# **ScaleHLS:** A New Scalable High-Level Synthesis Framework on Multi-Level Intermediate Representation

Hanchen Ye[1], Cong Hao[2], Jianyi Cheng[3], Hyunmin Jeong[1], Jack Huang[1],
Stephen Neuendorffer[4], Deming Chen[1]

[1]University of Illinois at Urbana-Champaign, [2]Georgia Institute of Technology,
[3]Imperial College London, [4]Xilinx Inc.

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Georgia Tech

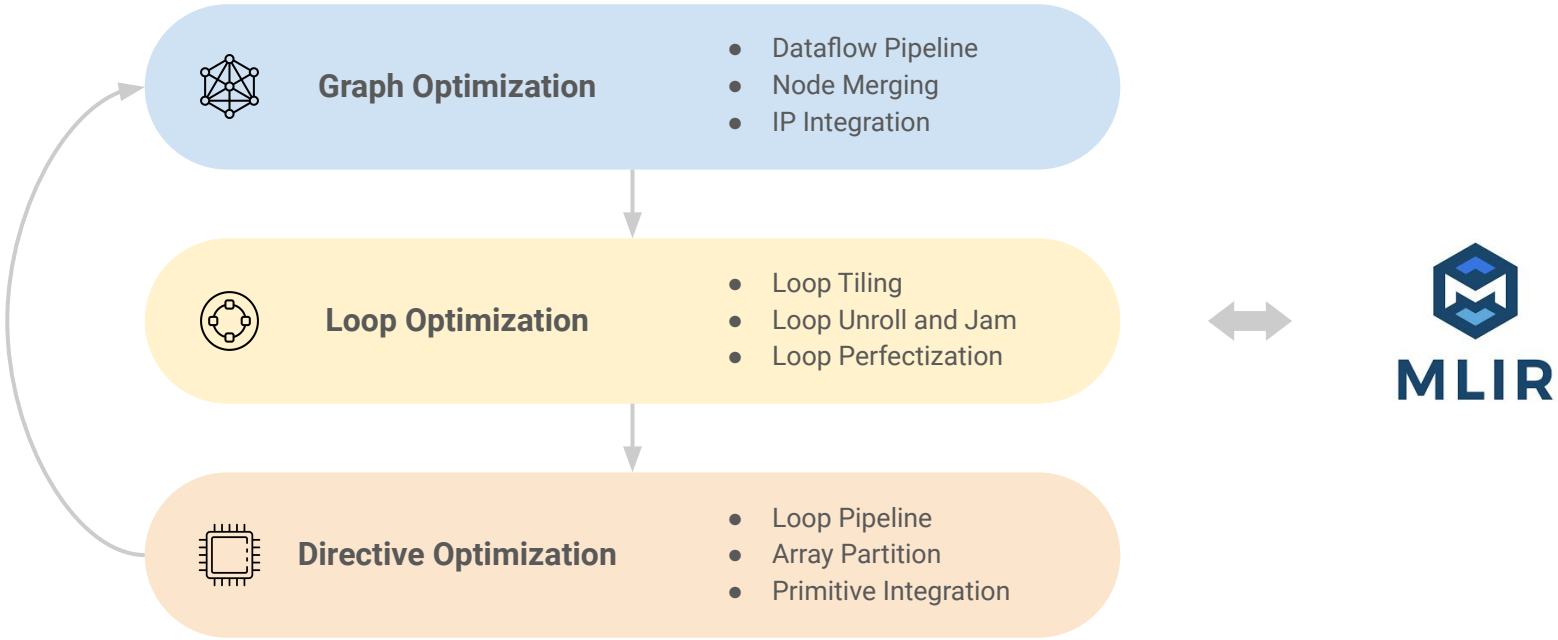
Imperial College London

XILINX

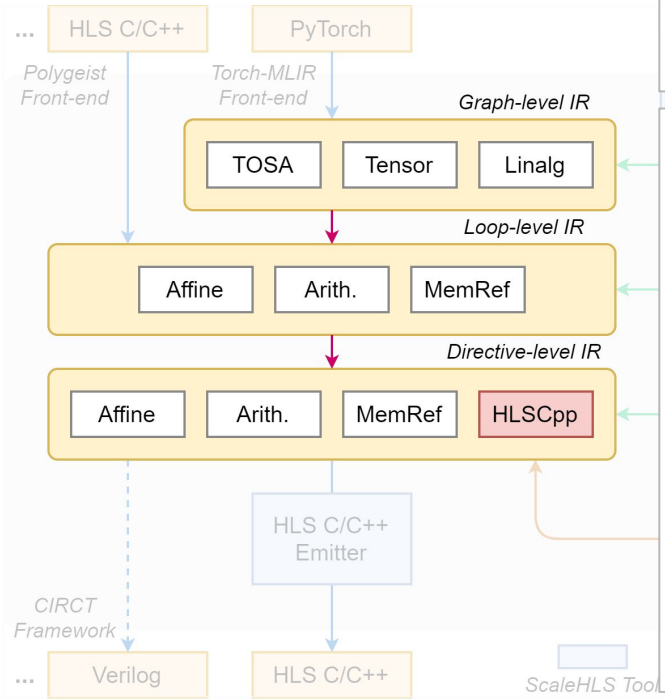
# Outline

- Motivation
- ScaleHLS Framework
- Evaluation of C/C++ Kernels
- Evaluation of PyTorch Models
- New Features
- ScaleHLS is Open-Sourced!

# **Motivation:** Marry HLS and MLIR



# **ScaleHLS Framework:** Representation



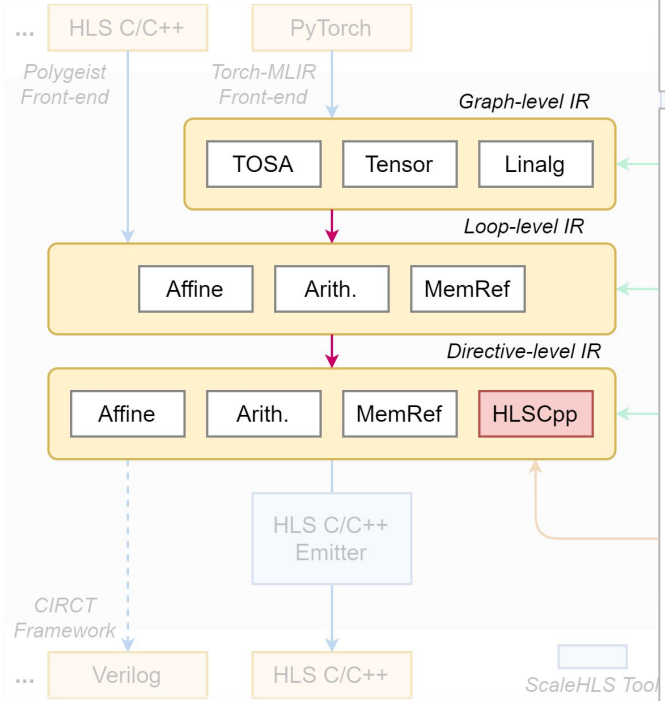

```
%O = "tosa.conv2d"(%I, %W, ...) {...} : (tensor<1x32x32x3xi8>, tensor<64x3x3x3xi8>, ...) ->
                                        tensor<1x30x30x64xi8>
```

Conv2D Graph-level MLIR

```
affine.for %x = 0 to 30 {
  affine.for %y = 0 to 30 {
    affine.for %k = 0 to 64 {
      affine.for %r = 0 to 3 {
        affine.for %s = 0 to 3 {
          affine.for %c = 0 to 3 {
            %i = affine.load %I[0, %x + %r, %y + %s, %c] : memref<1x32x32x3xi8>
            %w = affine.load %W[%r, %s, %c, %k] : memref<3x3x3x64xi8>
            %o = affine.load %O[0, %x, %y, %k] : memref<1x30x30x64xi8>
            %mul = arith.muli %i, %w : i8
            %add = arith.addi %o, %mul : i8
            affine.store %add, %O[0, %x, %y, %k] : memref<1x30x30x64xi8>
          }
        }
      }
    }
  }
}
```

