An End-to-End Deep Learning Compiler for Occamy

Targeting a RISC-V-Based Accelerator Using IREE & xDSL

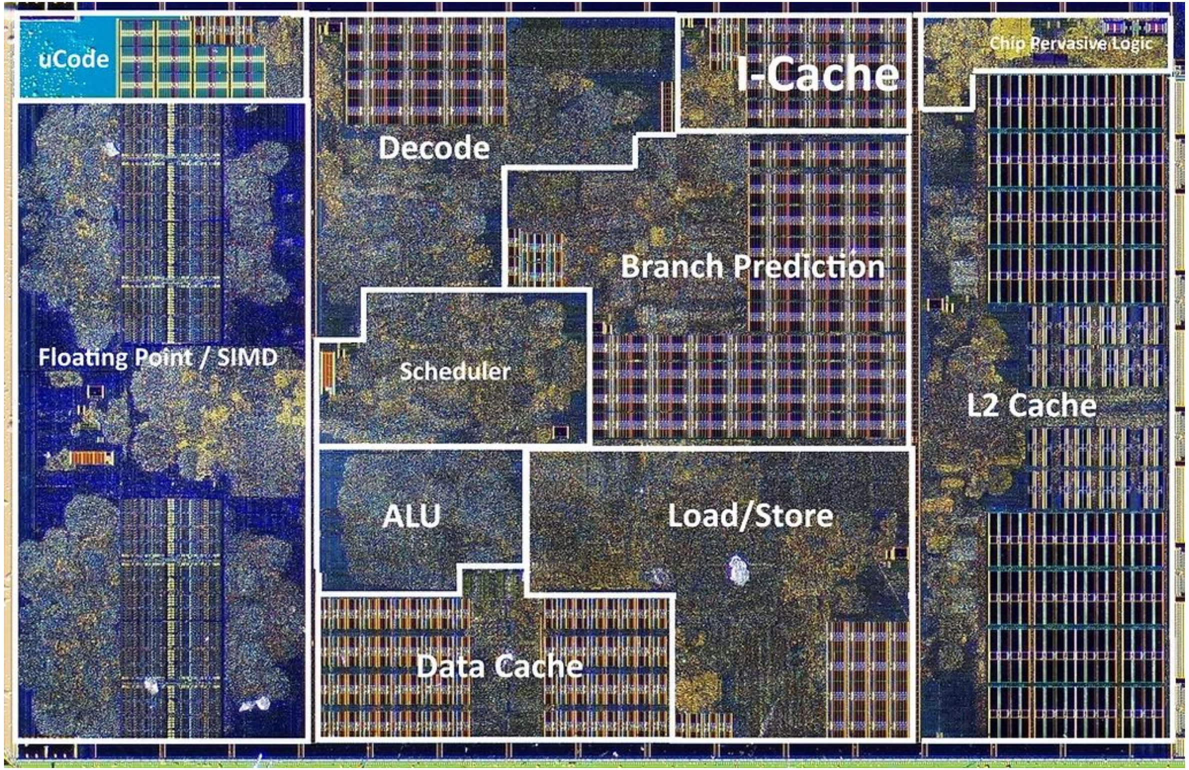
Federico Ficarelli

Anton Lydike
Alban Dutilleul
Chris Vasiladiotis

Sasha Lopoukhine
Markus Böck
Tobias Grosser

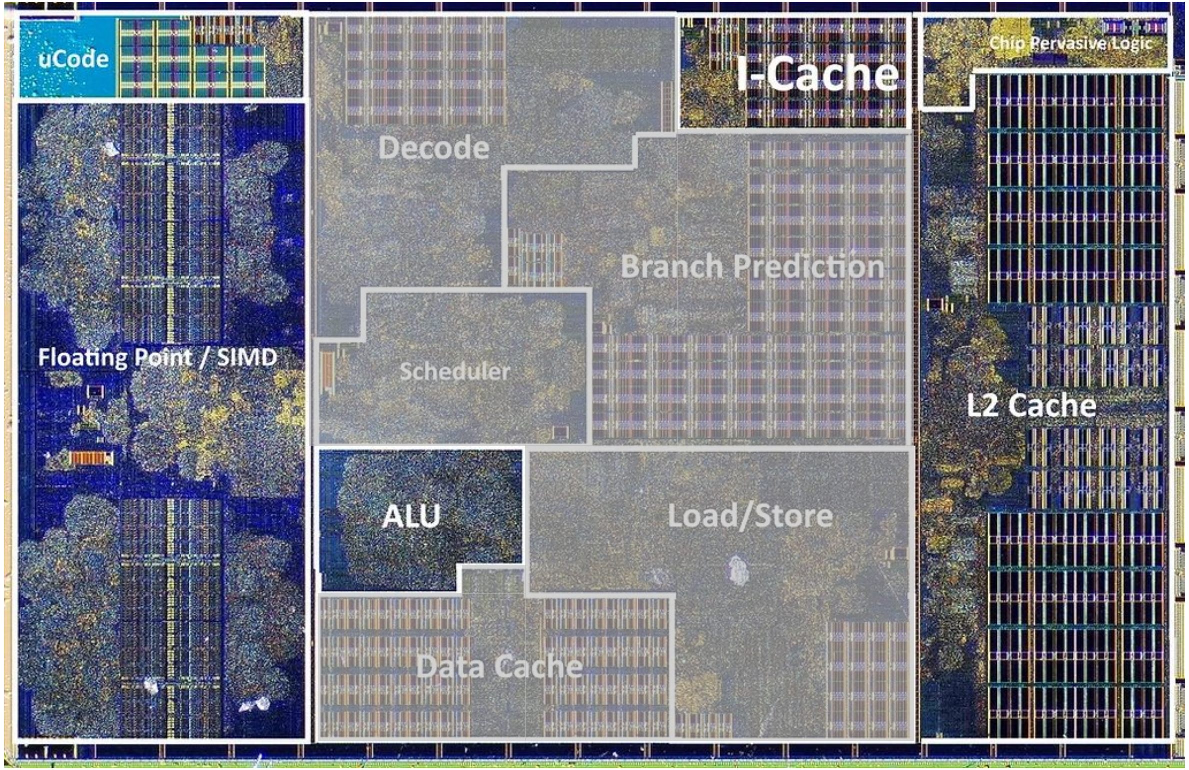
Josse Van Delm

Utilizing CPUs requires complex μ-architecture



Die shot by Fritzchens Fritz, Annotation from AMD Zen 2 slides at ISSCC

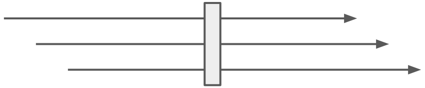
Linalg on CPUs **doesn’t** require complex µ-architecture



