Analyzing MLIR GPU Dialect for Neural Network Workloads and Proposing Optimizations

KV Tuhil, Aditya Garg, Anirudh Saikrishnan

November 2025

1 Introduction

Deep Neural Networks (DNNs) like ResNet-50 rely on GPUs for intensive tensor computations, a process increasingly managed by the LLVM-based MLIR framework and its hardware-aware GPU dialect. While MLIR effectively handles abstractions such as thread mapping and memory hierarchy to generate CUDA or ROCm kernels, there remains significant potential to improve how these workloads are compiled. This project focuses on analyzing the MLIR GPU dialect generated for ResNet-50 to identify specific performance bottlenecks such as inefficiencies in synchronization or memory usage and proposes novel optimization strategies to enhance execution efficiency.

2 Overview

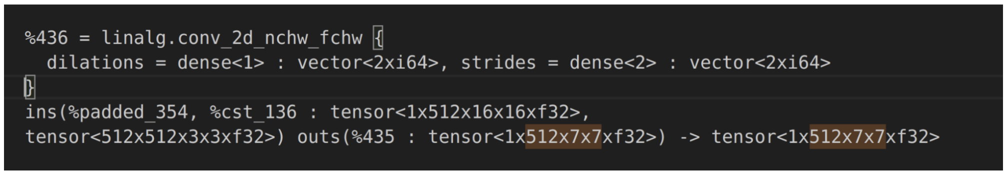
The project workflow proceeded through the following stages:

1. **Investigation of MLIR Dialects:** We investigated various MLIR dialects, specifically focusing on the SCF (Structured Control Flow) parallel and GPU dialects. The primary objective was to identify opportunities to improve thread block mapping, memory coalescing, and tiling strategies.
2. **Analysis of Lowering Pipeline:** We analyzed the dialect lowering process across various passes, utilizing a convolution kernel as a representative example to trace the transformation of the intermediate representation.
3. **Modification of Standard Passes:** We modified the `ParallelLoopPass` to implement an optimized thread mapping policy, aimed at increasing GPU occupancy.
4. **Integration with IREE:** We transitioned to the IREE compiler infrastructure to validate the modified pass. However, we discovered that IREE utilizes its own custom lowering passes and effectively bypasses the standard SCF parallel dialect.
5. **Optimization and Results:** Consequently, we shifted focus to the IREE `LLVMGPU` kernel configuration. By optimizing tile sizes and thread mapping parameters, we successfully reduced the kernel runtime from 19.7 ms to 6.84 ms.

3 MLIR Dialect Walkthrough

To analyze the lowering pipeline from high-level IR to the GPU dialect and finally to LLVM, we utilized a convolution kernel as a representative benchmark.

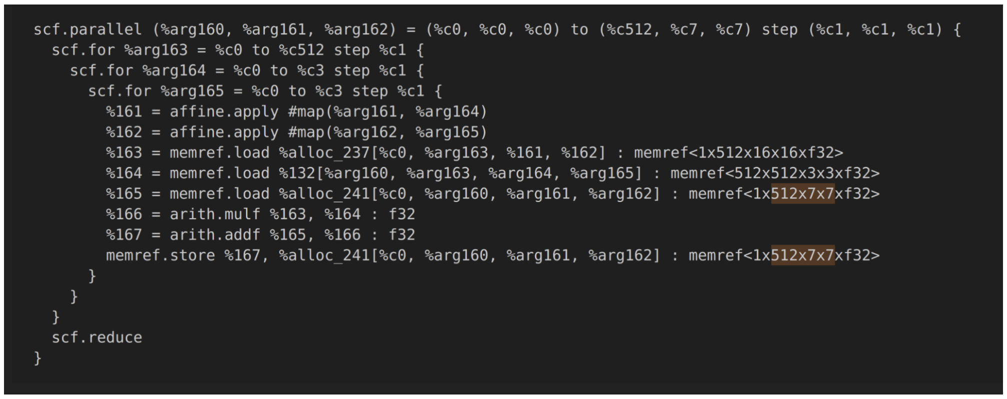
- **Linalg Dialect:** The process begins with a convolution operation in the `linalg` dialect. This operation accepts a $512 \times 16 \times 16$ input matrix and a $512 \times 3 \times 3$ kernel, producing a $512 \times 7 \times 7$ output matrix.



```
%436 = linalg.conv_2d_nchw_fchw {
  dilations = dense<1> : vector<2xi64>, strides = dense<2> : vector<2xi64>
}
ins(%padded_354, %cst_136 : tensor<1x512x16x16xf32>,
tensor<512x512x3x3xf32>) outs(%435 : tensor<1x512x7x7xf32>) -> tensor<1x512x7x7xf32>
```

Figure 1: Snippet of the Convolution operation in the Linalg dialect.

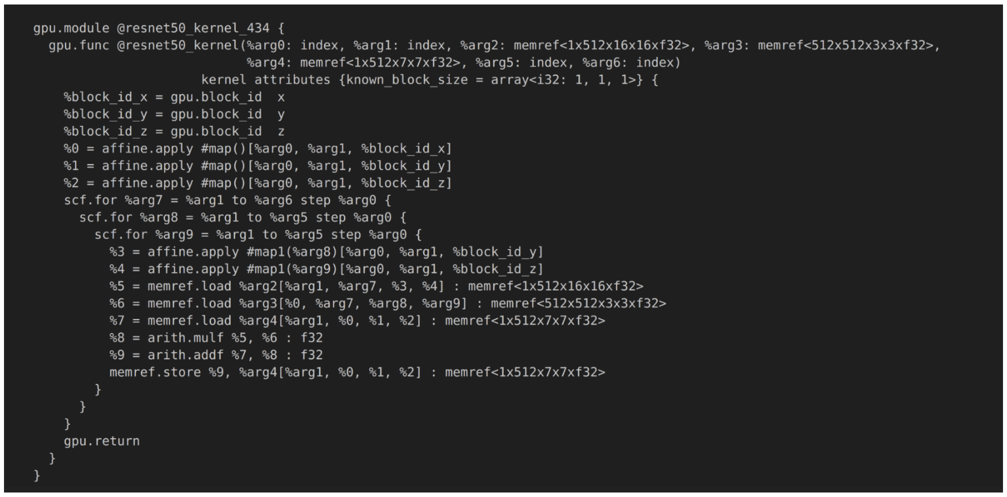
- **Lowering to SCF Parallel:** Subsequent lowering to the `scf` (Structured Control Flow) parallel dialect results in nested loops where the first three loops are parallelized, while the remainder are sequential. Notably, the step size for the parallel loops is set to 1. This default configuration restricts the thread block size to 1, which severely limits GPU occupancy.



```
scf.parallel (%arg160, %arg161, %arg162) = (%c0, %c0, %c0) to (%c512, %c7, %c7) step (%c1, %c1, %c1) {
  scf.for %arg163 = %c0 to %c512 step %c1 {
    scf.for %arg164 = %c0 to %c3 step %c1 {
      scf.for %arg165 = %c0 to %c3 step %c1 {
        %161 = affine.apply #map(%arg161, %arg164)
        %162 = affine.apply #map(%arg162, %arg165)
        %163 = memref.load %alloc_237[%c0, %arg163, %161, %162] : memref<1x512x16x16xf32>
        %164 = memref.load %132[%arg160, %arg163, %arg164, %arg165] : memref<512x512x3x3xf32>
        %165 = memref.load %alloc_241[%c0, %arg160, %arg161, %arg162] : memref<1x512x7x7xf32>
        %166 = arith.mulf %163, %164 : f32
        %167 = arith.addf %165, %166 : f32
        memref.store %167, %alloc_241[%c0, %arg160, %arg161, %arg162] : memref<1x512x7x7xf32>
      }
    }
  }
  scf.reduce
}
```

Figure 2: Lowered SCF Parallel loops showing unit step sizes.

- **GPU Kernel Mapping:** This IR is mapped to a GPU kernel launch configuration. As shown in the snippet below, the resulting block size is $\{1, 1, 1\}$. This configuration prevents memory coalescing and fails to utilize the GPU's parallelism effectively.