Conv2D Loop-level MLIR

# **ScaleHLS Framework:** Representation



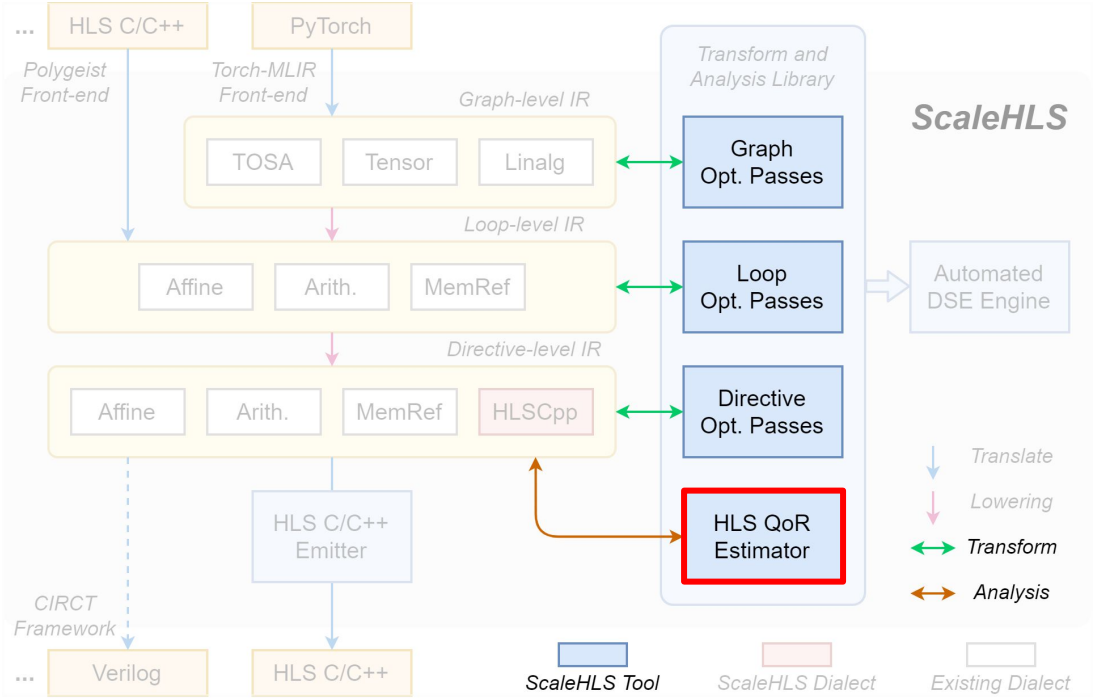

```
%O = "tosa.conv2d"(%I, %W, ...) {...} : (tensor<1x32x32x3xi8>, tensor<64x3x3x3xi8>, ...) ->
                                        tensor<1x30x30x64xi8>
```

**Conv2D Graph-level MLIR**

```
affine.for %x = 0 to 30 {
  affine.for %y = 0 to 30 {
    affine.for %k = 0 to 64 step 2 {
      affine.for %r = 0 to 3 {
        affine.for %s = 0 to 3 {
          affine.for %c = 0 to 3 {
            %i = affine.load %I[0, %x + %r, %y + %s, %c] : memref<1x32x32x3xi8>
            %w = vector.transfer_read %W[%r, %s, %c, %k], : memref<3x3x3x64xi8>, vector<2xi8>
            ... ...
            %mul = "hlscpp.mul_prim"(%i, %w) : (i8, vector<2xi8>) -> vector<2xi16>
            ... ...
            %mul32 = "hlscpp.cast_prim"(%mul) : (vector<2xi16>) -> vector<2xi32>
            ... ...
          } {loop_directive = #hlscpp.ld<pipeline=true, targetII=1, ...}
        }
      }
    }
  }
}
```