Die shot by Fritzchens Fritz, Annotation from AMD Zen 2 slides at ISSCC

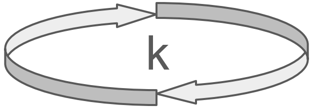
Snitch Cores Shift Complexity from Hardware to Software

~~Out Of Order Execution~~



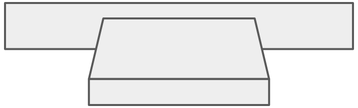
Streaming Registers

~~Branch Prediction~~



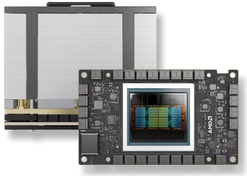
Hardware Loops

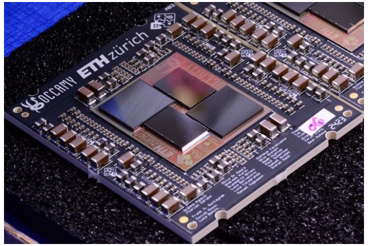
~~Hardware-managed Cache~~



Software-managed Cache

Occamy - A NN Accelerator for Performance/Watt





304 x

Compute Unit

64 Threads

32-Bit VGPRs

64kB LDS

48 x

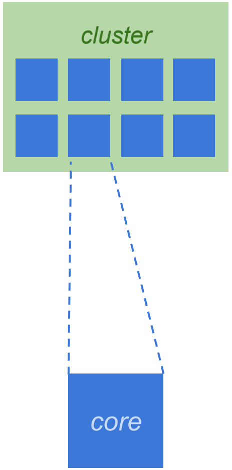
Snitch Cluster

8 Compute Cores

64-Bit VGPRs

128kB TCDM

Low-Level Constraints In High-Level Compilation Passes



128 KB L1
8 Compute Cores
1 DMA Core
DMA - 64 B/s

NN Compiler

Read or Write Streams
3 Streams 64b
4D Access Patterns
Multi-Stage FPU

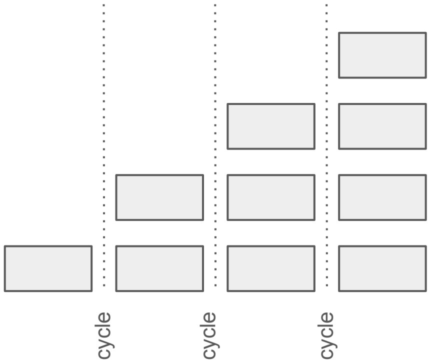
Micro-Kernel Compiler

Objective: Fast Micro-Kernels

$$Z = X_i + Y_i$$

Linear Algebra

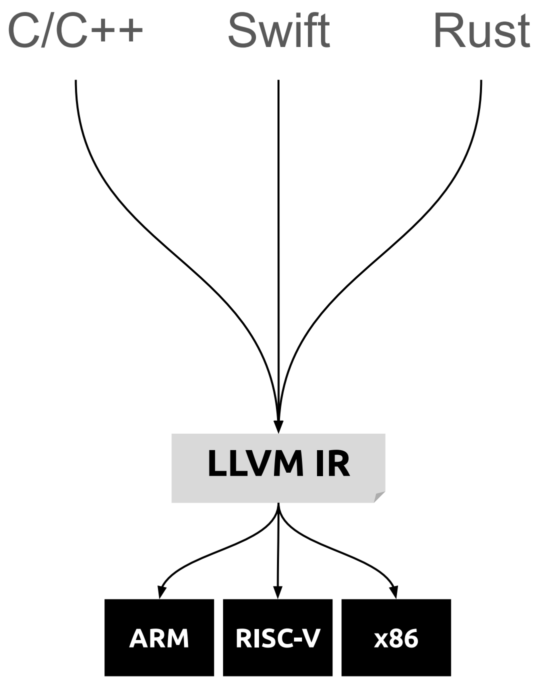
Single Core



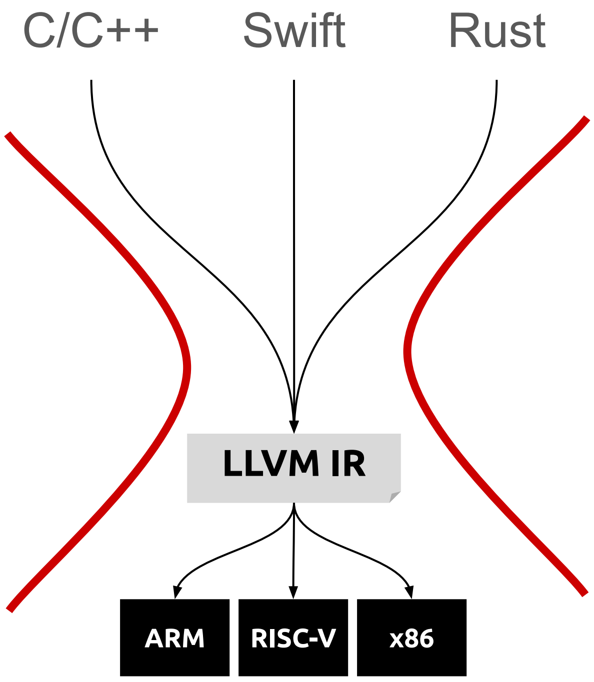
FPU Always Busy

How do we automate this?

