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.

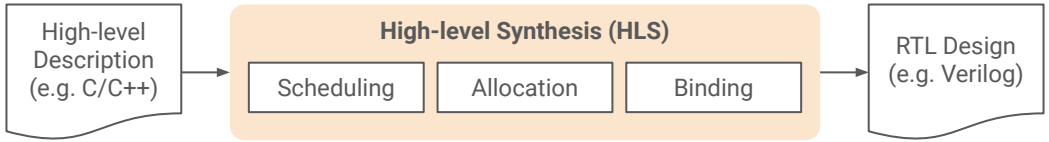
Outline

- Motivation
- ScaleHLS Framework
  - Integration
  - Representation
  - Optimization
  - Exploration
- Evaluation Results
  - C/C++ Kernels
  - PyTorch Models
- Conclusion

ScaleHLS GitHub Repository

https://github.com/hanchenye/scalehls

Motivation: Challenges of HLS



High-level Synthesis (HLS) is great

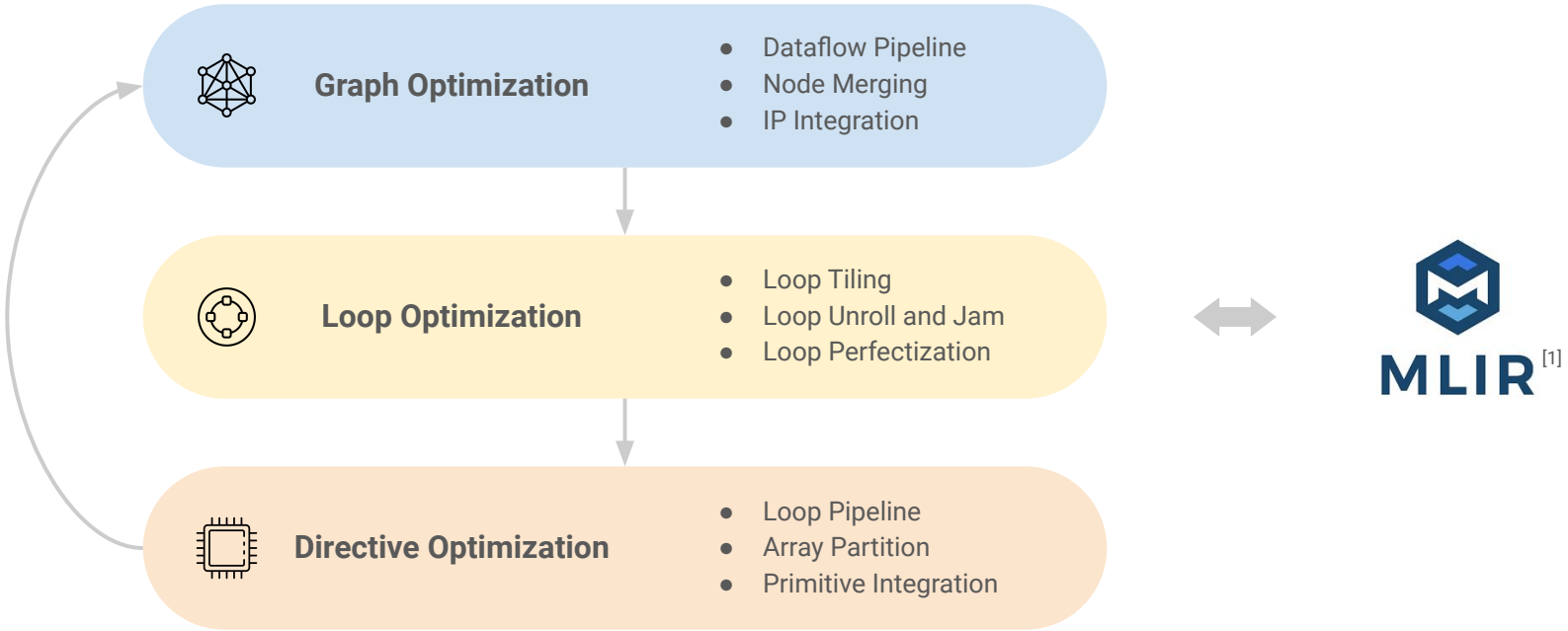
- **Reduce design complexity:** Code density can be reduced by 7x - 8x moving from RTL to C/C++ [1]
- **Improve design productivity:** Get to working designs faster and reduce time-to-market [1]
- **Identify performance-area trade-offs:** Implement design choices quickly and avoid premature optimization [2]