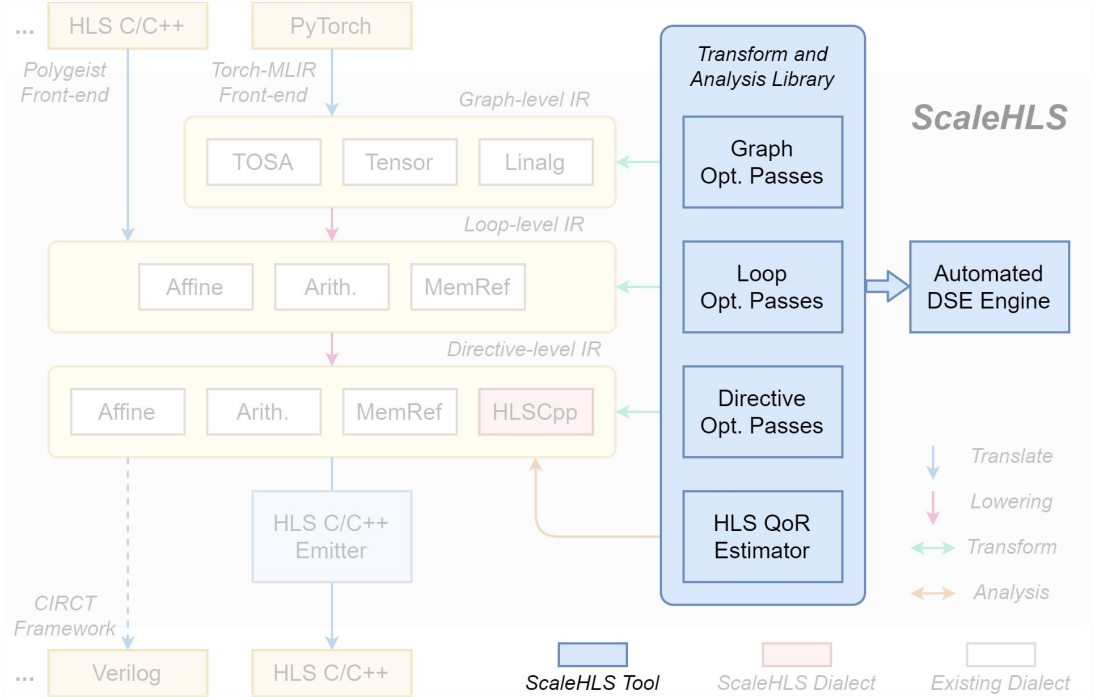
**Conv2D Directive-level MLIR**

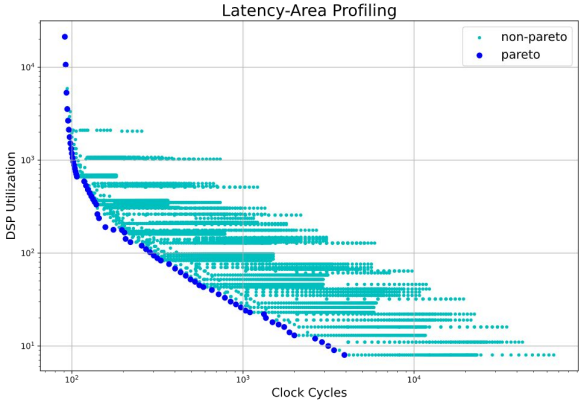
# **ScaleHLS Framework:** Optimization

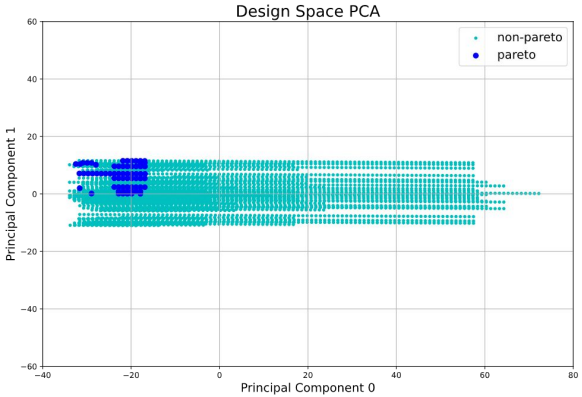



| Category | ScaleHLS Passes |
|---|---|
| **Graph** | `SimplifyTosaGraph`<br>`LegalizeDataflow`<br>`SplitFunction`<br>`CreateRuntimeMain` |
| **Loop** | `MaterializeReduction`<br>`AffineLoopPerfectization`<br>`RemoveVariableBound`<br>`AffineLoopOrderOpt`<br>`AffineLoopUnrollAndPipeline`<br>`ReduceInitialInterval` |
| **Simplification** | `SimplifyAffineIf`<br>`AffineStoreForward`<br>`SimplifyMemrefAccess` |
| **Directive** | `ArrayPartition`<br>`LoopPipeline`<br>`FunctionPipeline`<br>`CreateHLSCppPrimitive` |