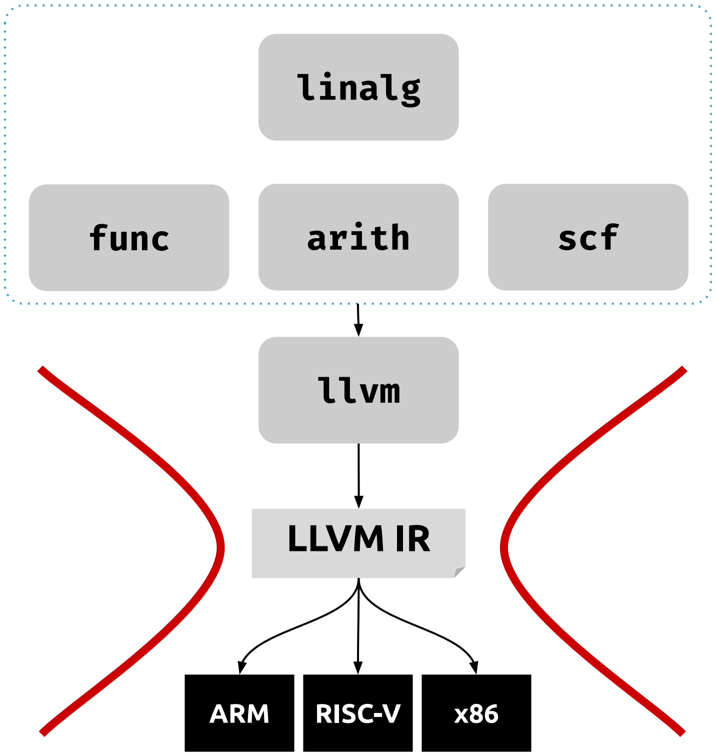
LLVM IR Is Well-Suited for General-Purpose Targets



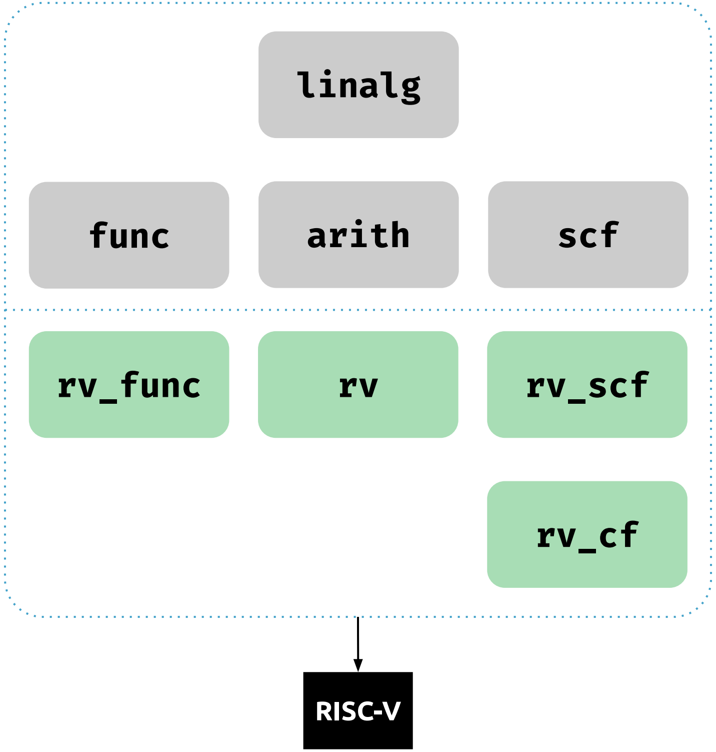
LLVM IR Discards Semantic Information



High-Level Transformations in MLIR



A Multi-Level RISC-V Backend



Low-Level Backend Operations Model Assembly

```
%c1 = riscv.li 1 : !riscv.reg<t0>
              ↓
          li t0, 1
```

```
^0(%0 : !riscv.reg<t3>):
  riscv.label scf_body_0_for
              ↓
       scf_body_0_for:
```

```
%res = riscv.fmadd.d %a, %b, %acc : (
             !riscv.freg<ft0>,
             !riscv.freg<ft1>,
             !riscv.freg<ft2>
         ) -> !riscv.freg<ft0>
                    ↓
       fmadd.d ft0, ft0, ft1, ft2
```

```
riscv_cf.blt %1 : !riscv.reg<t3>,
             %2 : !riscv.reg<t4>,
         ^0(%1 : !riscv.reg<t3>),
         ^1(%1 : !riscv.reg<t3>)
                  ↓
     blt t3, t4, scf_body_0_for
```