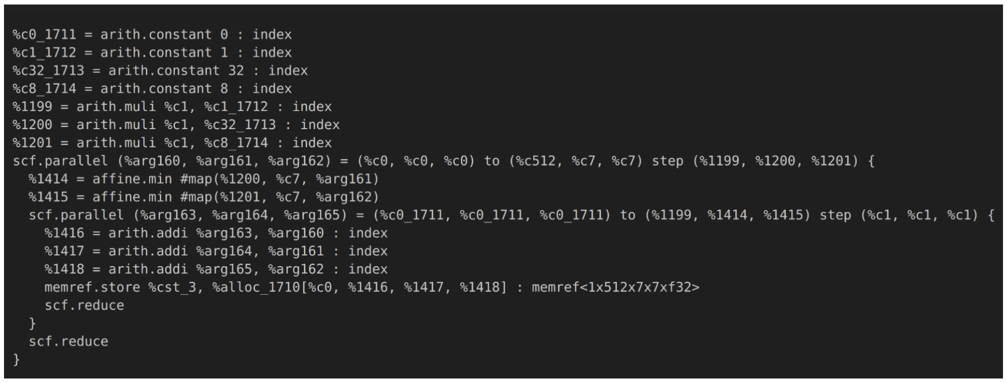


```
gpu.module @resnet50_kernel_434 {
  gpu.func @resnet50_kernel(%arg0: index, %arg1: index, %arg2: memref<1x512x16x16xf32>, %arg3: memref<512x512x3x3xf32>,
                            %arg4: memref<1x512x7x7xf32>, %arg5: index, %arg6: index)
                      kernel attributes {known_block_size = array<i32: 1, 1, 1>} {
    %block_id_x = gpu.block_id  x
    %block_id_y = gpu.block_id  y
    %block_id_z = gpu.block_id  z
    %0 = affine.apply #map()[%arg0, %arg1, %block_id_x]
    %1 = affine.apply #map()[%arg0, %arg1, %block_id_y]
    %2 = affine.apply #map()[%arg0, %arg1, %block_id_z]
    scf.for %arg7 = %arg1 to %arg6 step %arg0 {
      scf.for %arg8 = %arg1 to %arg5 step %arg0 {
        scf.for %arg9 = %arg1 to %arg5 step %arg0 {
          %3 = affine.apply #map1(%arg8)[%arg0, %arg1, %block_id_y]
          %4 = affine.apply #map1(%arg9)[%arg0, %arg1, %block_id_z]
          %5 = memref.load %arg2[%arg1, %arg7, %3, %4] : memref<1x512x16x16xf32>
          %6 = memref.load %arg3[%0, %arg7, %arg8, %arg9] : memref<512x512x3x3xf32>
          %7 = memref.load %arg4[%arg1, %0, %1, %2] : memref<1x512x7x7xf32>
          %8 = arith.mulf %5, %6 : f32
          %9 = arith.addf %7, %8 : f32
          memref.store %9, %arg4[%arg1, %0, %1, %2] : memref<1x512x7x7xf32>
        }
      }
    }
    gpu.return
  }
}
```

Figure 3: Generated GPU Kernel with inefficient block size mapping.

- **Optimization Strategy:** To address this bottleneck, we modified the mapping policy within the `ParallelLoopPass`. We implemented a greedy strategy to maximize thread utilization.
The approach is defined as follows:

– Tuhil please write it



```
%c0_1711 = arith.constant 0 : index
%c1_1712 = arith.constant 1 : index
%c32_1713 = arith.constant 32 : index
%c8_1714 = arith.constant 8 : index
%1199 = arith.muli %c1, %c1_1712 : index
%1200 = arith.muli %c1, %c32_1713 : index
%1201 = arith.muli %c1, %c8_1714 : index
scf.parallel (%arg160, %arg161, %arg162) = (%c0, %c0, %c0) to (%c512, %c7, %c7) step (%1199, %1200, %1201) {
  %1414 = affine.min #map(%1200, %c7, %arg161)
  %1415 = affine.min #map(%1201, %c7, %arg162)
  scf.parallel (%arg163, %arg164, %arg165) = (%c0_1711, %c0_1711, %c0_1711) to (%1199, %1414, %1415) step (%c1, %c1, %c1) {
    %1416 = arith.addi %arg163, %arg160 : index
    %1417 = arith.addi %arg164, %arg161 : index
    %1418 = arith.addi %arg165, %arg162 : index
    memref.store %cst_3, %alloc_1710[%c0, %1416, %1417, %1418] : memref<1x512x7x7xf32>
    scf.reduce
  }
  scf.reduce
}
```

Figure 4: Optimized kernel generation showing improved block size mapping.

The figure above demonstrates the result of this modification. The block size is now mapped to $\{1, 32, 8\}$, providing significantly higher occupancy compared to the baseline.

4 Switch to IREE

4.1 Transition to IREE

We transitioned our experimental framework to IREE to remove issues faced when executing binaries generated directly through the standard MLIR pipeline. While IREE offered a more streamlined workflow for binary generation and runtime verification, the migration showed a fundamental mismatch between the upstream MLIR and IREE code generation pipelines:

- **Incompatibility with Upstream Mechanisms:** We initially targeted the upstream MLIR `ParallelLoopMapper` (implemented in `ParallelLoopMapper.cpp`). This pass typically annotates `scf.parallel` operations with mapping attributes, allowing the `convert-parallel-loops-to-gpu` pass to lower them to `gpu.launch`.
- **IREE Codegen Architecture:** However, we discovered that IREE's default GPU pipelines do not rely on the `scf.parallel` lowering path. Instead, IREE primarily utilizes `scf.forall` to handle workgroup and thread semantics, employing its own custom passes for GPU distribution and tiling.
- **Adaptation of Strategy:** Because IREE effectively bypassed the upstream mapper we had modified, we could not use it to benchmark our optimization. Consequently, we shifted our focus to modifying IREE's internal `LLVMGPU` kernel configuration to optimize tile sizes and thread mapping directly within their custom infrastructure.

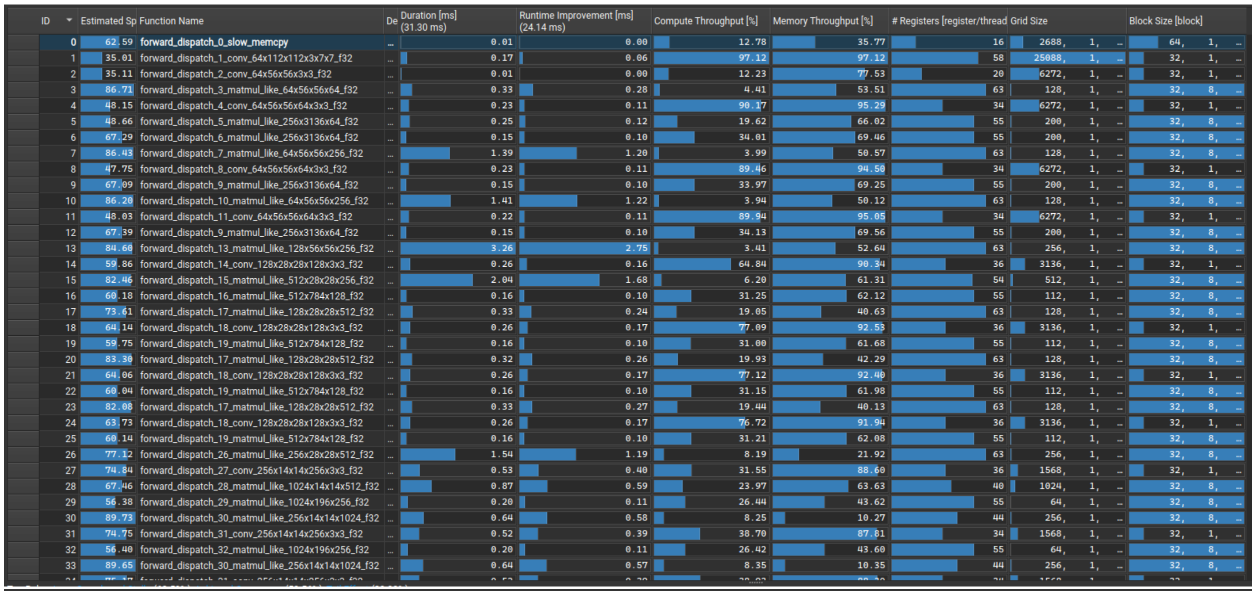
4.2 Analysis

First we profiled the vanilla IREE based resnet50 executible using ncu and obtained the following results (shown in the figure):