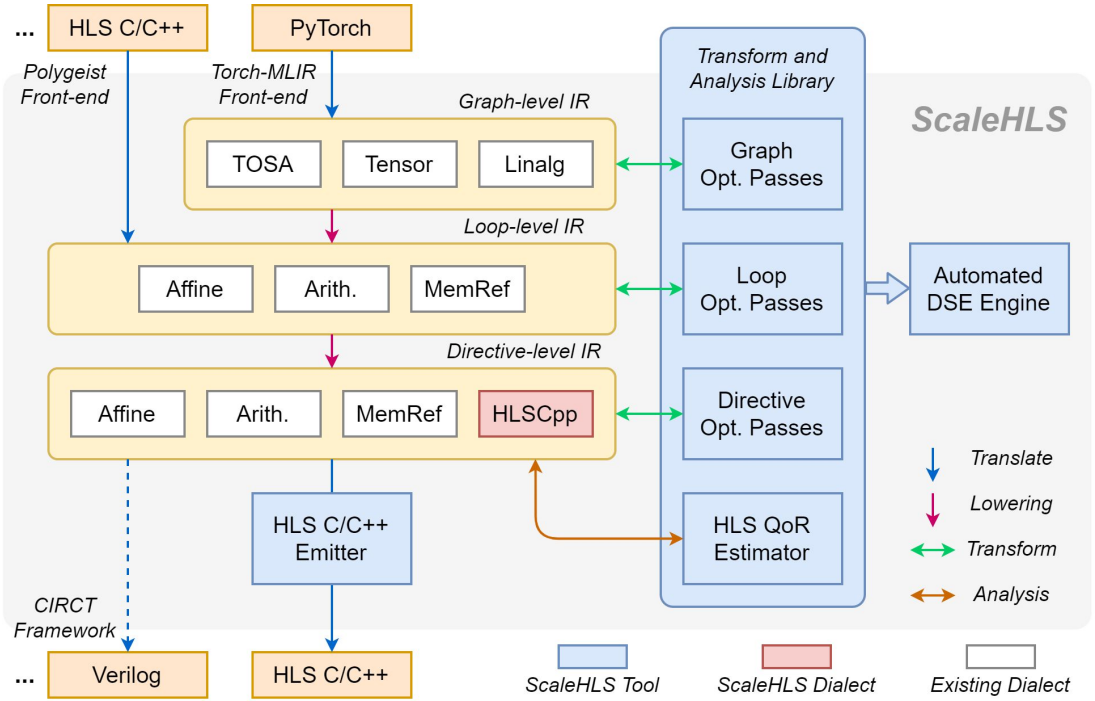
# **ScaleHLS Framework:** Exploration



# **ScaleHLS Framework:** Integration




**Inputs**

**C/C++** Polygeist

**PyTorch** Torch-MLIR

**Outputs**

**C/C++** C/C++ Emitter

**Verilog** CIRCT (work-in-progress)

# **Evaluation of C/C++ Kernels:** Design Space Exploration

| Kernel | Speedup | Loop Perfectize | Variable Loop Bound | Loop Permute Order | Loop Unroll Sizes | Loop Pipeline II | Array Partition Factors |
|---|---|---|---|---|---|---|---|
| **BICG** | 41.7x | No | No | [1, 0] | [16, 8] | 43 | A:[8, 16], s:[16], q:[8], p:[16], r:[8] |
| **GEMM** | 768.1x | Yes | No | [1, 2, 0] | [8, 1, 16] | 3 | C:[1, 16], A:[1, 8], B:[8, 16] |
| **GESUMMV** | 199.1x | Yes | No | [1, 0] | [8, 16] | 9 | A:[16, 8], B:[16, 8], tmp:[16], x:[8], y:[16] |
| **SYR2K** | 384.0x | Yes | Yes | [1, 2, 0] | [8, 4, 4] | 8 | C:[4, 4], A:[4, 8], B:[4, 8] |
