

# 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"]
}

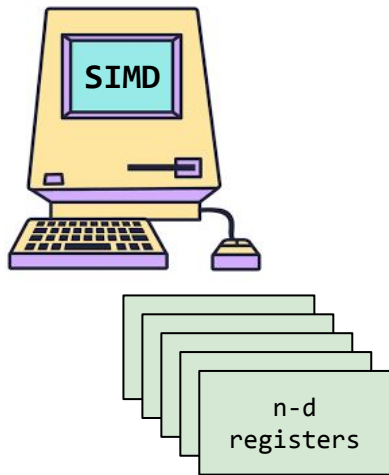
ins(%A, %B : tensor<32x64xf32>, tensor<64x32xf32>)
outs(%C: tensor<64x32xf32>) {
  ^bb0(%a: f32, %b: f32, %c: f32):
    ...
} -> tensor<64x32xf32>
```

# Structured Codegen

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

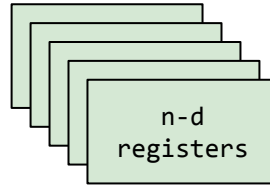
```
...  
%r0 = vector.transfer_read %A_slice[0, 0] : vector<4x4xf32>  
%r1 = vector.transfer_read %B_slice[0, 0] : vector<4x4xf32>  
%trsp = vector.transpose %r0, [1, 0] : vector<4x4xf32>  
%add = arith.addf %trsp, %trsp : vector<4x4xf32>  
%out = special_op %add : vector<4x4xf32>  
...
```

# The SIMD Programming Model



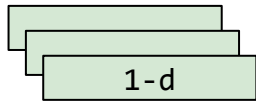
```
...  
%r0 = vector.transfer_read %A_slice[0, 0] : vector<4x4xf32>  
%r1 = vector.transfer_read %B_slice[0, 0] : vector<4x4xf32>  
%trsp = vector.transpose %r0, [1, 0] : vector<4x4xf32>  
%add = arith.addf %trsp, %trsp : vector<4x4xf32>  
%out = special_op %add : vector<4x4xf32>  
...
```

# Different Vector Machines

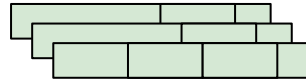


SIMD

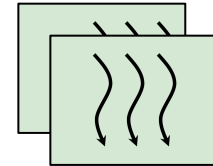
LLVM



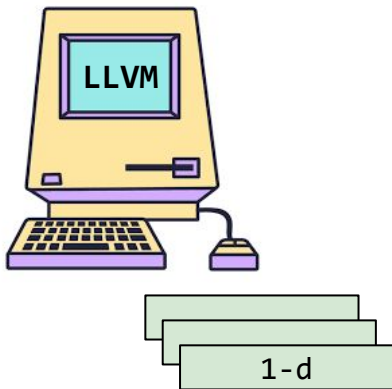
SPIRV



GPU



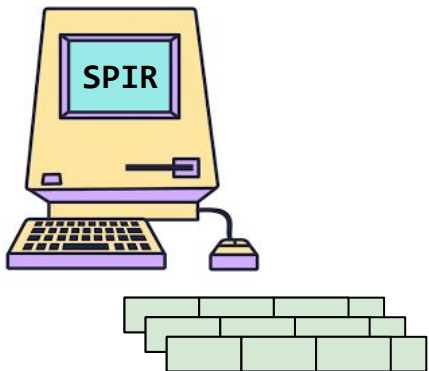
# Lowering to $M_{llvm}$ : Unrolling



```
%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 $M_{\text{spirv}}$ : Unrolling?



2, 3, 4 or custom vendor

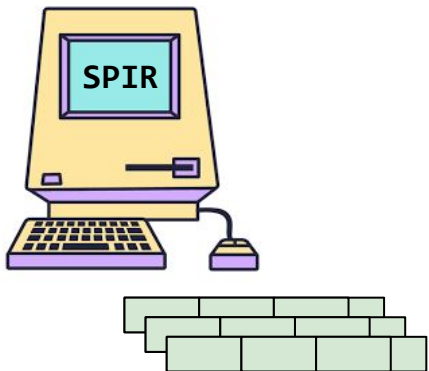
```
...  
%add = arith.addf %trsp, %r1 : vector<4x4xf32>  
%out = special_op %add      : vector<4x4xf32>  
...
```

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

?