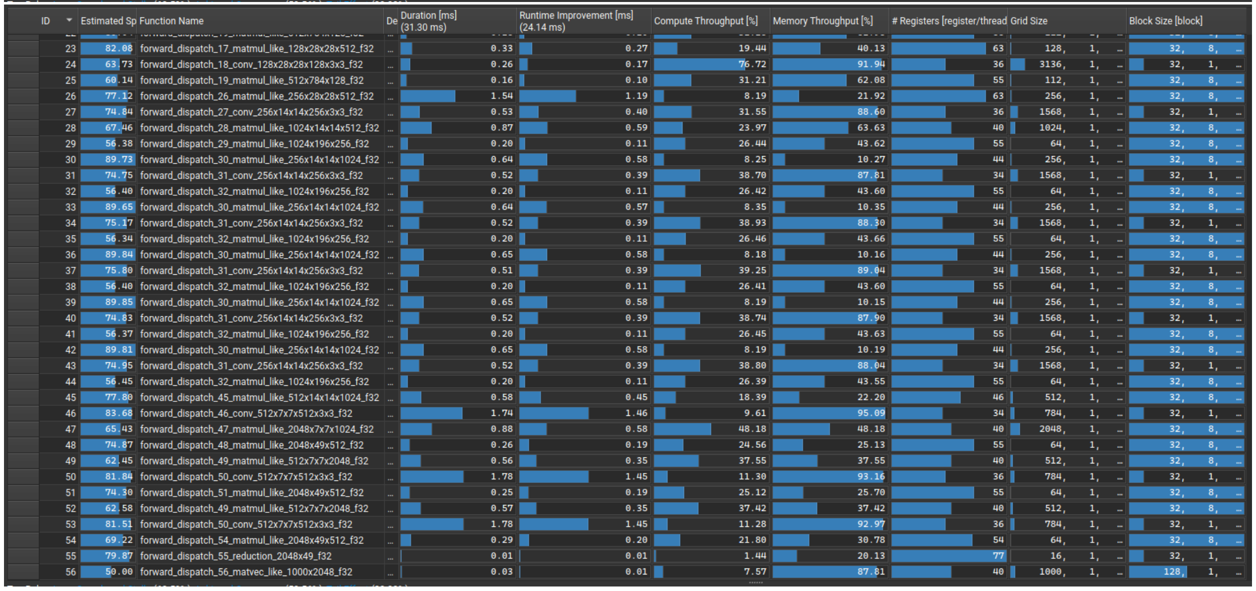
We also ran iree-benchmark-module on the module to get the actual run time without the overhead of ncu, and got the following time: 19.7ms

We could make the following inferences:

- Compute throughput was very low for most kernels.
- The slowest kernels which contributed the most to total time are the matrix multiplication kernels.



ID	Estimated Sp	Function Name	De	Duration [ms] (31.30 ms)	Runtime Improvement [ms] (24.14 ms)	Compute Throughput [%]	Memory Throughput [%]	# Registers [register/thread	Grid Size	Block Size [block]
0	62.59	forward_dispatch_0_slow_memcpy	…	0.01	0.00	12.78	35.77	16	2688, 1, …	64, 1, …
1	35.01	forward_dispatch_1_conv_64x112x112x3x7x7_f32	…	0.17	0.06	97.12	97.12	58	25088, 1, …	32, 1, …
2	35.11	forward_dispatch_2_conv_64x56x56x3x3_f32	…	0.01	0.00	12.23	77.53	20	6272, 1, …	32, 1, …
3	86.71	forward_dispatch_3_matmul_like_64x56x56x64_f32	…	0.33	0.28	4.41	53.51	63	128, 1, …	32, 8, …
4	48.15	forward_dispatch_4_conv_64x56x56x64x3x3_f32	…	0.23	0.11	90.17	95.29	34	6272, 1, …	32, 1, …
5	48.66	forward_dispatch_5_matmul_like_256x3136x64_f32	…	0.25	0.12	19.62	66.02	55	200, 1, …	32, 8, …
6	67.29	forward_dispatch_6_matmul_like_256x3136x64_f32	…	0.15	0.10	34.01	69.46	55	200, 1, …	32, 8, …
7	86.43	forward_dispatch_7_matmul_like_64x56x56x256_f32	…	1.39	1.20	3.99	50.57	63	128, 1, …	32, 8, …
8	47.75	forward_dispatch_8_conv_64x56x56x64x3x3_f32	…	0.23	0.11	89.46	94.50	34	6272, 1, …	32, 1, …
9	67.09	forward_dispatch_9_matmul_like_256x3136x64_f32	…	0.15	0.10	33.97	69.25	55	200, 1, …	32, 8, …
10	86.20	forward_dispatch_10_matmul_like_64x56x56x256_f32	…	1.41	1.22	3.94	50.12	63	128, 1, …	32, 8, …
11	48.03	forward_dispatch_11_conv_64x56x56x64x3x3_f32	…	0.22	0.11	89.94	95.05	34	6272, 1, …	32, 1, …
12	67.39	forward_dispatch_9_matmul_like_256x3136x64_f32	…	0.15	0.10	34.13	69.56	55	200, 1, …	32, 8, …
13	84.60	forward_dispatch_13_matmul_like_128x56x56x256_f32	…	3.26	2.75	3.41	52.64	63	256, 1, …	32, 8, …
14	59.86	forward_dispatch_14_conv_128x28x28x128x3x3_f32	…	0.26	0.16	64.84	90.34	36	3136, 1, …	32, 1, …
15	82.46	forward_dispatch_15_matmul_like_512x28x28x256_f32	…	2.04	1.68	6.20	61.31	54	512, 1, …	32, 8, …
16	60.18	forward_dispatch_16_matmul_like_512x784x128_f32	…	0.16	0.10	31.25	62.12	55	112, 1, …	32, 8, …
17	73.61	forward_dispatch_17_matmul_like_128x28x28x512_f32	…	0.33	0.24	19.05	40.63	63	128, 1, …	32, 8, …
18	64.14	forward_dispatch_18_conv_128x28x28x128x3x3_f32	…	0.26	0.17	77.09	92.53	36	3136, 1, …	32, 1, …
19	59.75	forward_dispatch_19_matmul_like_512x784x128_f32	…	0.16	0.10	31.00	61.68	55	112, 1, …	32, 8, …
20	83.30	forward_dispatch_17_matmul_like_128x28x28x512_f32	…	0.32	0.26	19.93	42.29	63	128, 1, …	32, 8, …
21	64.06	forward_dispatch_18_conv_128x28x28x128x3x3_f32	…	0.26	0.17	77.12	92.40	36	3136, 1, …	32, 1, …
22	60.04	forward_dispatch_19_matmul_like_512x784x128_f32	…	0.16	0.10	31.15	61.98	55	112, 1, …	32, 8, …
23	82.08	forward_dispatch_17_matmul_like_128x28x28x512_f32	…	0.33	0.27	19.44	40.13	63	128, 1, …	32, 8, …
24	63.73	forward_dispatch_18_conv_128x28x28x128x3x3_f32	…	0.26	0.17	76.72	91.94	36	3136, 1, …	32, 1, …
25	60.14	forward_dispatch_19_matmul_like_512x784x128_f32	…	0.16	0.10	31.21	62.08	55	112, 1, …	32, 8, …
26	77.12	forward_dispatch_26_matmul_like_256x28x28x512_f32	…	1.54	1.19	8.19	21.92	63	256, 1, …	32, 8, …
27	74.84	forward_dispatch_27_conv_256x14x14x256x3x3_f32	…	0.53	0.40	31.55	88.60	36	1568, 1, …	32, 1, …
28	67.46	forward_dispatch_28_matmul_like_1024x14x14x512_f32	…	0.87	0.59	23.97	63.63	40	1024, 1, …	32, 8, …
29	56.38	forward_dispatch_29_matmul_like_1024x196x256_f32	…	0.20	0.11	26.44	43.62	55	64, 1, …	32, 8, …
30	89.73	forward_dispatch_30_matmul_like_256x14x14x1024_f32	…	0.64	0.58	8.25	10.27	44	256, 1, …	32, 8, …
31	74.75	forward_dispatch_31_conv_256x14x14x256x3x3_f32	…	0.52	0.39	38.70	87.81	34	1568, 1, …	32, 1, …
32	56.40	forward_dispatch_32_matmul_like_1024x196x256_f32	…	0.20	0.11	26.42	43.60	55	64, 1, …	32, 8, …
33	89.65	forward_dispatch_30_matmul_like_256x14x14x1024_f32	…	0.64	0.57	8.35	10.35	44	256, 1, …	32, 8, …



