)
    outs(%C: tensor<64x32xf32>) {

  ^bb0(%a: f32, %b: f32, %c: f32):
      …

} -> tensor<64x32xf32>
```

Structured Codegen

```
C[i, j] = A[j, i] + B[i, j]
out = special_op(C[i, j])
```

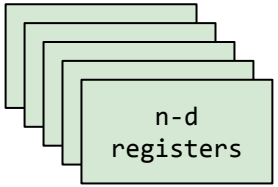
```
...
%r0 = vector.transfer_read %A_slice[0, 0] : vector<4x4xf32>
%r1 = vector.transfer_read %B_slice[0, 0] : vector<4x4xf32>
%trsp = vector.transpose %r0, [1, 0]      : vector<4x4xf32>
%add = arith.addf %trsp, %trsp            : vector<4x4xf32>
%out = special_op %add                    : vector<4x4xf32>
...
```

The SIMD Programming Model



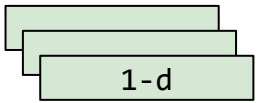

```
…
%r0 = vector.transfer_read %A_slice[0, 0] : vector<4x4xf32>
%r1 = vector.transfer_read %B_slice[0, 0] : vector<4x4xf32>
%trsp = vector.transpose %r0, [1, 0]      : vector<4x4xf32>
%add = arith.addf %trsp, %trsp            : vector<4x4xf32>
%out = special_op %add                    : vector<4x4xf32>
…
```

Different Vector Machines

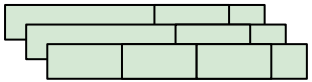


SIMD

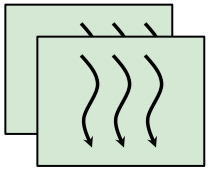
LLVM



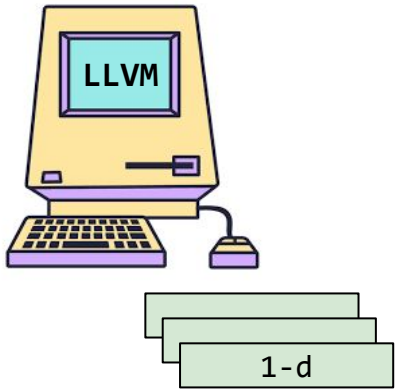
SPIRV



GPU



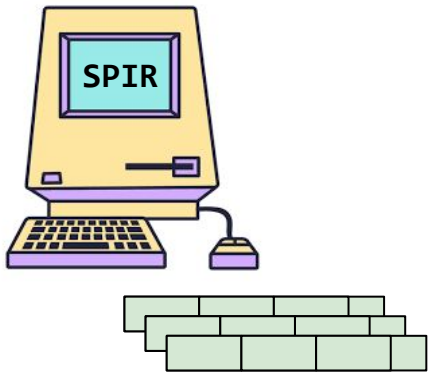
Lowering to M_{llvm} : Unrolling




```
%r0 = vector.transfer_read %A_slice[0, 0] : vector<4x4xf32>
```

```
%r0_0 = vector.load %A_slice[0, 0] : vector<4xf32>
%r0_1 = vector.load %A_slice[1, 0] : vector<4xf32>
%r0_2 = vector.load %A_slice[2, 0] : vector<4xf32>
%r0_3 = vector.load %A_slice[3, 0] : vector<4xf32>
```

Lowering to M_{spirv} : Unrolling?

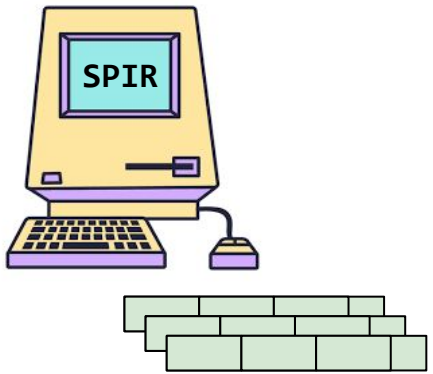


2, 3, 4 or custom vendor

```
…
%add = arith.addf %trsp, %r1 : vector<4x4xf32>
%out = special_op %add       : vector<4x4xf32>
…
```

```
…
%add_0 = arith.addf %trsp_0, %r1 : vector<4xf32>
%add_1 = arith.addf %trsp_0, %r1 : vector<4xf32>
…