Instruction Selection As IR Rewrites

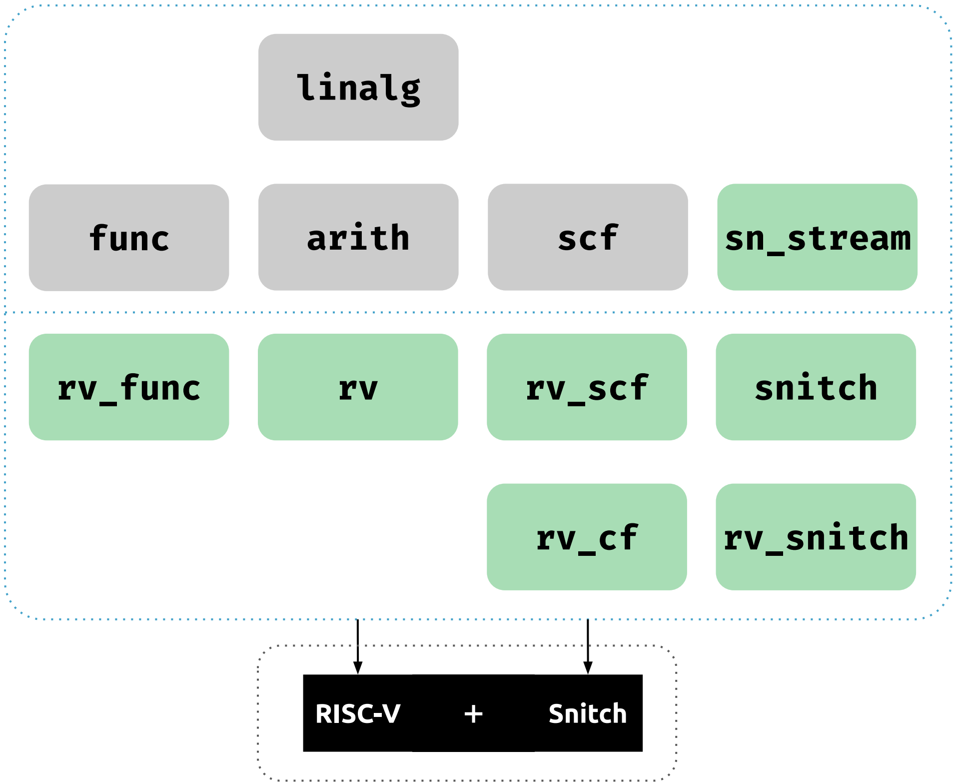
```
%c1 = arith.constant 1 : index
                    ↓
%c1 = riscv.li       1 : !riscv.reg
```

```
%sum  = arith.addi %a, %b : index
                ↓
%sum  = riscv.add  %a, %b :
  (!riscv.reg, !riscv.reg)   -> !riscv.reg
```

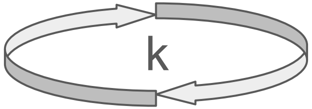
```
%prod = arith.mulf   %c, %d : f64
                ↓
%prod = riscv.fmul.d %c, %d :
     (!riscv.freg, !riscv.freg) -> !riscv.freg
```

```
%res = memref.load %ref[%i] : memref<128xf64>
                         ↓
%c8         = riscv.li 8                    : …
%ptr_offset = riscv.mul %i, %c8             : …
%ptr        = riscv.add %ref, %ptr_offset : …
%res        = riscv.fld %ptr, 0             : …
```

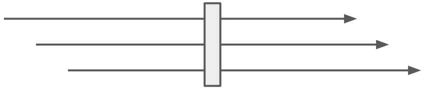
Modular ISA Extensions



ISA Extensions as Dialects



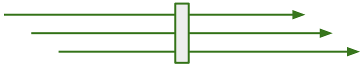

```
%res = riscv_snitch.frep_outer %ub iter_args(%acc = %init) -> !riscv.reg<ft3> {
    …
    riscv_snitch.yield %acc_out : !riscv.reg<ft3>
}
```



```
snitch_stream.streaming_region {
  patterns = [
      #snitch_stream.stride_pattern<ub = [6], strides = [8]>
  ]
} ins(%ptr : !riscv.reg) {
^0(%stream : !stream.readable<!riscv.freg<ft0>>):
   %val = riscv_snitch.read from %stream : !riscv.freg<ft0>
   …
}
```

Separate access from execute

Micro-Kernel Compilation Pipeline (`f64`)



Backend?

Read or Write Streams

Multi-Stage FPU

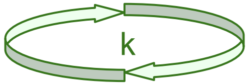
4D Access Patterns

Use Streams

Optimization

Lowering

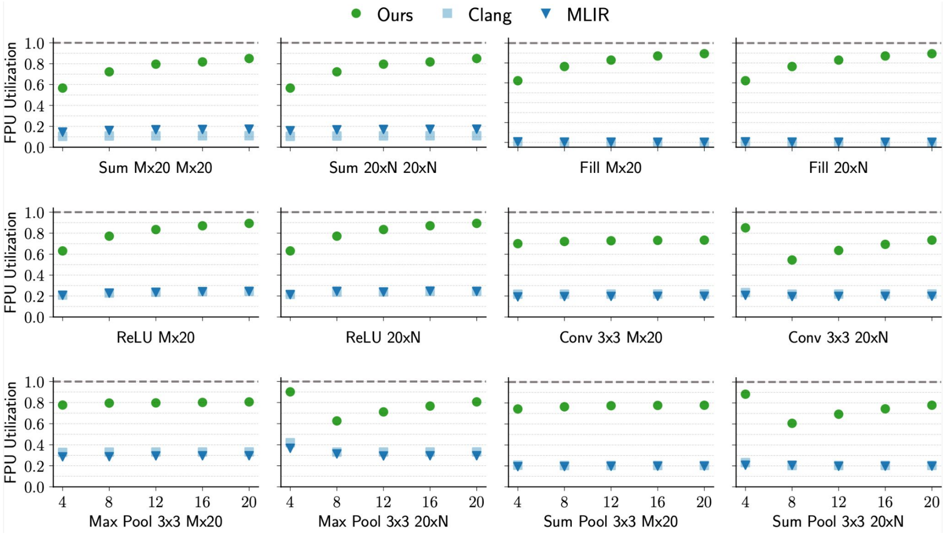
Lower to RISC-V + Snitch

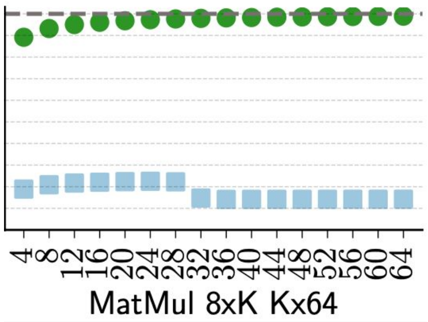


Backend?