ID	Estimated Sp	Function Name	De	Duration [ms] (31.30 ms)	Runtime Improvement [ms] (24.14 ms)	Compute Throughput [%]	Memory Throughput [%]	# Registers [register/thread	Grid Size	Block Size [block]
23	82.08	forward_dispatch_17_matmul_like_128x28x28x512_f32	…	0.33	0.27	19.44	40.13	63	128, 1, …	32, 8, …
24	63.73	forward_dispatch_18_conv_128x28x28x128x3x3_f32	…	0.26	0.17	76.72	91.94	36	3136, 1, …	32, 1, …
25	60.14	forward_dispatch_19_matmul_like_512x784x128_f32	…	0.16	0.10	31.21	62.08	55	112, 1, …	32, 8, …
26	77.12	forward_dispatch_26_matmul_like_256x28x28x512_f32	…	1.54	1.19	8.19	21.92	63	256, 1, …	32, 8, …
27	74.84	forward_dispatch_27_conv_256x14x14x256x3x3_f32	…	0.53	0.40	31.55	88.60	36	1568, 1, …	32, 1, …
28	67.46	forward_dispatch_28_matmul_like_1024x14x14x512_f32	…	0.87	0.59	23.97	63.63	40	1024, 1, …	32, 8, …
29	56.38	forward_dispatch_29_matmul_like_1024x196x256_f32	…	0.20	0.11	26.44	43.62	55	64, 1, …	32, 8, …
30	89.73	forward_dispatch_30_matmul_like_256x14x14x1024_f32	…	0.64	0.58	8.25	10.27	44	256, 1, …	32, 8, …
31	74.75	forward_dispatch_31_conv_256x14x14x256x3x3_f32	…	0.52	0.39	38.70	87.81	34	1568, 1, …	32, 1, …
32	56.40	forward_dispatch_32_matmul_like_1024x196x256_f32	…	0.20	0.11	26.42	43.60	55	64, 1, …	32, 8, …
33	89.65	forward_dispatch_30_matmul_like_256x14x14x1024_f32	…	0.64	0.57	8.35	10.35	44	256, 1, …	32, 8, …
34	75.17	forward_dispatch_31_conv_256x14x14x256x3x3_f32	…	0.52	0.39	38.93	88.30	34	1568, 1, …	32, 1, …
35	56.34	forward_dispatch_32_matmul_like_1024x196x256_f32	…	0.20	0.11	26.46	43.66	55	64, 1, …	32, 8, …
36	89.84	forward_dispatch_30_matmul_like_256x14x14x1024_f32	…	0.65	0.58	8.18	10.16	44	256, 1, …	32, 8, …
37	75.80	forward_dispatch_31_conv_256x14x14x256x3x3_f32	…	0.51	0.39	39.25	89.04	34	1568, 1, …	32, 1, …
38	56.40	forward_dispatch_32_matmul_like_1024x196x256_f32	…	0.20	0.11	26.41	43.60	55	64, 1, …	32, 8, …
39	89.85	forward_dispatch_30_matmul_like_256x14x14x1024_f32	…	0.65	0.58	8.19	10.15	44	256, 1, …	32, 8, …
40	74.83	forward_dispatch_31_conv_256x14x14x256x3x3_f32	…	0.52	0.39	38.74	87.90	34	1568, 1, …	32, 1, …
41	56.37	forward_dispatch_32_matmul_like_1024x196x256_f32	…	0.20	0.11	26.45	43.63	55	64, 1, …	32, 8, …
42	89.81	forward_dispatch_30_matmul_like_256x14x14x1024_f32	…	0.65	0.58	8.19	10.19	44	256, 1, …	32, 8, …
43	74.95	forward_dispatch_31_conv_256x14x14x256x3x3_f32	…	0.52	0.39	38.80	88.04	34	1568, 1, …	32, 1, …
44	56.45	forward_dispatch_32_matmul_like_1024x196x256_f32	…	0.20	0.11	26.39	43.55	55	64, 1, …	32, 8, …
45	77.80	forward_dispatch_45_matmul_like_512x14x14x1024_f32	…	0.58	0.45	18.39	22.20	46	512, 1, …	32, 8, …
46	83.68	forward_dispatch_46_conv_512x7x7x512x3x3_f32	…	1.74	1.46	9.61	95.09	34	784, 1, …	32, 1, …
47	65.43	forward_dispatch_47_matmul_like_2048x7x7x1024_f32	…	0.88	0.58	48.18	48.18	40	2048, 1, …	32, 8, …
48	74.87	forward_dispatch_48_matmul_like_2048x49x512_f32	…	0.26	0.19	24.56	25.13	55	64, 1, …	32, 8, …
49	62.45	forward_dispatch_49_matmul_like_512x7x7x2048_f32	…	0.56	0.35	37.55	37.55	40	512, 1, …	32, 8, …
50	81.84	forward_dispatch_50_conv_512x7x7x512x3x3_f32	…	1.78	1.45	11.30	93.16	36	784, 1, …	32, 1, …
51	74.30	forward_dispatch_51_matmul_like_2048x49x512_f32	…	0.25	0.19	25.12	25.70	55	64, 1, …	32, 8, …
52	62.58	forward_dispatch_49_matmul_like_512x7x7x2048_f32	…	0.57	0.35	37.42	37.42	40	512, 1, …	32, 8, …
53	81.51	forward_dispatch_50_conv_512x7x7x512x3x3_f32	…	1.78	1.45	11.28	92.97	36	784, 1, …	32, 1, …
54	69.22	forward_dispatch_54_matmul_like_2048x49x512_f32	…	0.29	0.20	21.80	30.78	54	64, 1, …	32, 8, …
55	79.87	forward_dispatch_55_reduction_2048x49_f32	…	0.01	0.01	1.44	20.13	77	16, 1, …	32, 1, …
56	50.00	forward_dispatch_56_matvec_like_1000x2048_f32	…	0.03	0.01	7.57	87.81	40	1000, 1, …	128, 1, …

Figure 5: Default iree performance with O3

- The thread configuration in most of the blocks is (32, 8, 1) and some of them (mostly convolution kernels) are (32, 1, 1)
- Most kernels have a very high percentage of uncoalesced accesses.

4.3 Trying to improve throughput of matrix multiplication kernels

First, we focused on trying to improve the throughput of matrix multiplication kernels as they were the bottleneck. We looked through the code in the ”KernelConfig.cpp” file, and observed the following:

- There is a fixed set of tile sizes and thread block mapping in the following function:

```
static SmallVector<TileWorkgroupSizePair>
getMatmulConfig(IREE::GPU::TargetAttr target) {
  SmallVector<TileWorkgroupSizePair> tileSizes;
  // Pick tile size so that M*K and K*N divisible by wgSize * \*vecSize=*\4.
  // This way workgroup memory copy don’t need to be masked. Once we support
  // masked load we can get performance out of more configuration.

  // Make use of the full subgroup when possible.
  if (target.getPreferredSubgroupSize() == 64) {
    tileSizes.push_back(TileWorkgroupSizePair({{64, 128, 64}, {64, 16, 1}, 1}));
  }

  llvm::append_values(tileSizes,
                      TileWorkgroupSizePair({{32, 128, 32}, {32, 8, 1}, 1}),
                      TileWorkgroupSizePair({{128, 64, 8}, {16, 8, 1}, 1}),
                      TileWorkgroupSizePair({{16, 256, 32}, {64, 2, 1}, 1}),
                      TileWorkgroupSizePair({{8, 32, 32}, {8, 8, 1}, 1}),

                      TileWorkgroupSizePair({{32, 128, 4}, {32, 8, 1}, 1}),
                      TileWorkgroupSizePair({{8, 128, 4}, {32, 1, 1}, 1}),
                      TileWorkgroupSizePair({{16, 64, 4}, {16, 2, 1}, 1}),
                      TileWorkgroupSizePair({{1, 128, 8}, {32, 1, 1}, 1}));
  return tileSizes;
}
```

  Here, the first three values in TileWorkgroupSizePair correspond to the M, N and K dimensions of the tiles, while the next three correspond to the x, y and z block dimensions.