# Lowering to $M_{\text{spirv}}$ : Unrolling?



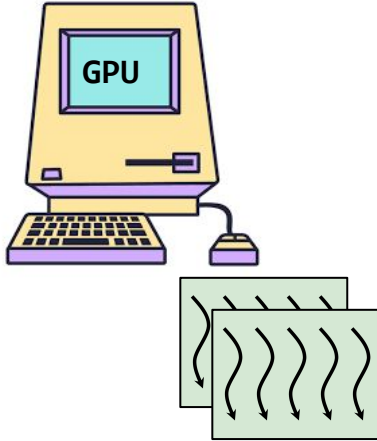
2, 3, 4 or custom vendor

```
...
%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>
```

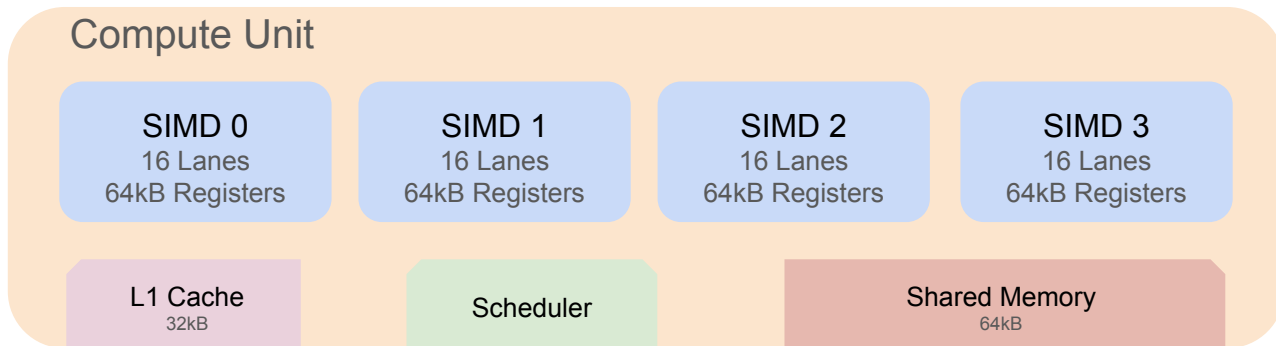
Lowering to  $M_{\text{gpu}}$



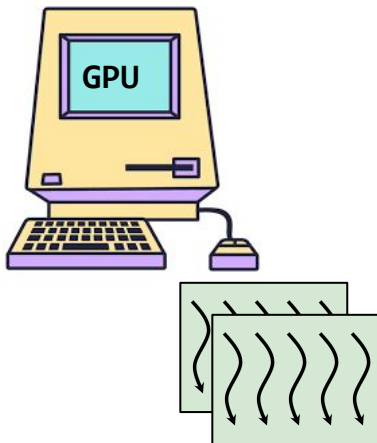
# The GPU Programming Model: SIMT

The SIMT (Single Instruction Multiple Threads) execution model:

- Thread
- Subgroup
- Workgroup



# Lowering to $M_{\text{gpu}}$ : Distribution



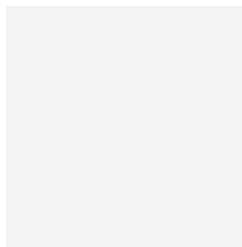
```
...  
%add = arith.addf %trsp, %r1 : vector<4x4xf32>  
%out = vendor_specific %add : vector<4x4xf32>  
...
```