?
```

Lowering to M_{spirv} : Unrolling?

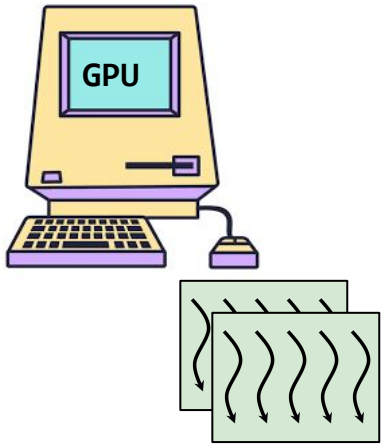


2, 3, 4 or custom vendor

```
…
%add = arith.addf %trsp, %r1 : vector<4x4xf32>
%out = vendor_specific %add  : vector<4x4xf32>
…
```

```
…
%add_0 = arith.addf %trsp_0, %r1 : vector<4xf32>
%add_1 = arith.addf %trsp_0, %r1 : vector<4xf32>
…

out    = vendor_specific %<...>  : vector<4x4xf32>
```

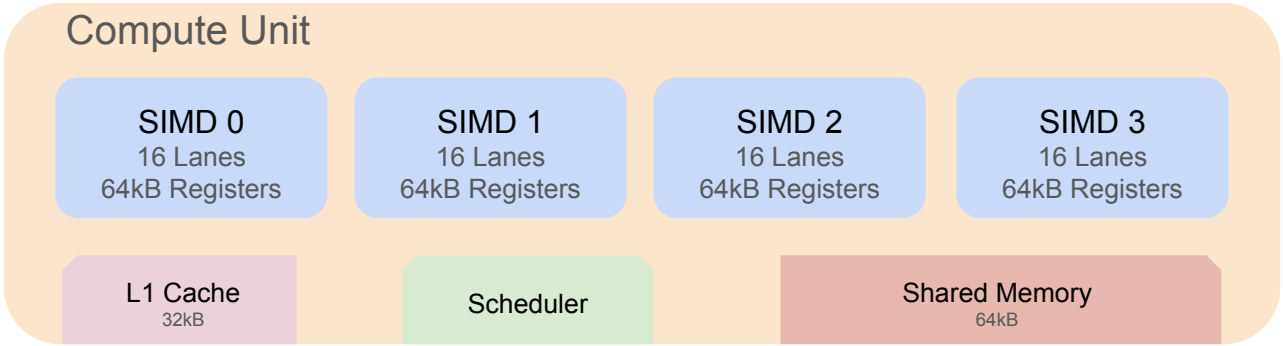
Lowering to M_{gpu}



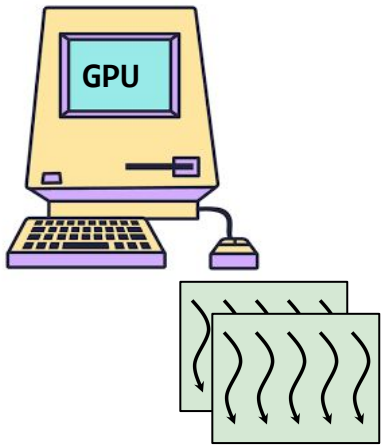
The GPU Programming Model: SIMT

The SIMT (Single Instruction Multiple Threads) execution model:

- Thread
- Subgroup
- Workgroup



Lowering to M_{gpu} : Distribution




```
…
%add = arith.addf %trsp, %r1 : vector<4x4xf32>
%out = vendor_specific %add  : vector<4x4xf32>
…
```

Lowering to M_{gpu} : Distribution

```
…
%add = arith.addf %trsp, %r1 : vector<4x4xf32>
%out = vendor_specific %add  : vector<4x4xf32>
…
```

%add:

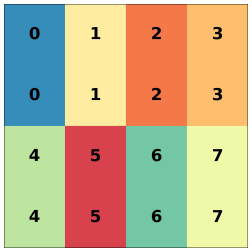
%trsp:

%r1:

Lowering to M_{gpu} : Distribution