- Also, in the function which assigns the tiling dimensions to each kernel (setContractConfig), we noticed that if the boolean ”isStaticSize” was true, which was not the case for all our kernels, it would simply use the first element of the tileSizes variable to assign tile size. This would be:
  `TileWorkgroupSizePair({{32, 128, 32}, {32, 8, 1}, 1})`
  which is not really optimal for resnet50.
- Our intuition was that a square tile size with width 16 will probably perform the best on our given GPU.
- So, we added this tile size to the top of the list:
  `TileWorkgroupSizePair({{16, 16, 16}, {16, 16, 1}, 1})`
- This gives a perfectly square 16x16 tile and also assigns a 16x16 thread block to each tile.

4.3.1 Results

With this simple optimization, The runtime of the module that we got with iree-benchmark-module went down to 10.3ms. The ncu report of that run is shown in the figures below:

4.3.2 Observations

- We observed that all the matrix multiplication kernels were quite fast now, with much higher memory and compute throughputs overall.
- However, now the bottleneck was the convolution kernels.
- So, next we focused on finding out why the convolution kernels were so slow, with very low compute throughput and a lot of uncoalesced accesses.

4.4 Improving the convolution kernels

With a lot of debugging and digging through the code, we found that the tile sizes for the convolutions were being mapped by the "setTileAndFuseLoweringConfig" function in ConfigUtils.cpp. The part of this function that actually does this mapping is the "distributeToThreads" lambda in this function. Here, we can take an example of a particular convolution kernel to explain what this does:

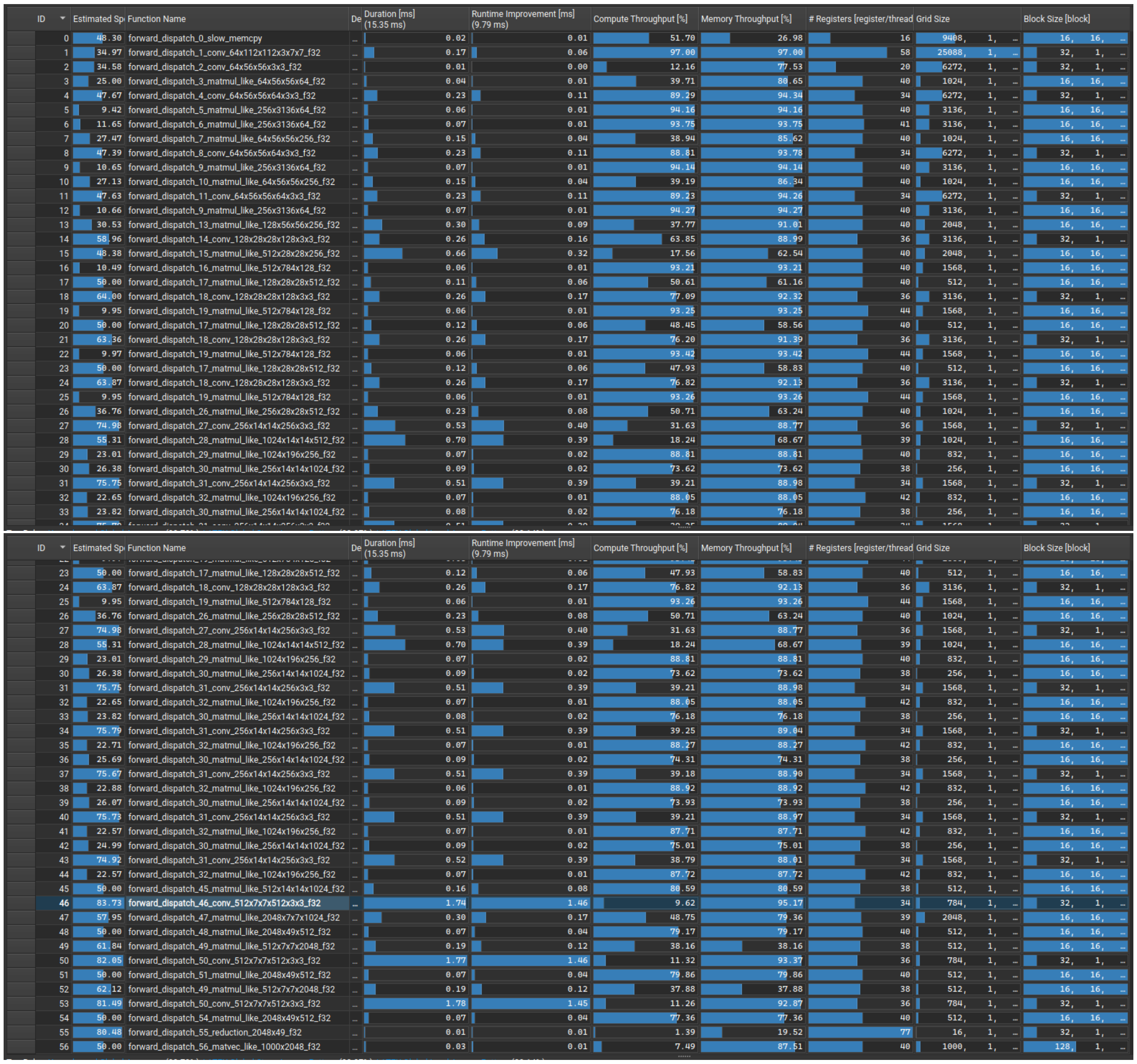
- We'll look at the `forward_dispatch_46_conv_512x7x7x512x3x3_f32` kernel, which outputs a tensor with dimension 512x7x7, with 512x3x3 kernels for each input channel.
- First, this lambda finds the "partitionable" loops in this kernel, which in this case are the ones with dimensions 512, 7 and 7.
- Then, it goes from the innermost dimension to the outermost dimension and tries to assign the highest number of threads (powers of 2) to each dimension.
- However, there is also a "lossFactor" parameter in this lambda, and it will only assign threads to a dimension if the fraction of idle threads are lesser than or equal to $\frac{1}{\texttt{lossFactor}}$
- However, if we look at where this lambda is called, we see that it first tries to assign threads with a 0 loss factor.
- This will actually result in threads being tiled only across the 512 dimension, since the dimension of 7 will always result in some threads being idle.
- Instead, we thought that maybe distributing threads across the 7x7 dimensions, it would give better throughput.
- So, we changed the first call to this lambda to take a lossFactor of 4, so that it allows $\frac{1}{4}$ of the threads to be idle. The results for that are shown below:

4.4.1 Results

From iree-benchmark-module, we got a runtime of 6.84ms.

4.4.2 Observations

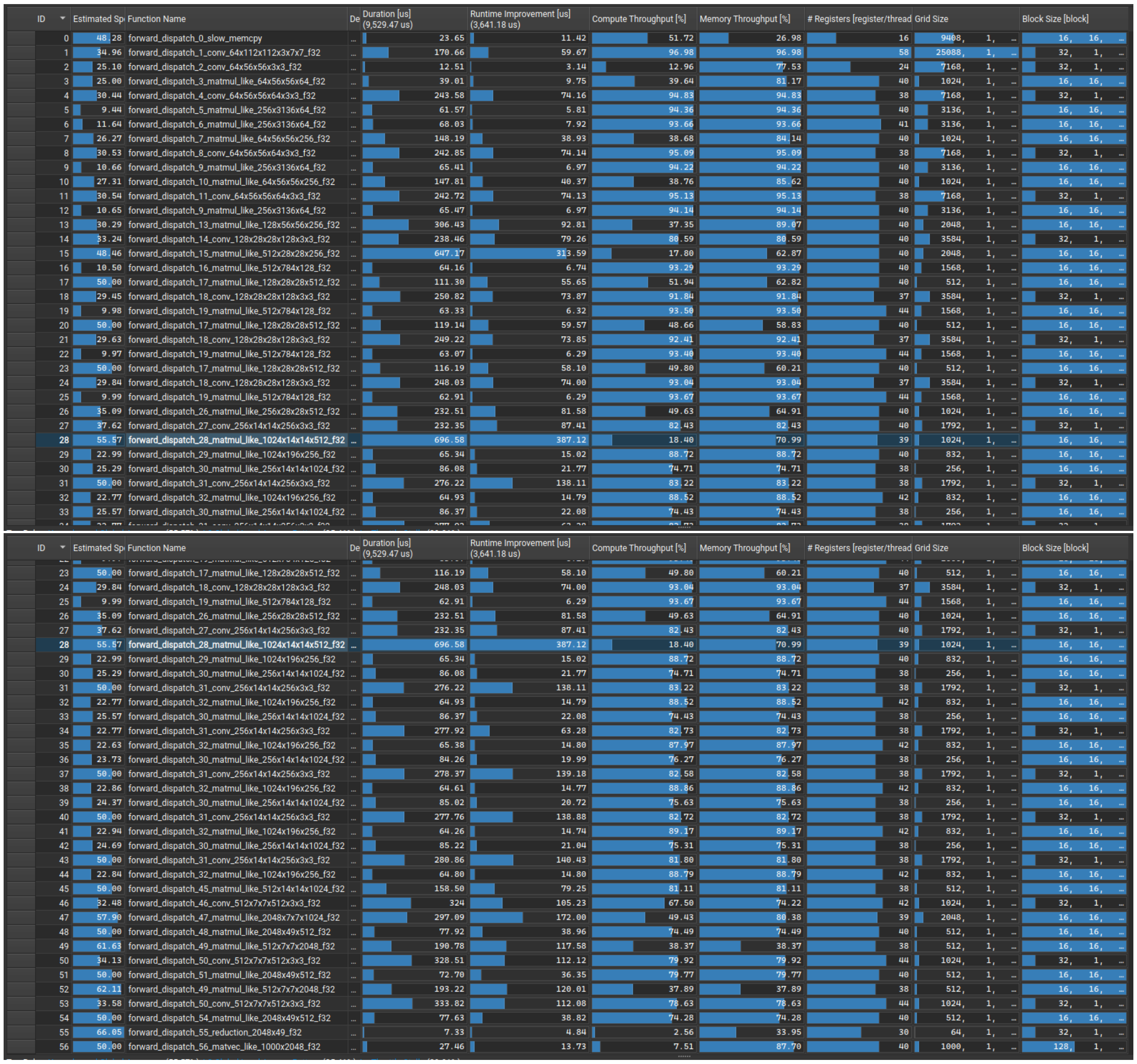
- Here, we observe that the compute throughput and coalescing has significantly improved for the convolution kernels.
- We tried to increase the number of threads per block for the convolution kernels, but they were giving worse results.