Lower towards assembly

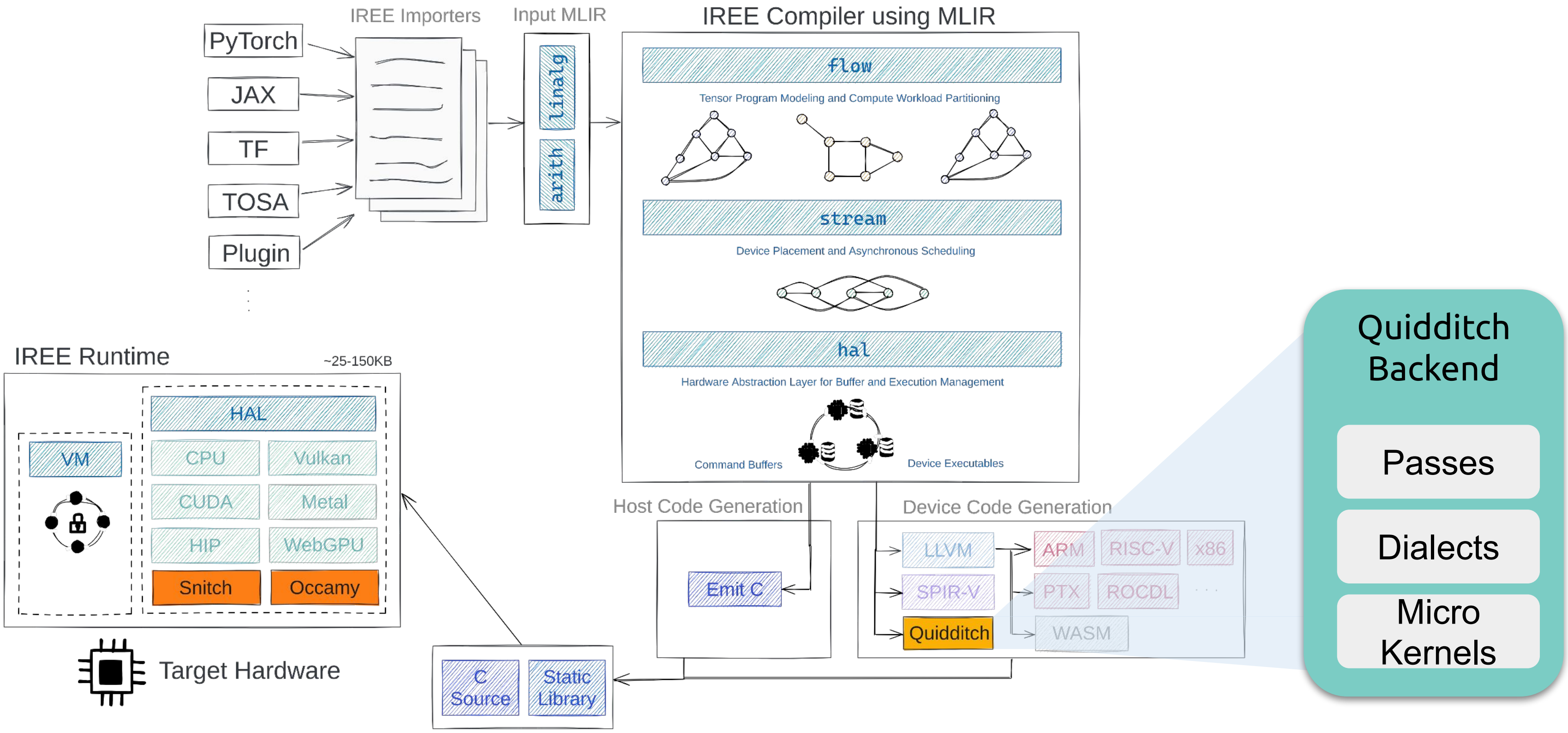
FPU Utilization on f64



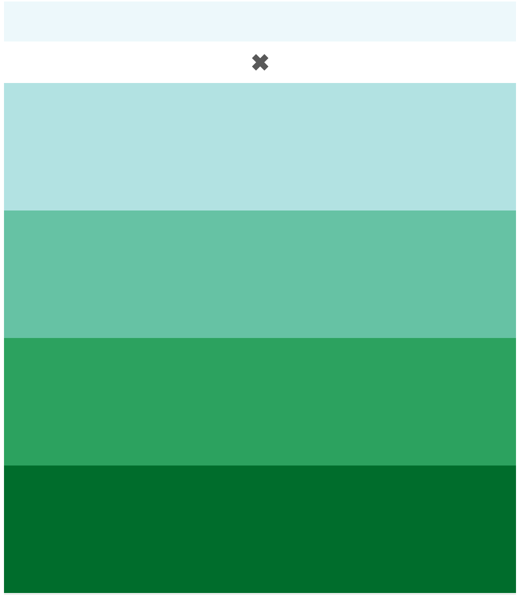


Unfair comparison - Naive C implementation and `linalg` to scf lowering

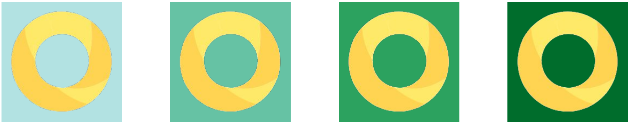
Quidditch Compiler



Tiling enables distribution and promotion

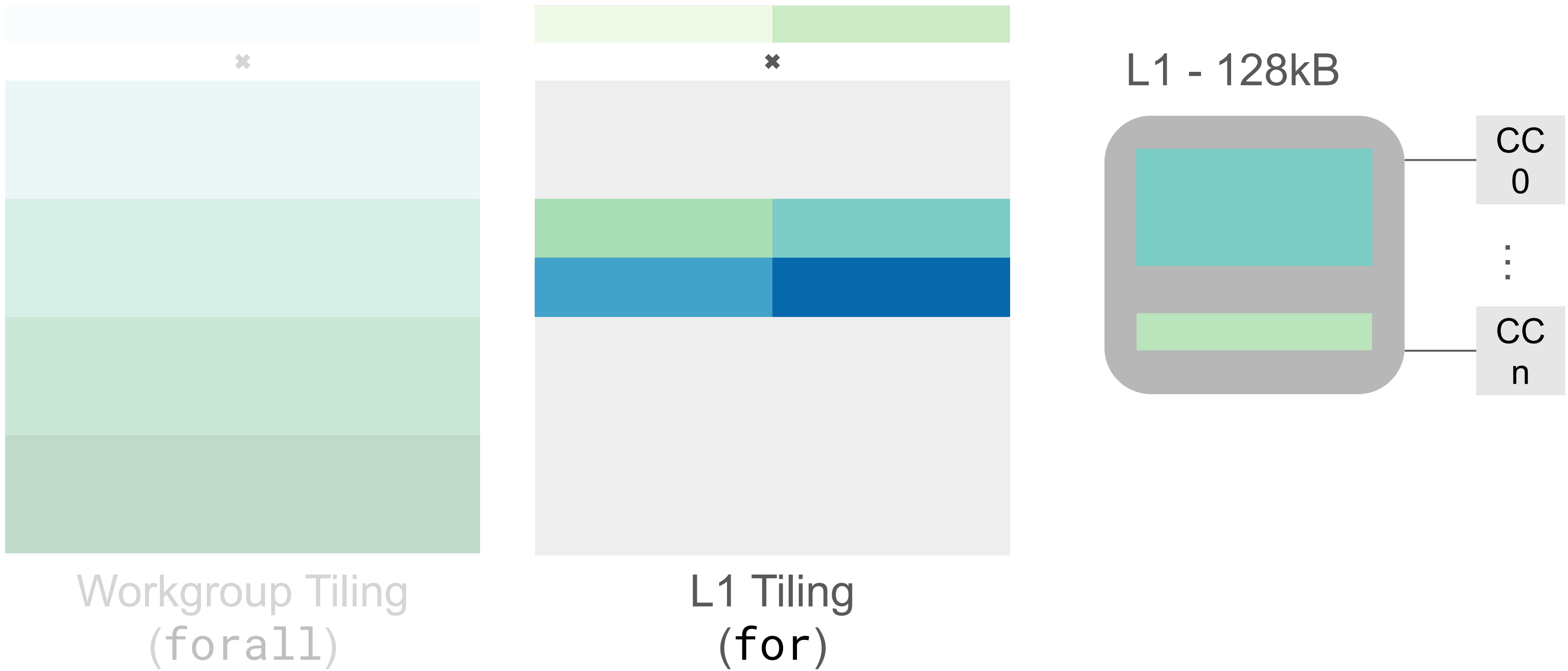


Workgroup Tiling
(`forall`)



Snitch Clusters

Tiling enables distribution and promotion



Tiling enables distribution and promotion

Tensor Promotion

```
%0, %token0 = start_tensor_copy %vec_re_slice to L1
%vec_re_l1 = wait_for_tensor_copy %0 using %token0

%1, %token1 = start_tensor_copy %weights_tile to L1
%weights_l1 = wait_for_tensor_copy %1 using %token1

%2, %token2 = start_tensor_copy %res_tile to L1
%res_l1 = wait_for_tensor_copy %2 using %token2

%output = linalg.matmul_transpose_b
 ins(%vec_re_l1, %weights_l1 : tensor<1x200>, tensor<40x200>)
 outs(%res_l1) -> tensor<1x40>
```