# Lowering to $M_{\text{gpu}}$ : Distribution

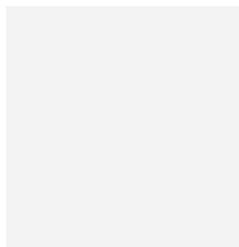


```
...  
%add = arith.addf %trsp, %r1 : vector<4x4xf32>  
%out = vendor_specific %add : vector<4x4xf32>  
...
```

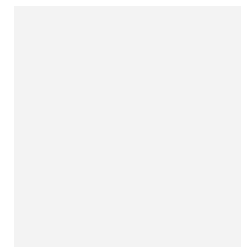
%add:



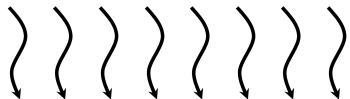
%trsp:



%r1:



# Lowering to $M_{\text{gpu}}$ : Distribution

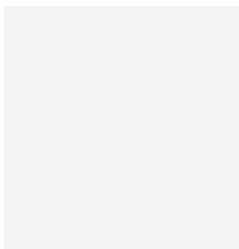


```
...
%add = arith.addf %trsp, %r1 : vector<4x4xf32>
%out = vendor_specific %add : vector<4x4xf32>
...
```

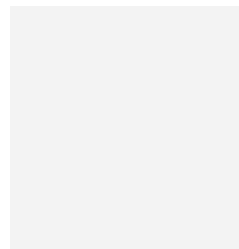
%add:

0	1	2	3
0	1	2	3
4	5	6	7
4	5	6	7

%trsp:

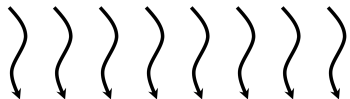


%r1:



Layout

# Lowering to $M_{\text{gpu}}$ : Distribution



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

%add:

0	1	2	3
0	1	2	3
4	5	6	7
4	5	6	7

%trsp:

0	1	2	3
0	1	2	3
4	5	6	7
4	5	6	7

%r1:

0	1	2	3
0	1	2	3
4	5	6	7
4	5	6	7

Layout

# GPU Tensor Core Layouts: MFMA

$C = A @ B$

MFMA\_16x16x16

Layout for B:

0	0	0	0	16	16	16	16	32	32	32	32	48	48	48	48
1	1	1	1	17	17	17	17	33	33	33	33	49	49	49	49
2	2	2	2	18	18	18	18	34	34	34	34	50	50	50	50
3	3	3	3	19	19	19	19	35	35	35	35	51	51	51	51
4	4	4	4	20	20	20	20	36	36	36	36	52	52	52	52
5	5	5	5	21	21	21	21	37	37	37	37	53	53	53	53
6	6	6	6	22	22	22	22	38	38	38	38	54	54	54	54
7	7	7	7	23	23	23	23	39	39	39	39	55	55	55	55
8	8	8	8	24	24	24	24	40	40	40	40	56	56	56	56
9	9	9	9	25	25	25	25	41	41	41	41	57	57	57	57
10	10	10	10	26	26	26	26	42	42	42	42	58	58	58	58
11	11	11	11	27	27	27	27	43	43	43	43	59	59	59	59
12	12	12	12	28	28	28	28	44	44	44	44	60	60	60	60
13	13	13	13	29	29	29	29	45	45	45	45	61	61	61	61
14	14	14	14	30	30	30	30	46	46	46	46	62	62	62	62
15	15	15	15	31	31	31	31	47	47	47	47	63	63	63	63



# GPU Tensor Core Layouts: MMA.SYNC

$C = A @ B$

MMA\_16x8x16

Layout for C:

0	0	1	1	2	2	3	3	0	0	1	1	2	2	3	3
4	4	5	5	6	6	7	7	4	4	5	5	6	6	7	7
8	8	9	9	10	10	11	11	8	8	9	9	10	10	11	11
12	12	13	13	14	14	15	15	12	12	13	13	14	14	15	15
16	16	17	17	18	18	19	19	16	16	17	17	18	18	19	19
20	20	21	21	22	22	23	23	20	20	21	21	22	22	23	23
24	24	25	25	26	26	27	27	24	24	25	25	26	26	27	27
28	28	29	29	30	30	31	31	28	28	29	29	30	30	31	31
0	0	1	1	2	2	3	3	0	0	1	1	2	2	3	3
4	4	5	5	6	6	7	7	4	4	5	5	6	6	7	7
8	8	9	9	10	10	11	11	8	8	9	9	10	10	11	11
12	12	13	13	14	14	15	15	12	12	13	13	14	14	15	15
16	16	17	17	18	18	19	19	16	16	17	17	18	18	19	19
20	20	21	21	22	22	23	23	20	20	21	21	22	22	23	23
24	24	25	25	26	26	27	27	24	24	25	25	26	26	27	27
28	28	29	29	30	30	31	31	28	28	29	29	30	30	31	31

# 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

0	1	2	3
0	1	2	3
4	5	6	7
4	5	6	7

Attribute Interface for Layouts

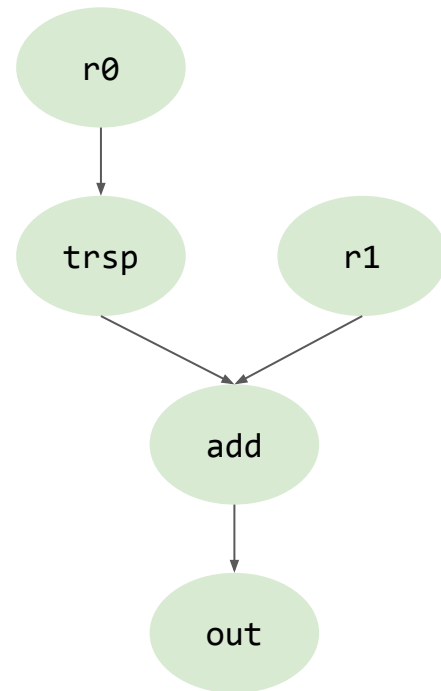
```
A = #layout<...>
```

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

```
A.project(0)
```

# Vector Layout Analysis

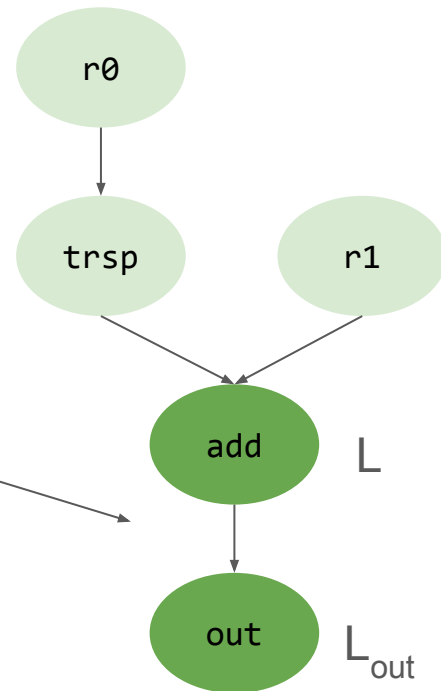
```
%r0  = vector.transfer_read ... : vector<64x32xf32>  
%r1  = vector.transfer_read ... : vector<32x64xf32>  
%trsp = vector.transpose %r0, [1, 0]  
%add  = arith.addf %trsp, %r1  
%out  = specific_register_layout_required %add
```