```
…
%add = arith.addf %trsp, %r1 : vector<4x4xf32>
%out = vendor_specific %add  : vector<4x4xf32>
…
```

%add:



Layout

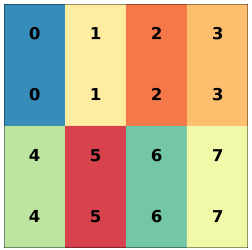
%trsp:

%r1:

Lowering to M_{gpu} : Distribution

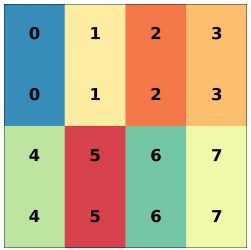
```
…
%add = arith.addf %trsp, %r1 : vector<4x4xf32>
%out = vendor_specific %add  : vector<4x4xf32>
…
```

%add:

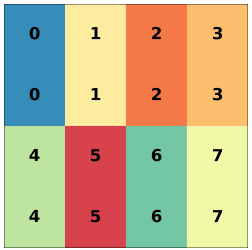


Layout

%trsp:



%r1:

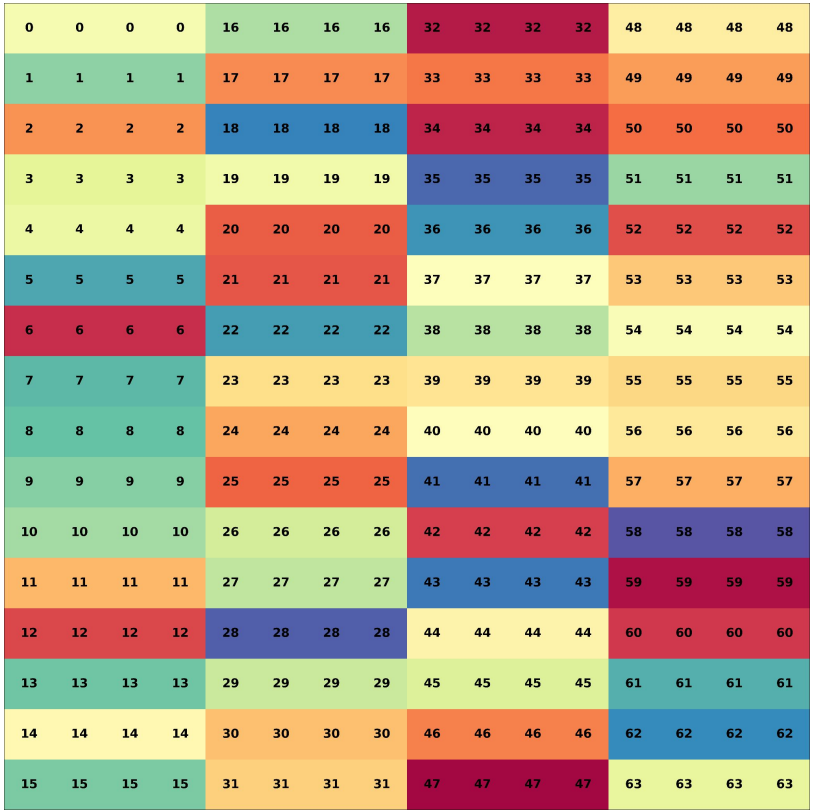


GPU Tensor Core Layouts: MFMA

```
C = A @ B
```

MFMA_16x16x16

Layout for B:

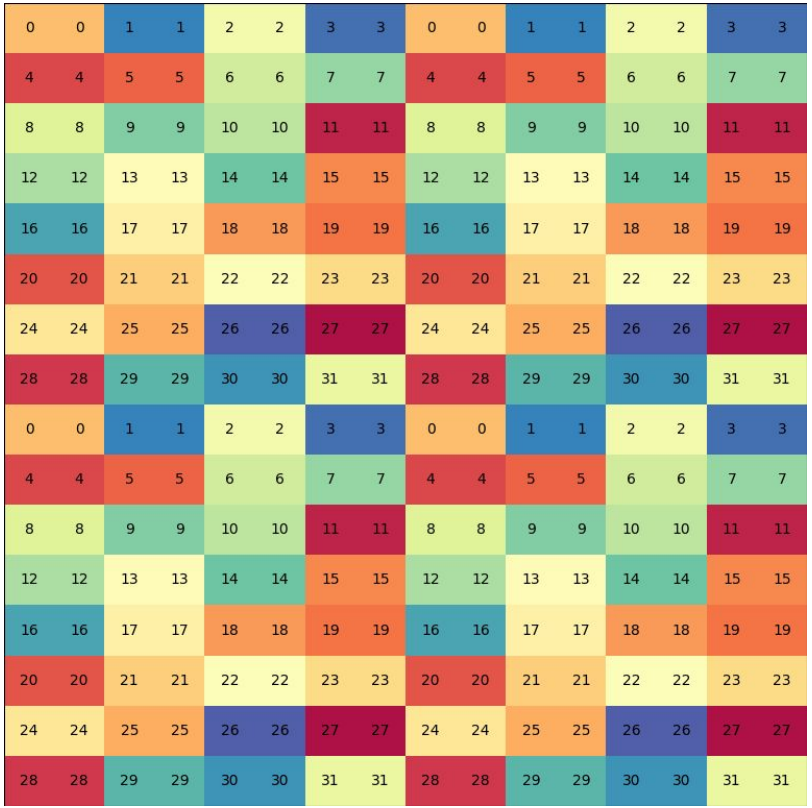


GPU Tensor Core Layouts: MMA.SYNC