Designing HLS accelerator is challenging

- **Friendly to experts:** Rely on the designers writing 'good' code to achieve high design quality [3]
- **Large design space:** Different combinations of applicable optimizations for large-scale designs [2]
- **Correlation of design factors:** It is difficult for human to discover the complicated correlations [4]

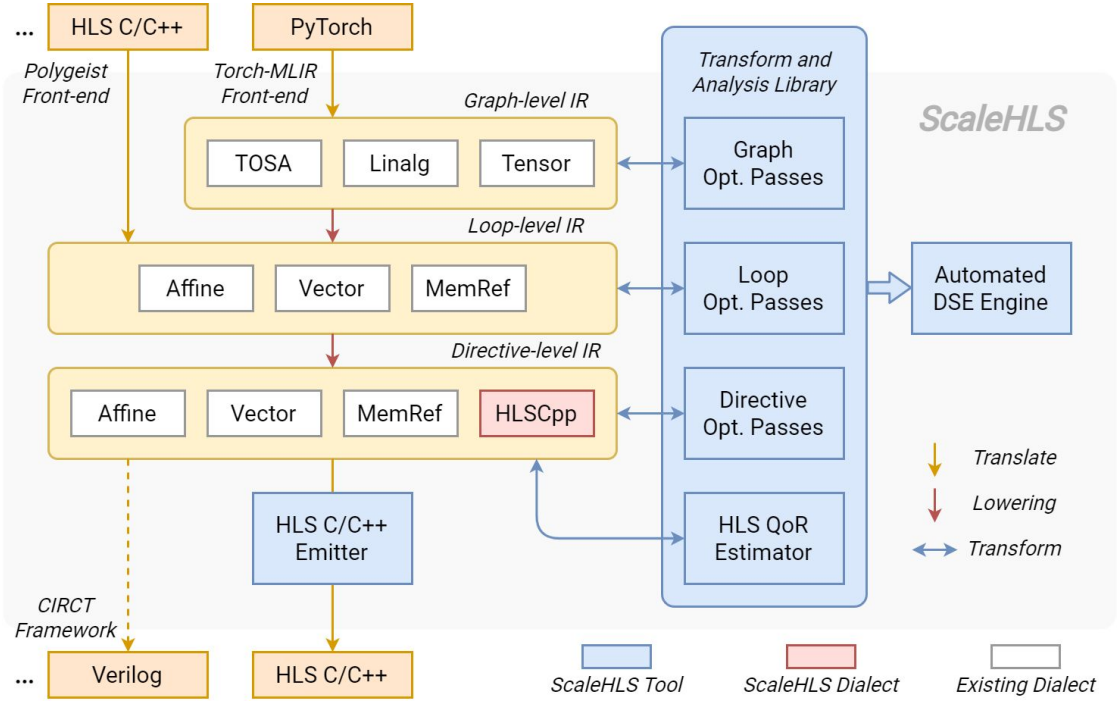
[1] J. Cong, et al. High-Level Synthesis for FPGAs: From Prototyping to Deployment. 2011. TCAD.
[2] B. C. Schafer, et al. High-Level Synthesis Design Space Exploration: Past, Present, and Future. 2020. TCAD.
[3] A. Sohrabizadeh, et al. AutoDSE: Enabling Software Programmers to Design Efficient FPGA Accelerators. 2022. TODAES.
[4] M. Yu. Chimera: A Hybrid Machine Learning-Driven Multi-Objective Design Space Exploration Tool for FPGA High-Level Synthesis. 2021. IDEAL.