# 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
```

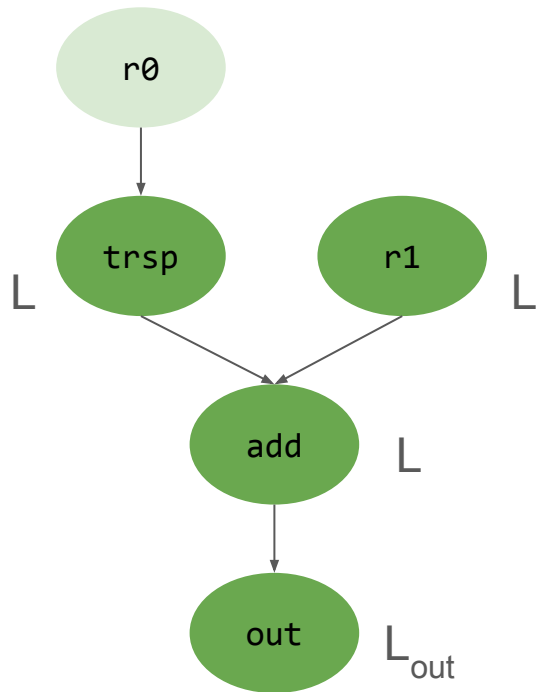
Layout Anchor



# 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
```

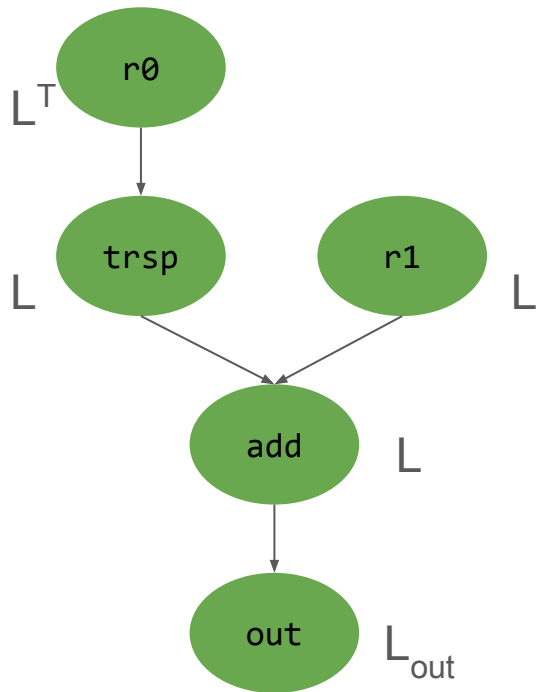
Analysis finds layouts for all vector values



# Vector Layout Analysis

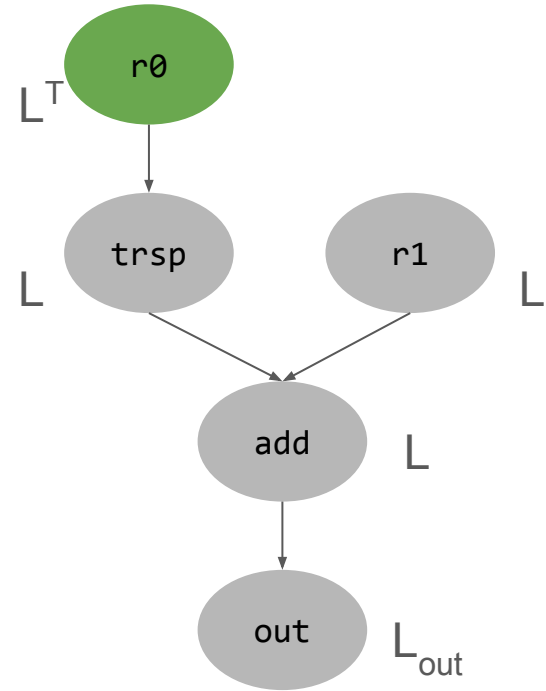
```
%r0  = vector.transfer_read ... : vector<64x32xf32>  
%r1  = vector.transfer_read ... : vector<32x64xf32>  
%trsp = vector.transpose %r0, [1, 0]  
%add  = arith.addf %trsp, %r1  
%out  = specific_register_layout_required %add
```