```
C = A @ B
```

MMA_16x8x16

Layout for C:



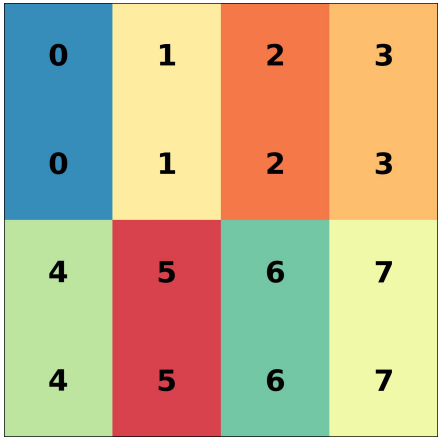
Need for layout-aware lowering

MLIR Upstream can distribute 1-D vectors: Greedy

Fails on multiple anchoring operations

Tensor core instructions are 2-D on a subgroup

An Interface for Layouts

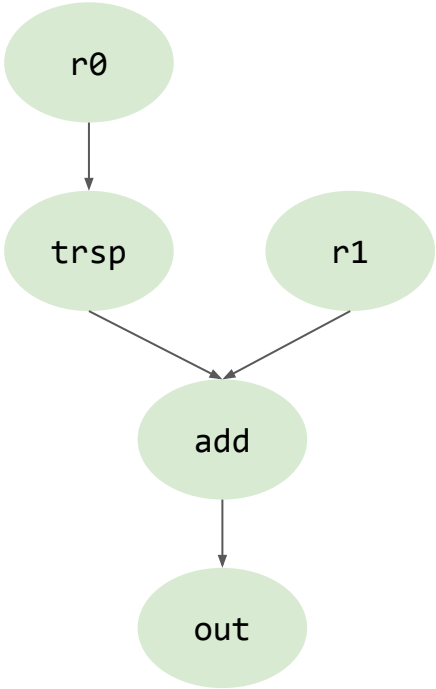


Attribute Interface for Layouts

```
A = #layout<...>

A.permute(0, 1)
A.project(0)
```

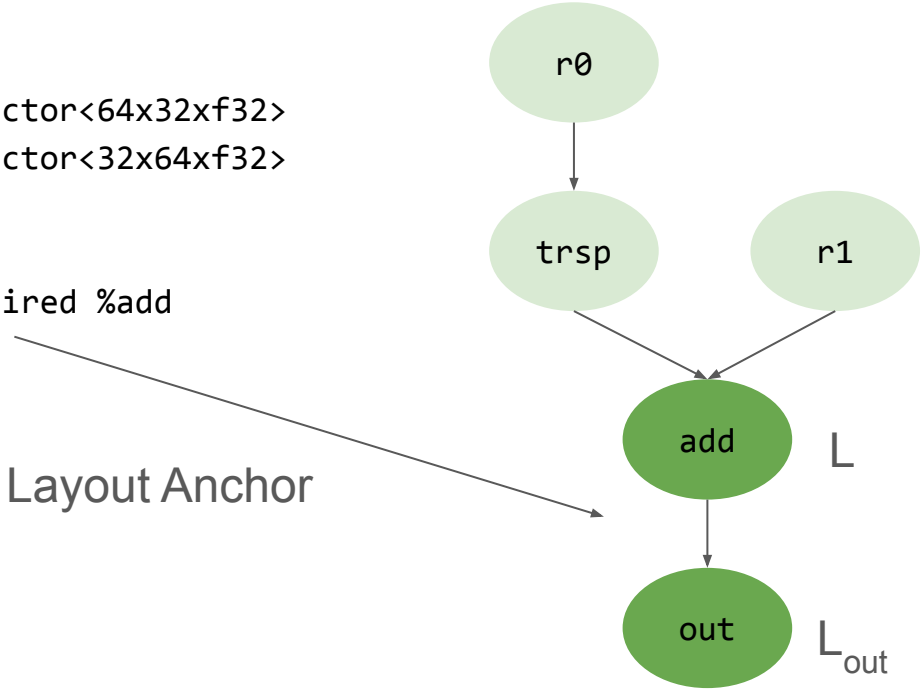
Vector Layout Analysis