ID	Estimated Sp	Function Name	De	Duration [ms] (15.35 ms)	Runtime Improvement [ms] (9.79 ms)	Compute Throughput [%]	Memory Throughput [%]	# Registers [register/thread	Grid Size	Block Size [block]
0	48.30	forward_dispatch_0_slow_memcpy	...	0.02	0.01	51.70	26.98	16	9408, 1, ...	16, 16, ...
1	34.97	forward_dispatch_1_conv_64x112x112x3x7x7_f32	...	0.17	0.06	97.00	97.00	58	25088, 1, ...	32, 1, ...
2	34.58	forward_dispatch_2_conv_64x56x56x3x3_f32	...	0.01	0.00	12.16	77.53	20	6272, 1, ...	32, 1, ...
3	25.00	forward_dispatch_3_matmul_like_64x56x56x64_f32	...	0.04	0.01	39.71	80.65	40	1024, 1, ...	16, 16, ...
4	47.67	forward_dispatch_4_conv_64x56x56x64x3x3_f32	...	0.23	0.11	89.29	94.34	34	6272, 1, ...	32, 1, ...
5	9.42	forward_dispatch_5_matmul_like_256x3136x64_f32	...	0.06	0.01	94.16	94.16	40	3136, 1, ...	16, 16, ...
6	11.65	forward_dispatch_6_matmul_like_256x3136x64_f32	...	0.07	0.01	93.75	93.75	41	3136, 1, ...	16, 16, ...
7	27.47	forward_dispatch_7_matmul_like_64x56x56x256_f32	...	0.15	0.04	38.94	85.62	40	1024, 1, ...	16, 16, ...
8	47.39	forward_dispatch_8_conv_64x56x56x64x3x3_f32	...	0.23	0.11	88.81	93.78	34	6272, 1, ...	32, 1, ...
9	10.65	forward_dispatch_9_matmul_like_256x3136x64_f32	...	0.07	0.01	94.14	94.14	40	3136, 1, ...	16, 16, ...
10	27.13	forward_dispatch_10_matmul_like_64x56x56x256_f32	...	0.15	0.04	39.19	86.34	40	1024, 1, ...	16, 16, ...
11	47.63	forward_dispatch_11_conv_64x56x56x64x3x3_f32	...	0.23	0.11	89.23	94.26	34	6272, 1, ...	32, 1, ...
12	10.66	forward_dispatch_9_matmul_like_256x3136x64_f32	...	0.07	0.01	94.27	94.27	40	3136, 1, ...	16, 16, ...
13	30.53	forward_dispatch_13_matmul_like_128x56x56x256_f32	...	0.30	0.09	37.77	91.01	40	2048, 1, ...	16, 16, ...
14	58.96	forward_dispatch_14_conv_128x28x28x128x3x3_f32	...	0.26	0.16	63.85	88.99	36	3136, 1, ...	32, 1, ...
15	48.38	forward_dispatch_15_matmul_like_512x28x28x256_f32	...	0.66	0.32	17.56	62.54	40	2048, 1, ...	16, 16, ...
16	10.49	forward_dispatch_16_matmul_like_512x784x128_f32	...	0.06	0.01	93.21	93.21	40	1568, 1, ...	16, 16, ...
17	50.00	forward_dispatch_17_matmul_like_128x28x28x512_f32	...	0.11	0.06	50.61	61.16	40	512, 1, ...	16, 16, ...
18	64.00	forward_dispatch_18_conv_128x28x28x128x3x3_f32	...	0.26	0.17	77.09	92.32	36	3136, 1, ...	32, 1, ...
19	9.95	forward_dispatch_19_matmul_like_512x784x128_f32	...	0.06	0.01	93.25	93.25	44	1568, 1, ...	16, 16, ...
20	50.00	forward_dispatch_17_matmul_like_128x28x28x512_f32	...	0.12	0.06	48.45	58.56	40	512, 1, ...	16, 16, ...
21	63.36	forward_dispatch_18_conv_128x28x28x128x3x3_f32	...	0.26	0.17	76.20	91.39	36	3136, 1, ...	32, 1, ...
22	9.97	forward_dispatch_19_matmul_like_512x784x128_f32	...	0.06	0.01	93.42	93.42	44	1568, 1, ...	16, 16, ...
23	50.00	forward_dispatch_17_matmul_like_128x28x28x512_f32	...	0.12	0.06	47.93	58.83	40	512, 1, ...	16, 16, ...
24	63.87	forward_dispatch_18_conv_128x28x28x128x3x3_f32	...	0.26	0.17	76.82	92.13	36	3136, 1, ...	32, 1, ...
25	9.95	forward_dispatch_19_matmul_like_512x784x128_f32	...	0.06	0.01	93.26	93.26	44	1568, 1, ...	16, 16, ...
26	36.76	forward_dispatch_26_matmul_like_256x28x28x512_f32	...	0.23	0.08	50.71	63.24	40	1024, 1, ...	16, 16, ...
27	74.98	forward_dispatch_27_conv_256x14x14x256x3x3_f32	...	0.53	0.40	31.63	88.77	36	1568, 1, ...	32, 1, ...
28	55.31	forward_dispatch_28_matmul_like_1024x14x14x512_f32	...	0.70	0.39	18.24	68.67	39	1024, 1, ...	16, 16, ...
29	23.01	forward_dispatch_29_matmul_like_1024x196x256_f32	...	0.07	0.02	88.81	88.81	40	832, 1, ...	16, 16, ...
30	26.38	forward_dispatch_30_matmul_like_256x14x14x1024_f32	...	0.09	0.02	73.62	73.62	38	256, 1, ...	16, 16, ...
31	75.75	forward_dispatch_31_conv_256x14x14x256x3x3_f32	...	0.51	0.39	39.21	88.98	34	1568, 1, ...	32, 1, ...
32	22.65	forward_dispatch_32_matmul_like_1024x196x256_f32	...	0.07	0.01	88.05	88.05	42	832, 1, ...	16, 16, ...
33	23.82	forward_dispatch_30_matmul_like_256x14x14x1024_f32	...	0.08	0.02	76.18	76.18	38	256, 1, ...	16, 16, ...