| **SYRK** | 384.1x | Yes | Yes | [1, 2, 0] | [64, 1, 1] | 3 | C:[1, 1], A:[1, 64] |
| **TRMM** | 590.9x | Yes | Yes | [1, 2, 0] | [4, 4, 32] | 13 | A:[4, 4], B:[4, 32] |
| **Spam-Filter** | 72.6x | No | No | [0], [0], [0] | [32], [32], [32] | 7, 1, 1 | data:[32], label:[32], theta:[32] |
| **3MM** | 230.2x | Yes | No | [1, 2, 0], [1, 2, 0], [1, 2, 0] | [3, 8, 2], [2, 10, 2], [5, 5, 5] | 1, 1, 3 | A:[8, 3], B:[3, 2], C:[10, 2], D:[2, 2], E:[8, 5], F:[10, 5], G: [5, 5] |

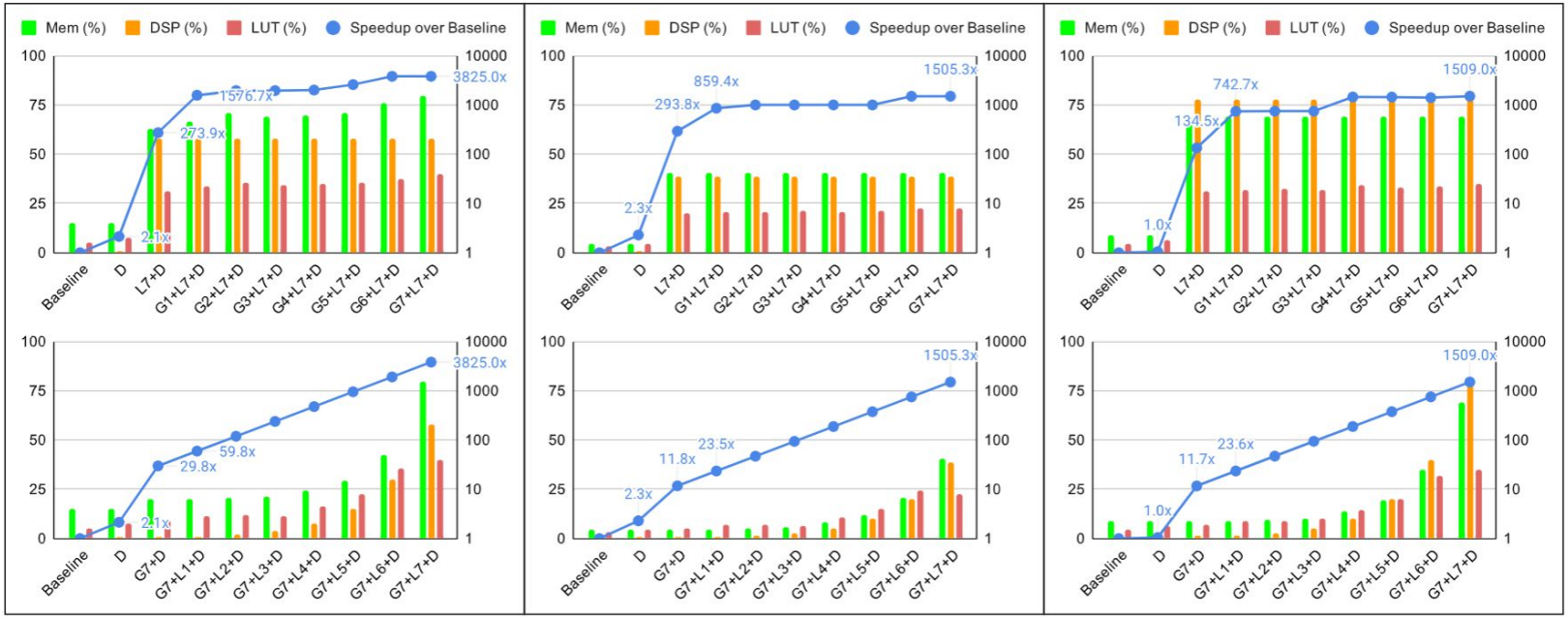
- The target platform is Xilinx XC7Z020 FPGA. Benchmarks are from PolyBench-C and Rosetta.
- Optimization parameters are automatically selected by the design space exploration (DSE) engine.
- The speedups are compared to the original computation kernels without DSE or any manual optimizations.