```
%r0   = vector.transfer_read ... : vector<64x32xf32>
%r1   = vector.transfer_read ... : vector<32x64xf32>
%trsp = vector.transpose %r0, [1, 0]
%add  = arith.addf %trsp, %r1
%out  = specific_register_layout_required %add
```



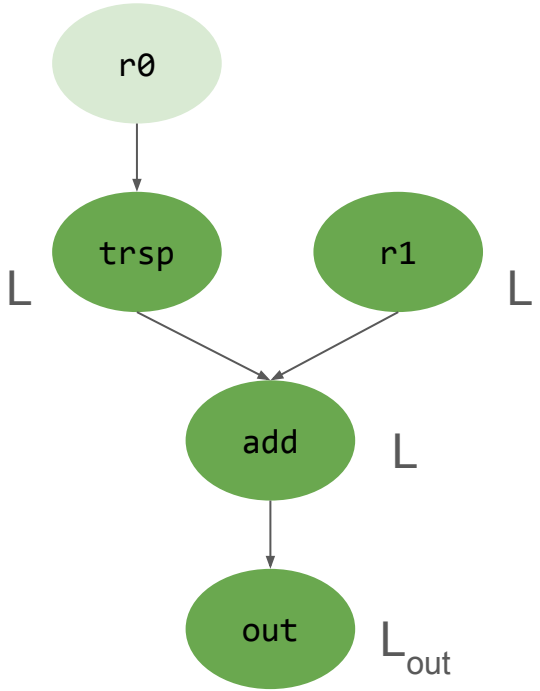
Vector Layout Analysis

```
%r0   = vector.transfer_read ... : vector<64x32xf32>
%r1   = vector.transfer_read ... : vector<32x64xf32>
%trsp = vector.transpose %r0, [1, 0]
%add  = arith.addf %trsp, %r1
%out  = specific_register_layout_required %add
```



Vector Layout Analysis

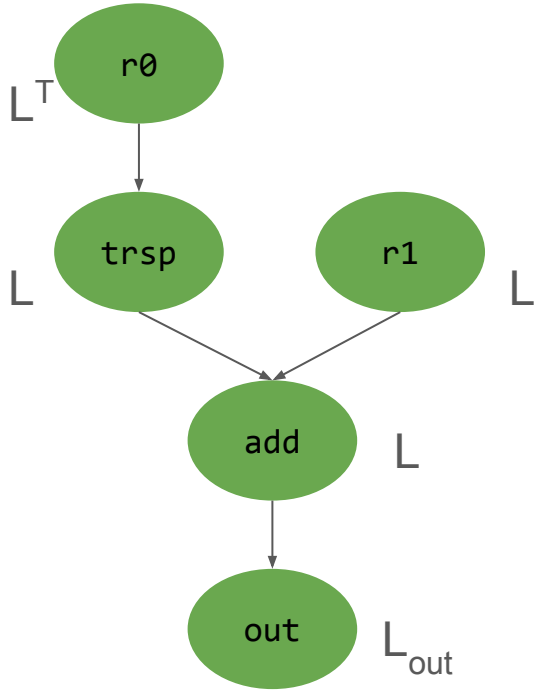
```
%r0   = vector.transfer_read ... : vector<64x32xf32>
%r1   = vector.transfer_read ... : vector<32x64xf32>
%trsp = vector.transpose %r0, [1, 0]
%add  = arith.addf %trsp, %r1
%out  = specific_register_layout_required %add
```



Analysis finds layouts for all vector values

Vector Layout Analysis

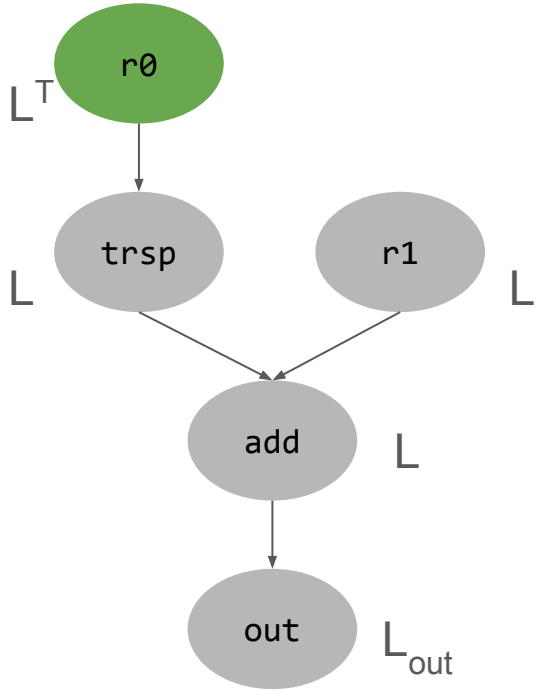
```
%r0   = vector.transfer_read ... : vector<64x32xf32>
%r1   = vector.transfer_read ... : vector<32x64xf32>
%trsp = vector.transpose %r0, [1, 0]
%add  = arith.addf %trsp, %r1
%out  = specific_register_layout_required %add
```



Analysis finds layouts for all vector values

Vector Layout Distribution