ID	Estimated Sp	Function Name	De	Duration [ms] (15.35 ms)	Runtime Improvement [ms] (9.79 ms)	Compute Throughput [%]	Memory Throughput [%]	# Registers [register/thread	Grid Size	Block Size [block]
23	50.00	forward_dispatch_17_matmul_like_128x28x28x512_f32	...	0.12	0.06	47.93	58.83	40	512, 1, ...	16, 16, ...
24	63.87	forward_dispatch_18_conv_128x28x28x128x3x3_f32	...	0.26	0.17	76.82	92.13	36	3136, 1, ...	32, 1, ...
25	9.95	forward_dispatch_19_matmul_like_512x784x128_f32	...	0.06	0.01	93.26	93.26	44	1568, 1, ...	16, 16, ...
26	36.76	forward_dispatch_26_matmul_like_256x28x28x512_f32	...	0.23	0.08	50.71	63.24	40	1024, 1, ...	16, 16, ...
27	74.98	forward_dispatch_27_conv_256x14x14x256x3x3_f32	...	0.53	0.40	31.63	88.77	36	1568, 1, ...	32, 1, ...
28	55.31	forward_dispatch_28_matmul_like_1024x14x14x512_f32	...	0.70	0.39	18.24	68.67	39	1024, 1, ...	16, 16, ...
29	23.01	forward_dispatch_29_matmul_like_1024x196x256_f32	...	0.07	0.02	88.81	88.81	40	832, 1, ...	16, 16, ...
30	26.38	forward_dispatch_30_matmul_like_256x14x14x1024_f32	...	0.09	0.02	73.62	73.62	38	256, 1, ...	16, 16, ...
31	75.75	forward_dispatch_31_conv_256x14x14x256x3x3_f32	...	0.51	0.39	39.21	88.98	34	1568, 1, ...	32, 1, ...
32	22.65	forward_dispatch_32_matmul_like_1024x196x256_f32	...	0.07	0.01	88.05	88.05	42	832, 1, ...	16, 16, ...
33	23.82	forward_dispatch_30_matmul_like_256x14x14x1024_f32	...	0.08	0.02	76.18	76.18	38	256, 1, ...	16, 16, ...
34	75.79	forward_dispatch_31_conv_256x14x14x256x3x3_f32	...	0.51	0.39	39.25	89.04	34	1568, 1, ...	32, 1, ...
35	22.71	forward_dispatch_32_matmul_like_1024x196x256_f32	...	0.07	0.01	88.27	88.27	42	832, 1, ...	16, 16, ...
36	25.69	forward_dispatch_30_matmul_like_256x14x14x1024_f32	...	0.09	0.02	74.31	74.31	38	256, 1, ...	16, 16, ...
37	75.67	forward_dispatch_31_conv_256x14x14x256x3x3_f32	...	0.51	0.39	39.18	88.90	34	1568, 1, ...	32, 1, ...
38	22.88	forward_dispatch_32_matmul_like_1024x196x256_f32	...	0.06	0.01	88.92	88.92	42	832, 1, ...	16, 16, ...
39	26.07	forward_dispatch_30_matmul_like_256x14x14x1024_f32	...	0.09	0.02	73.93	73.93	38	256, 1, ...	16, 16, ...
40	75.73	forward_dispatch_31_conv_256x14x14x256x3x3_f32	...	0.51	0.39	39.21	88.97	34	1568, 1, ...	32, 1, ...
41	22.57	forward_dispatch_32_matmul_like_1024x196x256_f32	...	0.07	0.01	87.71	87.71	42	832, 1, ...	16, 16, ...
42	24.99	forward_dispatch_30_matmul_like_256x14x14x1024_f32	...	0.09	0.02	75.01	75.01	38	256, 1, ...	16, 16, ...
43	74.92	forward_dispatch_31_conv_256x14x14x256x3x3_f32	...	0.52	0.39	38.79	88.01	34	1568, 1, ...	32, 1, ...
44	22.57	forward_dispatch_32_matmul_like_1024x196x256_f32	...	0.07	0.01	87.72	87.72	42	832, 1, ...	16, 16, ...
45	50.00	forward_dispatch_45_matmul_like_512x14x14x1024_f32	...	0.16	0.08	80.59	80.59	38	512, 1, ...	16, 16, ...
46	83.73	forward_dispatch_46_conv_512x7x7x512x3x3_f32	...	1.74	1.46	9.62	95.17	34	784, 1, ...	32, 1, ...
47	57.95	forward_dispatch_47_matmul_like_2048x7x7x1024_f32	...	0.30	0.17	48.75	79.36	39	2048, 1, ...	16, 16, ...
48	50.00	forward_dispatch_48_matmul_like_2048x49x512_f32	...	0.07	0.04	79.17	79.17	40	512, 1, ...	16, 16, ...
49	61.84	forward_dispatch_49_matmul_like_512x7x7x2048_f32	...	0.19	0.12	38.16	38.16	38	512, 1, ...	16, 16, ...
50	82.05	forward_dispatch_50_conv_512x7x7x512x3x3_f32	...	1.77	1.46	11.32	93.37	36	784, 1, ...	32, 1, ...
51	50.00	forward_dispatch_51_matmul_like_2048x49x512_f32	...	0.07	0.04	79.86	79.86	40	512, 1, ...	16, 16, ...
52	62.12	forward_dispatch_49_matmul_like_512x7x7x2048_f32	...	0.19	0.12	37.88	37.88	38	512, 1, ...	16, 16, ...
53	81.49	forward_dispatch_50_conv_512x7x7x512x3x3_f32	...	1.78	1.45	11.26	92.87	36	784, 1, ...	32, 1, ...
54	50.00	forward_dispatch_54_matmul_like_2048x49x512_f32	...	0.07	0.04	77.36	77.36	40	512, 1, ...	16, 16, ...
55	80.48	forward_dispatch_55_reduction_2048x49_f32	...	0.01	0.01	1.39	19.52	77	16, 1, ...	32, 1, ...
56	50.00	forward_dispatch_56_matvec_like_1000x2048_f32	...	0.03	0.01	7.49	87.51	40	1000, 1, ...	128, 1, ...