# **Evaluation of PyTorch Models:** Multi-Level Optimization

| Model | Speedup | Runtime (seconds) | Memory (SLR Util. %) | DSP (SLR Util. %) | LUT (SLR Util. %) | Our DSP Effi. (OP/Cycle/DSP) | DSP Effi. of TVM-VTA [49] |
|---|---|---|---|---|---|---|---|
| **ResNet-18** | 3825.0× | 60.8 | 91.7Mb (79.5%) | 1326 (58.2%) | 157902 (40.1%) | 1.343 | 0.344 |
| **VGG-16** | 1505.3× | 37.3 | 46.7Mb (40.5%) | 878 (38.5%) | 88108 (22.4%) | 0.744 | 0.296 |
| **MobileNet** | 1509.0× | 38.1 | 79.4Mb (68.9%) | 1774 (77.8%) | 138060 (35.0%) | 0.791 | 0.468 |

- The target platform is one SLR (super logic region) of Xilinx VU9P FPGA.
- The PyTorch models are parsed into ScaleHLS and optimized at multiple levels, including graph, loop, and directive optimizations.
- The speedups are compared to the baseline designs, which are compiled from PyTorch to HLS C/C++ through ScaleHLS but without the multi-level optimization applied.

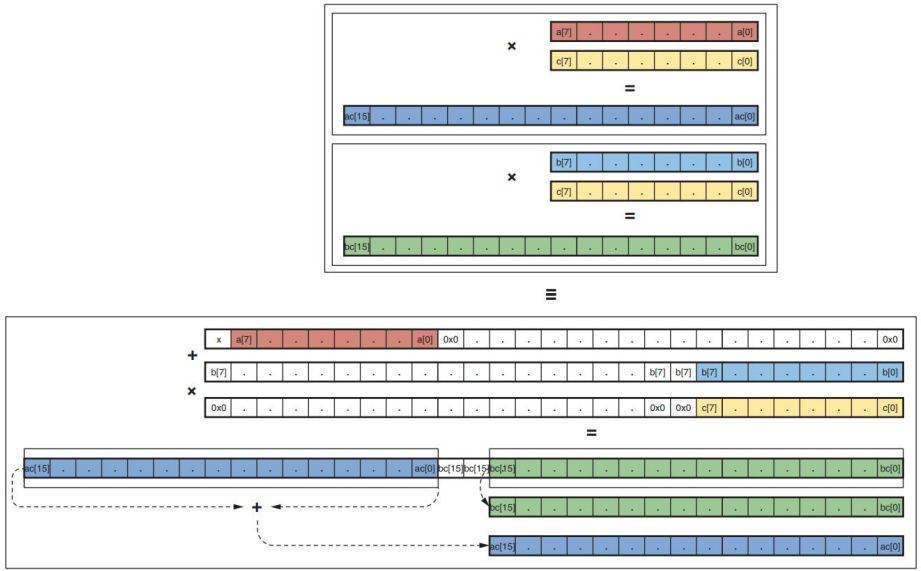
# **Evaluation of PyTorch Models:** Ablation Study

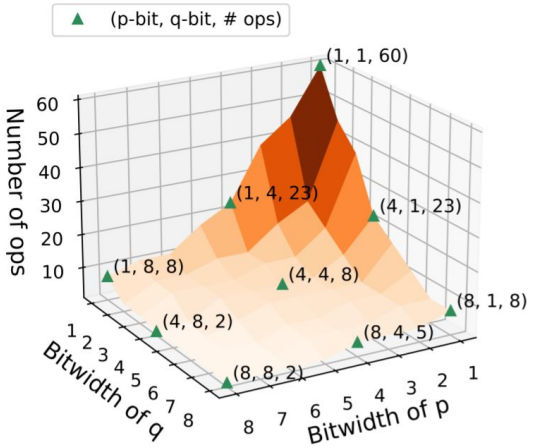



**(a)** ResNet-18 **(b)** VGG-16 **(c)** MobileNet

*D*, *L{n}*, and *G{n}* denote directive, loop, and graph optimizations, respectively. Larger *n* indicates larger loop unrolling factor and finer dataflow granularity for loop and graph optimizations, respectively.