```
%r0   = vector.transfer_read ... : vector<64x32xf32>
```



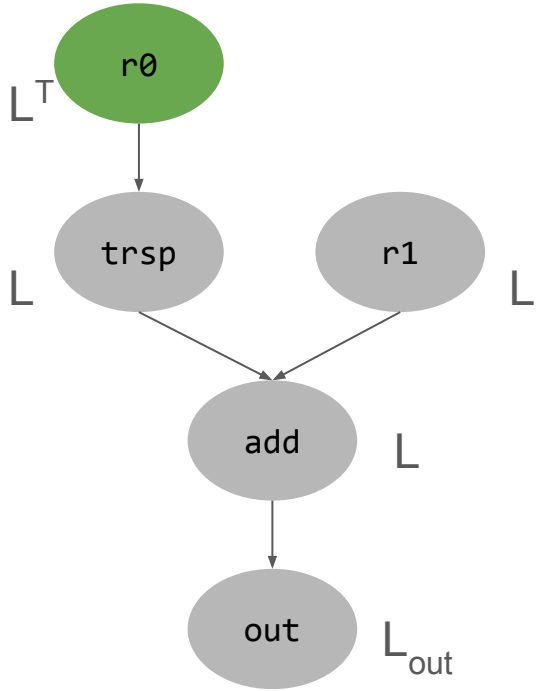
Vector Layout Distribution

```
%r0   = vector.transfer_read ... : vector<64x32xf32>
```



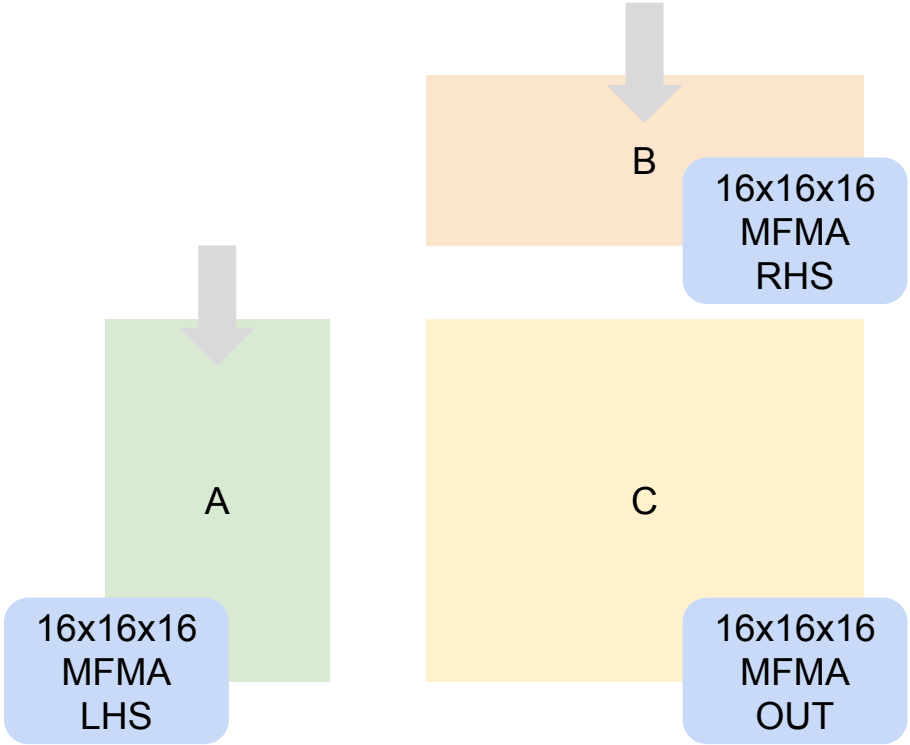
```
%r0   = layouted_read, L^T ... : vector<...xf32>
```

Example: `nvgpu.ld_matrix`



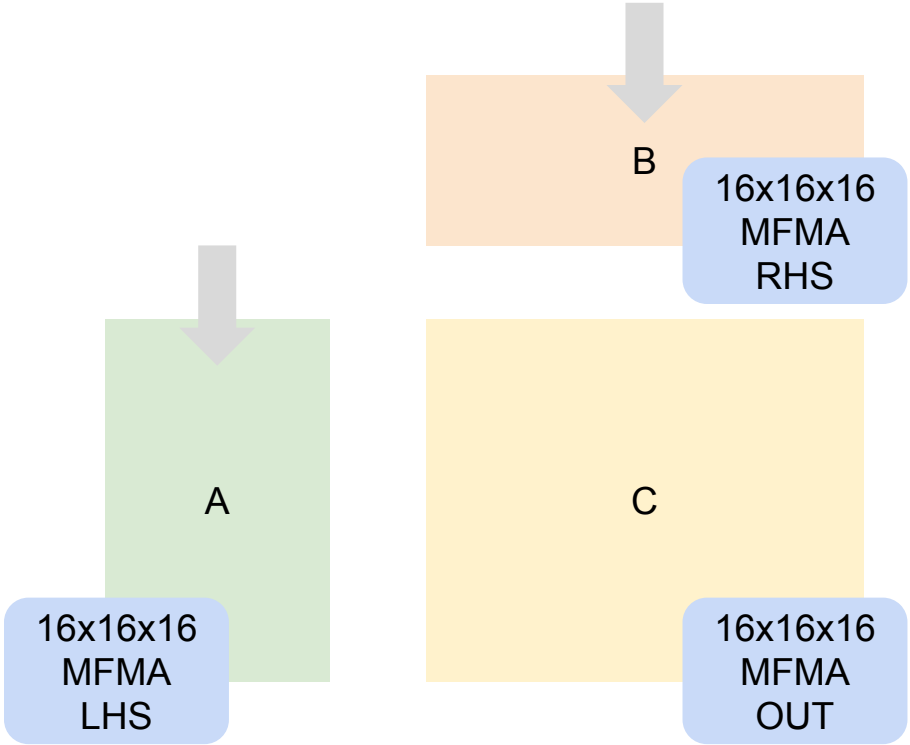
Layout Distribution In IREE