Analysis finds layouts for all vector values



# Vector Layout Distribution

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





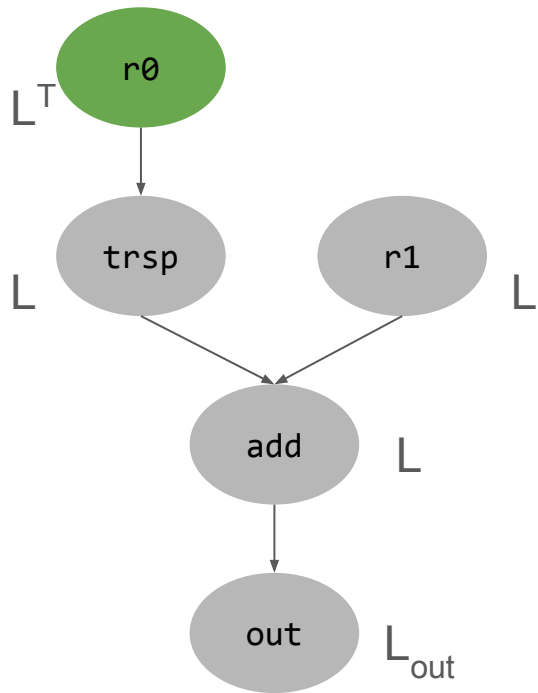
# Vector Layout Distribution

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



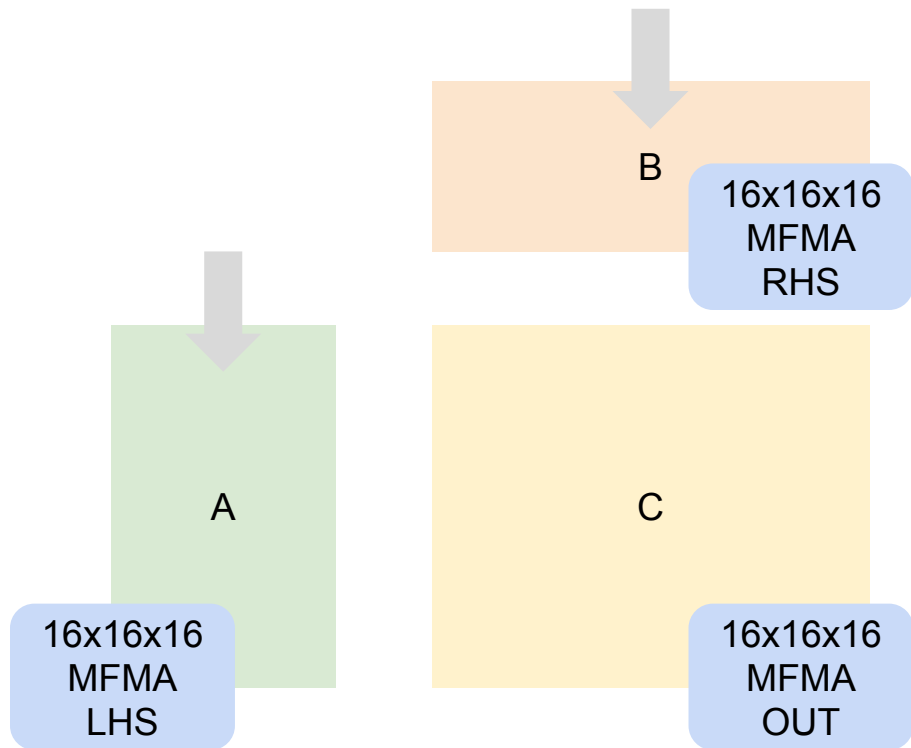
`%r0 = layouted_read,  $L^T$  ... : vector<...xf32>`