# **New Features:** Automatic Multiplication Packing

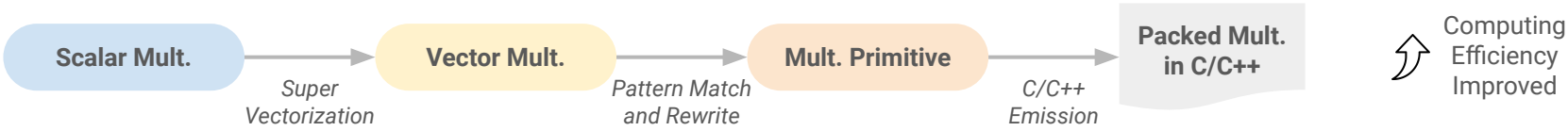



**Right Figure Source:** Liu, Xinheng, et al. "HiKonv: High Throughput Quantized Convolution With Novel Bit-wise Management and Computation." 2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC). 2022.

**Left Figure Source:** Xilinx WP486: Deep Learning with INT8 Optimization on Xilinx Devices

# **New Features:** Automatic Multiplication Packing (Con't)




```
affine.for %x = 0 to 30 {
  affine.for %y = 0 to 30 {
    affine.for %k = 0 to 64 {
      affine.for %r = 0 to 3 {
        affine.for %s = 0 to 3 {
          affine.for %c = 0 to 3 {
            %i = affine.load %I[0, %x + %r, %y + %s, %c] : ...
            %w = affine.load %W[%r, %s, %c, %k] : ...
            %o = affine.load %O[0, %x, %y, %k] : ...
            %mul = arith.muli %i, %w : i8
            %add = arith.addi %o, %mul : i8
            affine.store %add, %O[0, %x, %y, %k] : ...
          }
        }
      }
    }
  }
}
```

Conv2D Loop-level MLIR

```
affine.for %x = 0 to 30 {
  affine.for %y = 0 to 30 {
    affine.for %k = 0 to 64 step 2 {
      affine.for %r = 0 to 3 {
        affine.for %s = 0 to 3 {
          affine.for %c = 0 to 3 {
            %i = affine.load %I[0, %x + %r, %y + %s, %c] : ...
            %w = vector.transfer_read %W[%r, %s, %c, %k], : ..., vector<2xi8>
            ... ...
            %mul = "hlscpp.mul_prim"(%i, %w) : (i8, vector<2xi8>) -> vector<2xi16>
            ... ...
            %mul32 = "hlscpp.cast_prim"(%mul) : (vector<2xi16>) -> vector<2xi32>
            ... ...
          } {loop_directive = #hlscpp.ld<pipeline=true, targetII=1, ...}
        }
      }
    }
  }
}
```

Conv2D Directive-level MLIR

# ScaleHLS is Open-Sourced!

**GitHub Repository:** https://github.com/hanchenye/scalehls

**HPCA'22 Paper:** https://arxiv.org/abs/2107.11673

More new features will be available in March, stay tuned!

**For HLS Researchers**

1. Rapidly implement new HLS optimization algorithms on top of the multi-level IR
2. Investigate new DSE algorithms using the transform and analysis library
3. Rapidly build an end-to-end HLS optimization flow and demonstrate your awesome works!

**For HLS Users**

1. Optimize HLS designs using the multi-level optimization passes
2. Avoid premature design choices by using the QoR estimator to estimate the latency and utilization
3. Find optimized HLS designs with the automated DSE engine

# Questions?

Email: hanchen8@illinois.edu

Website: hanchenye.com