```
%A_slice = vector.transfer_read %A_shared[0, %i]
                        : vector<64x128xf16>
%B_slice = vector.transfer_read %B_shared[%i, 0]
                        : vector<128x64xf16>

%o = vector.contract #trait %A_slice, %B_slice, %acc
           : vector<64x128xf16>,
             vector<128x64xf16>
             into vector<64x64xf32>

scf.yield %o : vector<64x64xf32>
```

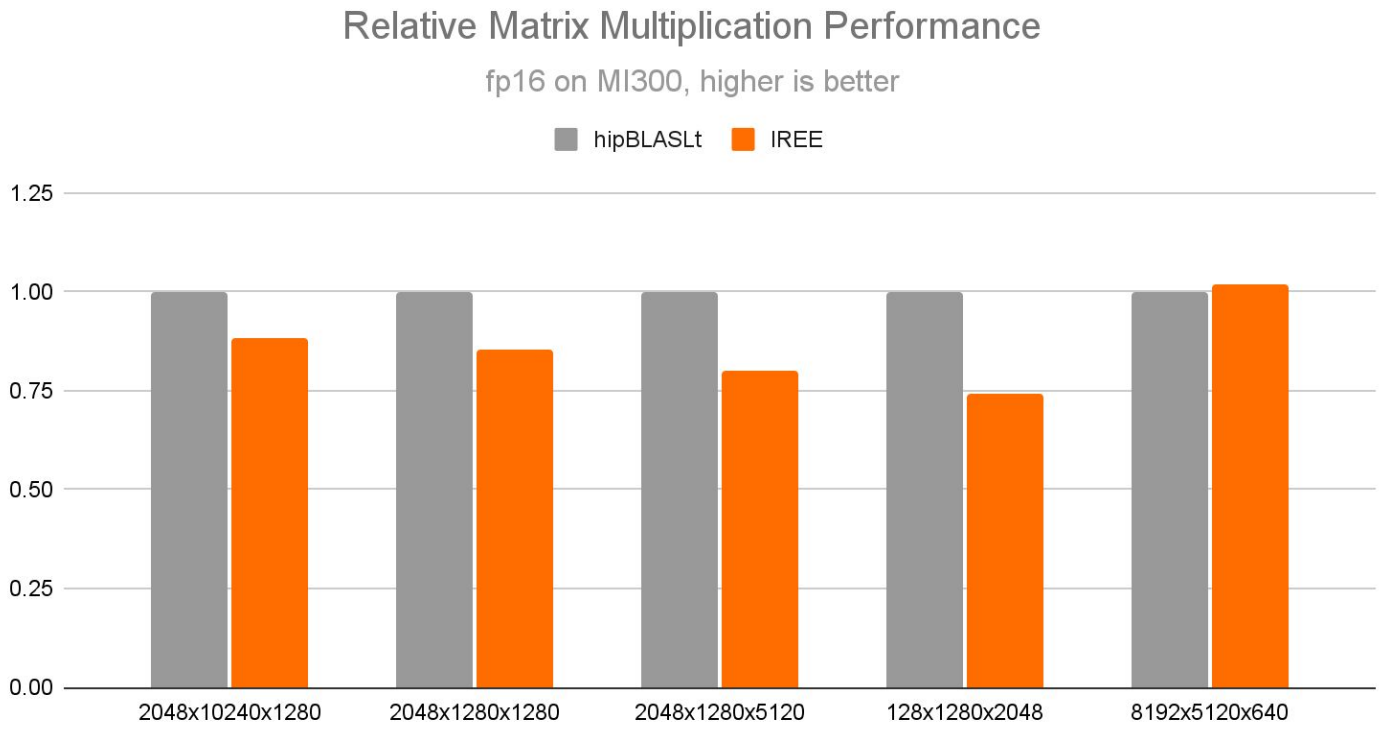


Layout Distribution