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 ...
```

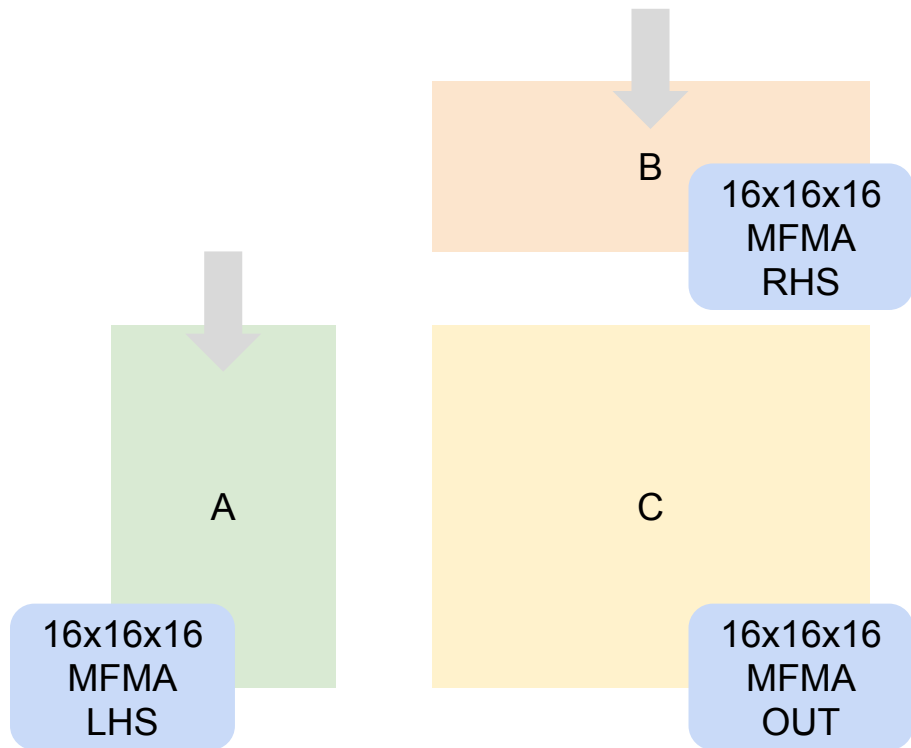
...

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

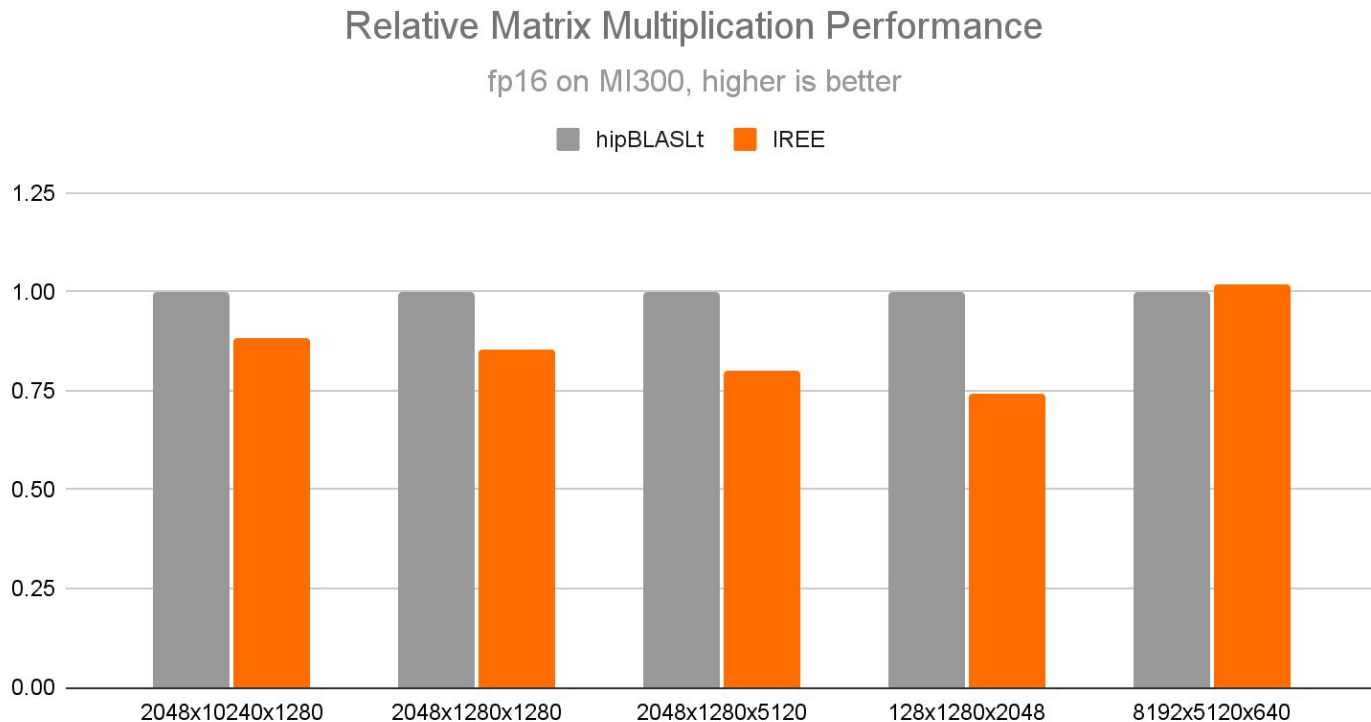
...

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

...



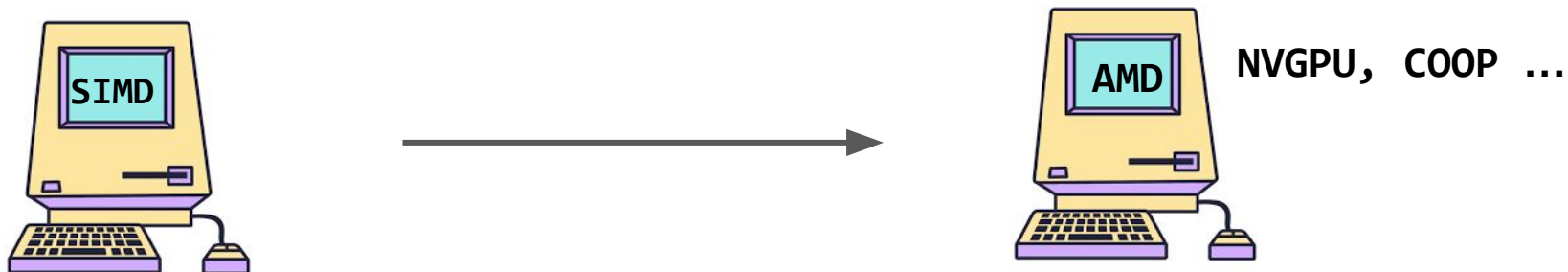