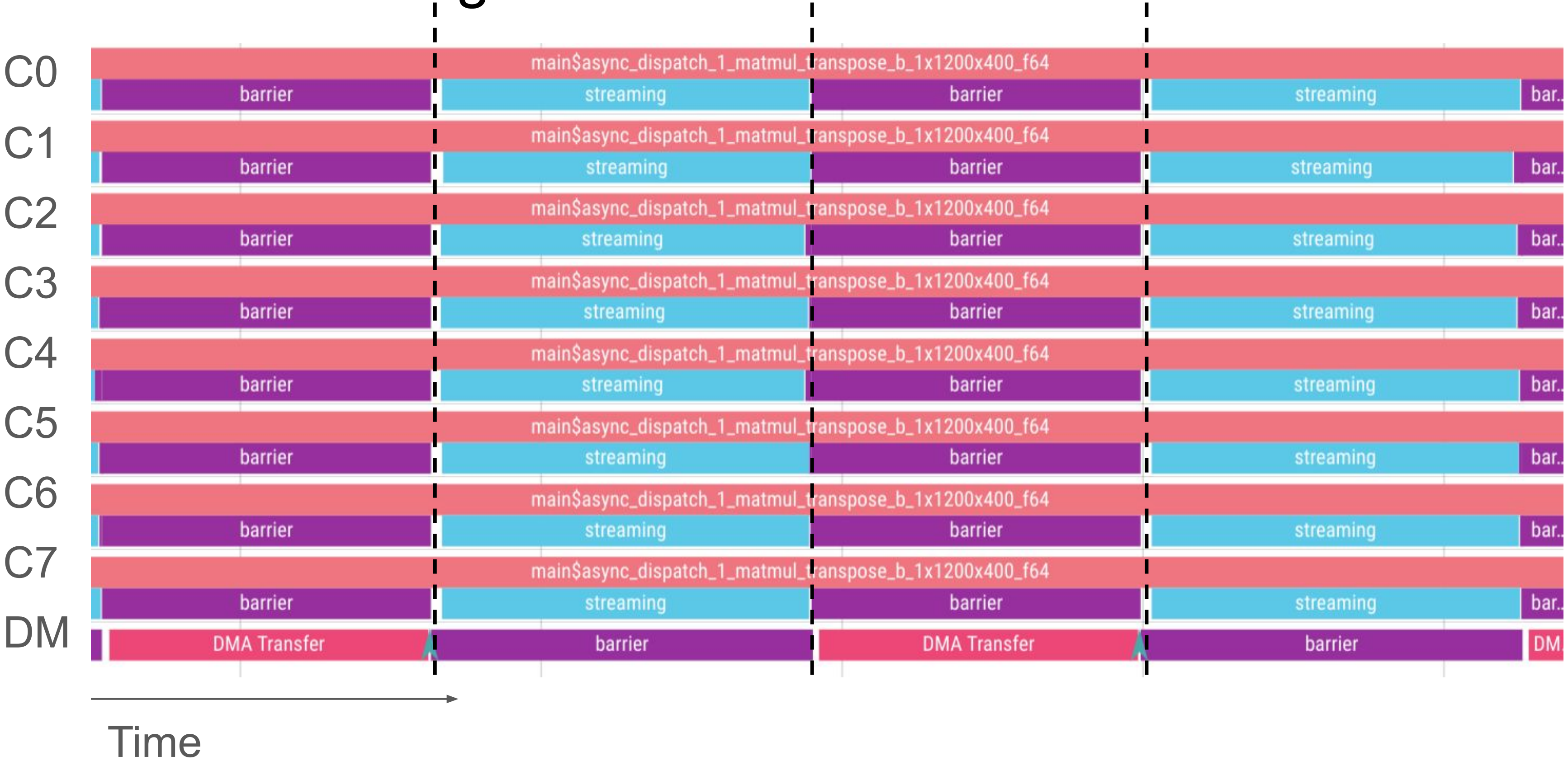
Tensor Promotion - Cleanup

```
%vec_re_slice = tensor.extract_slice %vector[0, %arg0]
%weights_re_slice = tensor.extract_slice %weights[0, %arg0]
%20 = scf.for %arg2 = 0 to 1200 step 40 iter_args(%arg3 = %arg1) {
  %weights_tile = tensor.extract_slice %weights_re_slice[%arg2, 0]
  %res_tile = tensor.extract_slice %arg3[0, %arg2]

  %0, %token0 = start_tensor_copy %vec_re_slice to L1
  %vec_re_l1 = wait_for_tensor_copy %0 using %token0
  %1, %token1 = start_tensor_copy %weights_tile to L1
  %weights_l1 = wait_for_tensor_copy %1 using %token1
  %2, %token2 = start_tensor_copy %res_tile to L1
  %res_l1 = wait_for_tensor_copy %2 using %token2

  %21 = linalg.matmul_transpose_b
   ins(%vec_re_l1, %weights_l1 : tensor<1x200xf64>, tensor<40x200xf64>)
   outs(%res_l1) -> tensor<1x40xf64>
  %inserted_slice = tensor.insert_slice %21 into %arg3[0, %arg2]
  scf.yield %inserted_slice : tensor<1x1200>
}
```

Tensor Promotion - Cleanup

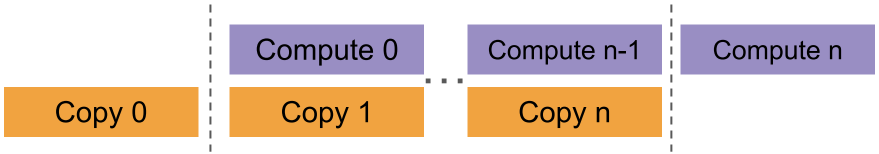
```
%vec_re_slice = tensor.extract_slice %vector[0, %arg0]
%weights_re_slice = tensor.extract_slice %weights[0, %arg0]
%0, %token0 = start_tensor_copy %vec_re_slice to L1
%vec_re_l1 = wait_for_tensor_copy %0 using %token0
%20 = scf.for %arg2 = 0 to 1200 step 40 iter_args(%arg3 = %arg1) {
  %weights_tile = tensor.extract_slice %weights_re_slice[%arg2, 0]
  %res_tile = tensor.extract_slice %arg3[0, %arg2]

  %1, %token1 = start_tensor_copy %weights_tile to L1
  %weights_l1 = wait_for_tensor_copy %1 using %token1
  %2, %token2 = start_tensor_copy %res_tile to L1
  %res_l1 = wait_for_tensor_copy %2 using %token2

  %21 = linalg.matmul_transpose_b
   ins(%vec_re_l1, %weights_l1 : tensor<1x200xf64>, tensor<40x200xf64>)
   outs(%res_l1) -> tensor<1x40xf64>
  %inserted_slice = tensor.insert_slice %21 into %arg3[0, %arg2]
  scf.yield %inserted_slice : tensor<1x1200>
 }
```