```
vector.transfer_read %A_shared ...
vector.transfer_read %A_shared ...

...

vector.transfer_read %B_shared ...
vector.transfer_read %B_shared ...

...

amdgpu.mfma MFMA_16x16x16 ...
amdgpu.mfma MFMA_16x16x16 ...

...
```

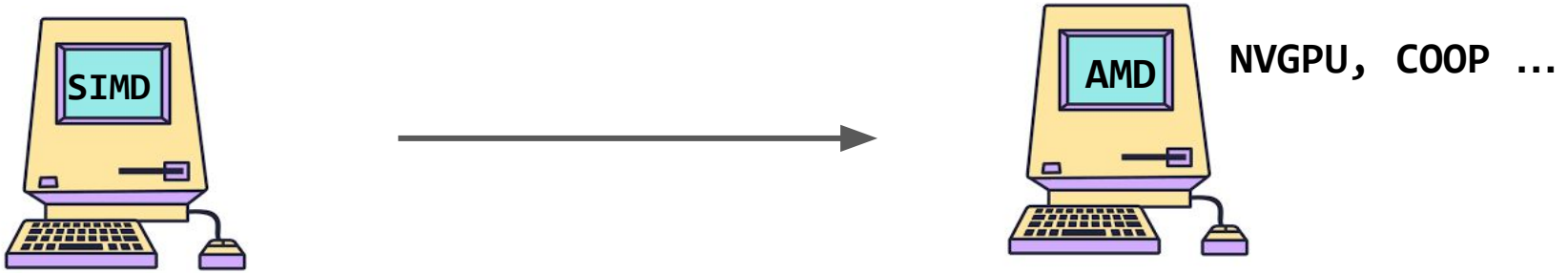


Benchmarks for Matmul



A General Framework

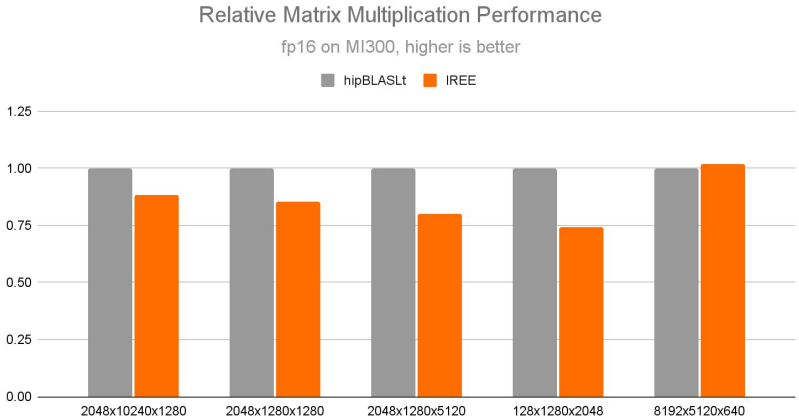
A General Framework

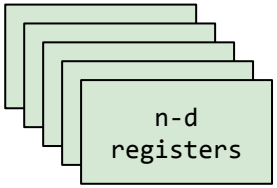


Related Work

- Triton
- Cooperative Matrix extensions

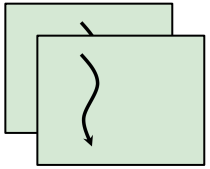
Conclusion



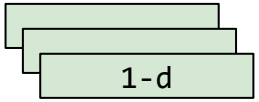


SIMD

GPU



LLVM



SPIRV