Figure 6: Report for run with mat-mult tilesize optimization



ID	Estimated Sp	Function Name	De	Duration [us] (9,529.47 us)	Runtime Improvement [us] (3,641.18 us)	Compute Throughput [%]	Memory Throughput [%]	# Registers [register/thread	Grid Size	Block Size [block]
0	48.28	forward_dispatch_0_slow_memcpy	...	23.65	11.42	51.72	26.98	16	9408, 1, ...	16, 16, ...
1	34.96	forward_dispatch_1_conv_64x112x112x3x7x7_f32	...	170.66	59.67	96.98	96.98	58	25088, 1, ...	32, 1, ...
2	25.10	forward_dispatch_2_conv_64x56x56x3x3_f32	...	12.51	3.14	12.96	77.53	24	7168, 1, ...	32, 1, ...
3	25.00	forward_dispatch_3_matmul_like_64x56x56x64_f32	...	39.01	9.75	39.64	81.17	40	1024, 1, ...	16, 16, ...
4	30.44	forward_dispatch_4_conv_64x56x56x64x3x3_f32	...	243.58	74.16	94.83	94.83	38	7168, 1, ...	32, 1, ...
5	9.44	forward_dispatch_5_matmul_like_256x3136x64_f32	...	61.57	5.81	94.36	94.36	40	3136, 1, ...	16, 16, ...
6	11.64	forward_dispatch_6_matmul_like_256x3136x64_f32	...	68.03	7.92	93.66	93.66	41	3136, 1, ...	16, 16, ...
7	26.27	forward_dispatch_7_matmul_like_64x56x56x256_f32	...	148.19	38.93	38.68	84.14	40	1024, 1, ...	16, 16, ...
8	30.53	forward_dispatch_8_conv_64x56x56x64x3x3_f32	...	242.85	74.14	95.09	95.09	38	7168, 1, ...	32, 1, ...
9	10.66	forward_dispatch_9_matmul_like_256x3136x64_f32	...	65.41	6.97	94.22	94.22	40	3136, 1, ...	16, 16, ...
10	27.31	forward_dispatch_10_matmul_like_64x56x56x256_f32	...	147.81	40.37	38.76	85.62	40	1024, 1, ...	16, 16, ...
11	30.54	forward_dispatch_11_conv_64x56x56x64x3x3_f32	...	242.72	74.13	95.13	95.13	38	7168, 1, ...	32, 1, ...
12	10.65	forward_dispatch_9_matmul_like_256x3136x64_f32	...	65.47	6.97	94.14	94.14	40	3136, 1, ...	16, 16, ...
13	30.29	forward_dispatch_13_matmul_like_128x56x56x256_f32	...	306.43	92.81	37.35	89.07	40	2048, 1, ...	16, 16, ...
14	33.24	forward_dispatch_14_conv_128x28x28x128x3x3_f32	...	238.46	79.26	80.59	80.59	40	3584, 1, ...	32, 1, ...
15	48.46	forward_dispatch_15_matmul_like_512x28x28x256_f32	...	647.17	313.59	17.80	62.87	40	2048, 1, ...	16, 16, ...
16	10.50	forward_dispatch_16_matmul_like_512x784x128_f32	...	64.16	6.74	93.29	93.29	40	1568, 1, ...	16, 16, ...
17	50.00	forward_dispatch_17_matmul_like_128x28x28x512_f32	...	111.30	55.65	51.94	62.82	40	512, 1, ...	16, 16, ...
18	29.45	forward_dispatch_18_conv_128x28x28x128x3x3_f32	...	250.82	73.87	91.84	91.84	37	3584, 1, ...	32, 1, ...
19	9.98	forward_dispatch_19_matmul_like_512x784x128_f32	...	63.33	6.32	93.50	93.50	44	1568, 1, ...	16, 16, ...
20	50.00	forward_dispatch_17_matmul_like_128x28x28x512_f32	...	119.14	59.57	48.66	58.83	40	512, 1, ...	16, 16, ...
21	29.63	forward_dispatch_18_conv_128x28x28x128x3x3_f32	...	249.22	73.85	92.41	92.41	37	3584, 1, ...	32, 1, ...
22	9.97	forward_dispatch_19_matmul_like_512x784x128_f32	...	63.07	6.29	93.40	93.40	44	1568, 1, ...	16, 16, ...
23	50.00	forward_dispatch_17_matmul_like_128x28x28x512_f32	...	116.19	58.10	49.80	60.21	40	512, 1, ...	16, 16, ...
24	29.84	forward_dispatch_18_conv_128x28x28x128x3x3_f32	...	248.03	74.00	93.04	93.04	37	3584, 1, ...	32, 1, ...
25	9.99	forward_dispatch_19_matmul_like_512x784x128_f32	...	62.91	6.29	93.67	93.67	44	1568, 1, ...	16, 16, ...
26	35.09	forward_dispatch_26_matmul_like_256x28x28x512_f32	...	232.51	81.58	49.63	64.91	40	1024, 1, ...	16, 16, ...
27	37.62	forward_dispatch_27_conv_256x14x14x256x3x3_f32	...	232.35	87.41	82.43	82.43	40	1792, 1, ...	32, 1, ...
28	55.57	forward_dispatch_28_matmul_like_1024x14x14x512_f32	...	696.58	387.12	18.40	70.99	39	1024, 1, ...	16, 16, ...
29	22.99	forward_dispatch_29_matmul_like_1024x196x256_f32	...	65.34	15.02	88.72	88.72	40	832, 1, ...	16, 16, ...
30	25.29	forward_dispatch_30_matmul_like_256x14x14x1024_f32	...	86.08	21.77	74.71	74.71	38	256, 1, ...	16, 16, ...
31	50.00	forward_dispatch_31_conv_256x14x14x256x3x3_f32	...	276.22	138.11	83.22	83.22	38	1792, 1, ...	32, 1, ...
32	22.77	forward_dispatch_32_matmul_like_1024x196x256_f32	...	64.93	14.79	88.52	88.52	42	832, 1, ...	16, 16, ...
33	25.57	forward_dispatch_30_matmul_like_256x14x14x1024_f32	...	86.37	22.08	74.43	74.43	38	256, 1, ...	16, 16, ...
34	22.77	forward_dispatch_31_conv_256x14x14x256x3x3_f32	...	277.92	63.28	82.73	82.73	38	1792, 1, ...	32, 1, ...
35	22.63	forward_dispatch_32_matmul_like_1024x196x256_f32	...	65.38	14.80	87.97	87.97	42	832, 1, ...	16, 16, ...
36	23.73	forward_dispatch_30_matmul_like_256x14x14x1024_f32	...	84.26	19.99	76.27	76.27	38	256, 1, ...	16, 16, ...
37	50.00	forward_dispatch_31_conv_256x14x14x256x3x3_f32	...	278.37	139.18	82.58	82.58	38	1792, 1, ...	32, 1, ...
38	22.86	forward_dispatch_32_matmul_like_1024x196x256_f32	...	64.61	14.77	88.86	88.86	42	832, 1, ...	16, 16, ...
39	24.37	forward_dispatch_30_matmul_like_256x14x14x1024_f32	...	85.02	20.72	75.63	75.63	38	256, 1, ...	16, 16, ...
40	50.00	forward_dispatch_31_conv_256x14x14x256x3x3_f32	...	277.76	138.88	82.72	82.72	38	1792, 1, ...	32, 1, ...
41	22.94	forward_dispatch_32_matmul_like_1024x196x256_f32	...	64.26	14.74	89.17	89.17	42	832, 1, ...	16, 16, ...
42	24.69	forward_dispatch_30_matmul_like_256x14x14x1024_f32	...	85.22	21.04	75.31	75.31	38	256, 1, ...	16, 16, ...
43	50.00	forward_dispatch_31_conv_256x14x14x256x3x3_f32	...	280.86	140.43	81.80	81.80	38	1792, 1, ...	32, 1, ...
44	22.84	forward_dispatch_32_matmul_like_1024x196x256_f32	...	64.80	14.80	88.79	88.79	42	832, 1, ...	16, 16, ...
45	50.00	forward_dispatch_45_matmul_like_512x14x14x1024_f32	...	158.50	79.25	81.11	81.11	38	512, 1, ...	16, 16, ...
46	32.48	forward_dispatch_46_conv_512x7x7x512x3x3_f32	...	324	105.23	67.50	74.22	42	1024, 1, ...	32, 1, ...
47	57.90	forward_dispatch_47_matmul_like_2048x7x7x1024_f32	...	297.09	172.00	49.43	80.38	39	2048, 1, ...	16, 16, ...
48	50.00	forward_dispatch_48_matmul_like_2048x49x512_f32	...	77.92	38.96	74.49	74.49	40	512, 1, ...	16, 16, ...
49	61.63	forward_dispatch_49_matmul_like_512x7x7x2048_f32	...	190.78	117.58	38.37	38.37	38	512, 1, ...	16, 16, ...
50	34.13	forward_dispatch_50_conv_512x7x7x512x3x3_f32	...	328.51	112.12	79.92	79.92	44	1024, 1, ...	32, 1, ...
51	50.00	forward_dispatch_51_matmul_like_2048x49x512_f32	...	72.70	36.35	79.77	79.77	40	512, 1, ...	16, 16, ...
52	62.11	forward_dispatch_49_matmul_like_512x7x7x2048_f32	...	193.22	120.01	37.89	37.89	38	512, 1, ...	16, 16, ...
53	33.58	forward_dispatch_50_conv_512x7x7x512x3x3_f32	...	333.82	112.08	78.63	78.63	44	1024, 1, ...	32, 1, ...
54	50.00	forward_dispatch_54_matmul_like_2048x49x512_f32	...	77.63	38.82	74.28	74.28	40	512, 1, ...	16, 16, ...
55	66.05	forward_dispatch_55_reduction_2048x49_f32	...	7.33	4.84	2.56	33.95	30	64, 1, ...	32, 1, ...
56	50.00	forward_dispatch_56_matvec_like_1000x2048_f32	...	27.46	13.73	7.51	87.70	40	1000, 1, ...	128, 1, ...

Figure 7: Report for run with both mat-mult and conv tilesize optimizations

Configuration	**Runtime (ms)**	Speedup
Original IREE compile	19.7	-
With 16×16 matmul	10.3	1.91x
With optimized convolutions	**6.84**	**2.88x**

Table 1: Final results

5 Final results

Overall, we see a 2.88x improvement over the default iree compiler with our improvements. The kernels went from mostly having low compute and memory throughput to now most of them having close to maximum compute and memory throughput. We also see that the number of uncoalesced accesses decreased significantly.

6 Challenges Faced

- **MLIR Documentation Issues:** Navigating through the MLIR passes was difficult because the documentation was often incomplete or unclear.
- **IREE Complexity:** Finding the right place to make modifications in IREE was even harder, as there was almost no documentation for its internal passes.
- **Execution Delays:** We faced significant issues when trying to run the generated executables. It took approximately two weeks of debugging to resolve these errors and successfully run the code.

7 Future Work

- Develop an interface that allows users to manually configure the tile sizes and thread block mappings.
- Extend the current implementation to support a wider variety of kernel sizes and input shapes.