Double Buffering



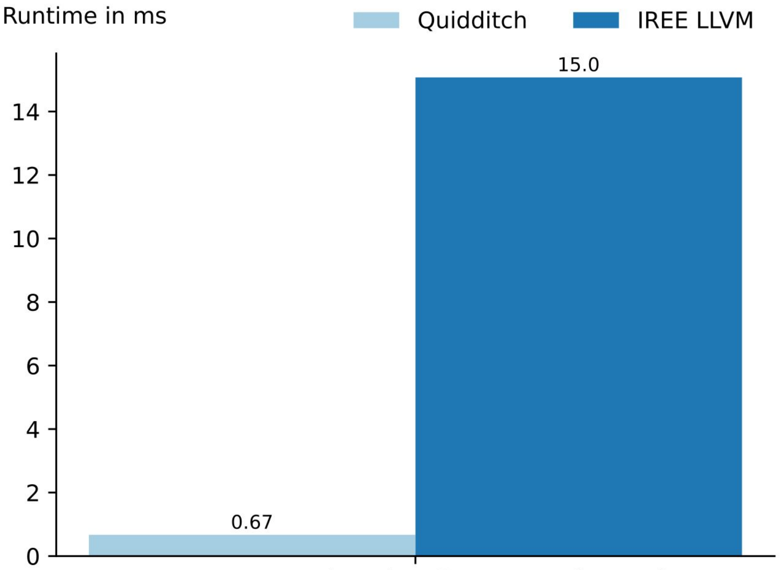
Double Buffering

Double Buffering

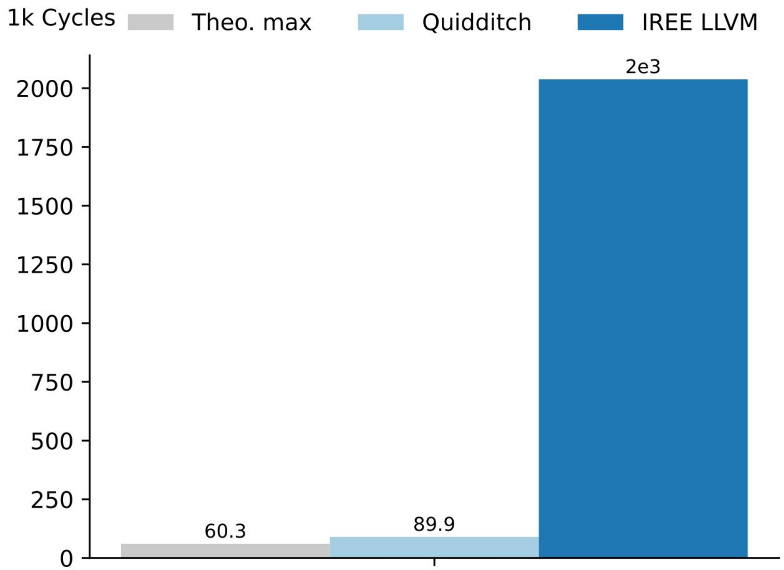



```
%20 = pipeline 0 to 1200 step 40 iter_args(%arg1) -> (tensor<1x1200>) {
^bb0(%iv, %arg3):
  %weights_tile = tensor.extract_slice %weights_re_slice[%iv, 0]
  %res_tile = tensor.extract_slice %arg3[0, %arg2]
  %1, %token1 = start_tensor_copy %weights_tile to L1
  %2, %token2 = start_tensor_copy %res_tile to L1
  pipeline_yield %1, %token1, %2, %token2
} {
^bb1(%iv, %1_in, %token1_in, %2_in, %token2_in):
  %weights_l1 = wait_for_tensor_copy %1_in using %token1_in
  %res_l1 = wait_for_tensor_copy %2_in using %token2_in
  %21 = linalg.matmul_transpose_b
    ins(%vec_re_l1, %weights_l1 : tensor<1x200xf64>, tensor<40x200xf64>)
    outs(%res_l1) -> tensor<1x40xf64>
  %inserted_slice = tensor.insert_slice %21 into %arg3[0, %iv]
  pipeline_yield %inserted_slice : tensor<1x1200>
}
```

Evaluation - Snitch Cluster (8 compute cores)

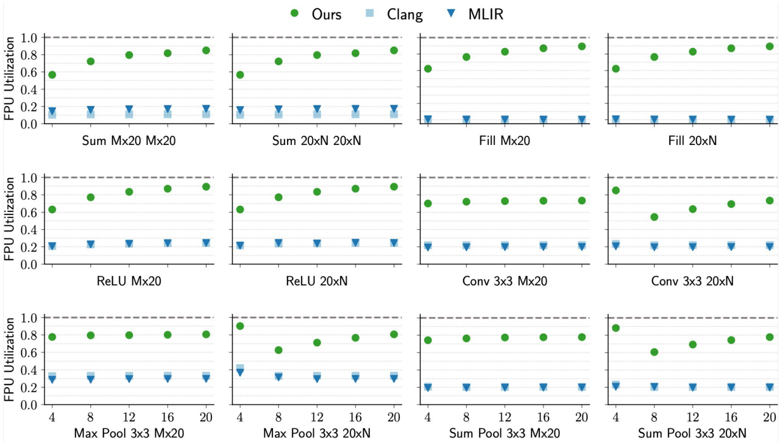


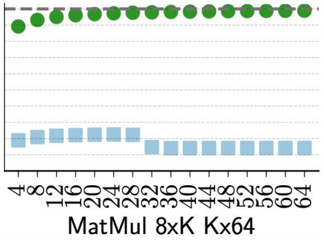
NsNet2 - Single frame denoise
22.4x Speedup

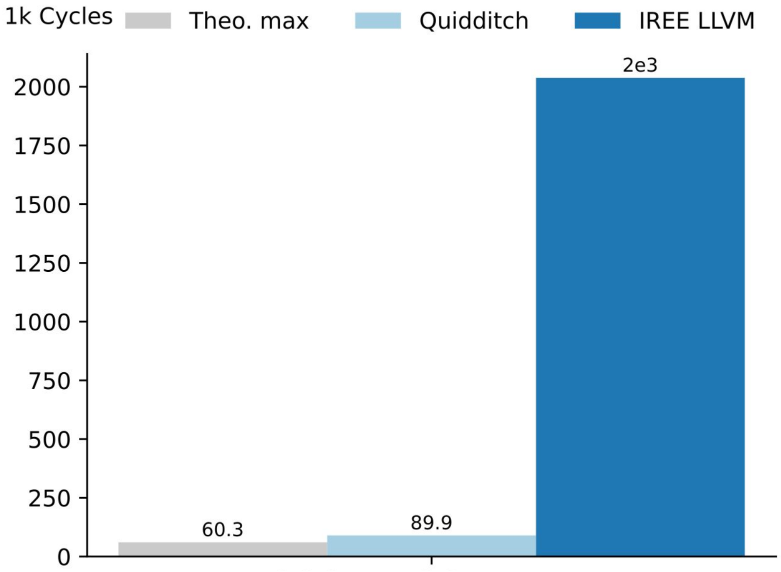


1200 x 400 Matvec
22.7x Speedup

High-Level Compilers can Fully Exploit Target Hardware







1200 x 400 Matvec
22.7x Speedup