# Benchmarks for Matmul



# A General Framework



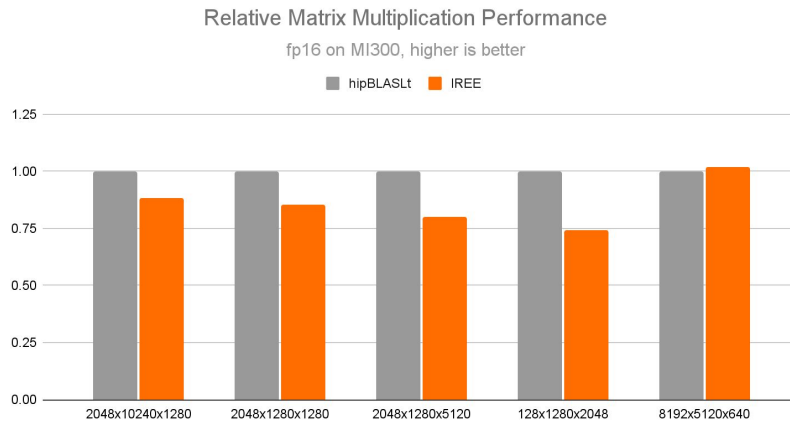
# A General Framework



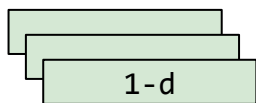
## Related Work

- Triton
- Cooperative Matrix extensions

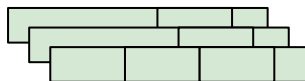
# Conclusion



LLVM



SPIRV



SIMD



GPU