Motivation: Marry HLS and MLIR



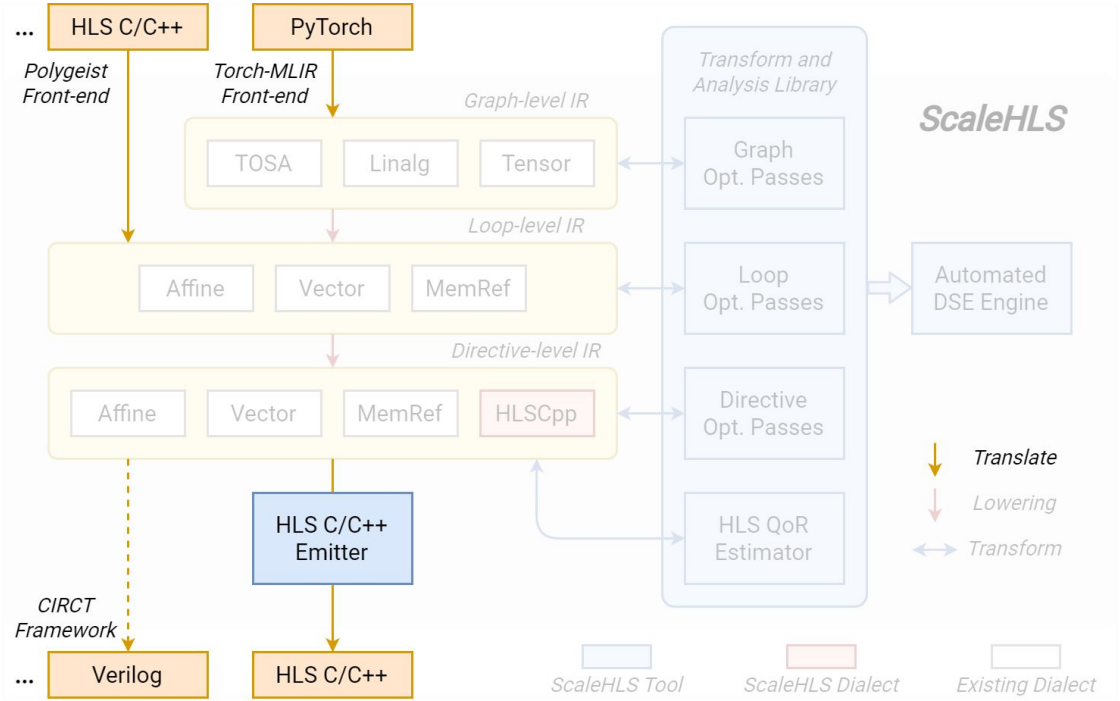
[1] C. Lattner, et al. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. 2021. CGO.

ScaleHLS Framework: Integration



[1] Polygeist: https://github.com/wsmoses/Polygeist [2] Torch-MLIR: https://github.com/llvm/torch-mlir [3] CIRCT: https://github.com/llvm/circt

ScaleHLS Framework: Integration (Cont'd)



Inputs

C/C++ Polygeist [1]

PyTorch Torch-MLIR [2]

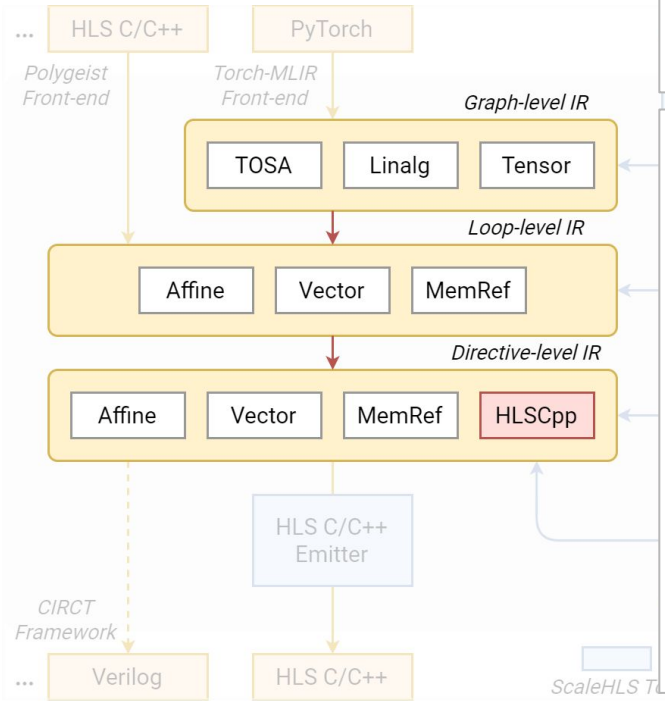
Outputs

C/C++ C/C++ Emitter

Verilog CIRCT [3] (work-in-progress)

[1] Polygeist: https://github.com/wsmoses/Polygeist [2] Torch-MLIR: https://github.com/llvm/torch-mlir [3] CIRCT: https://github.com/llvm/circt

ScaleHLS Framework: Representation



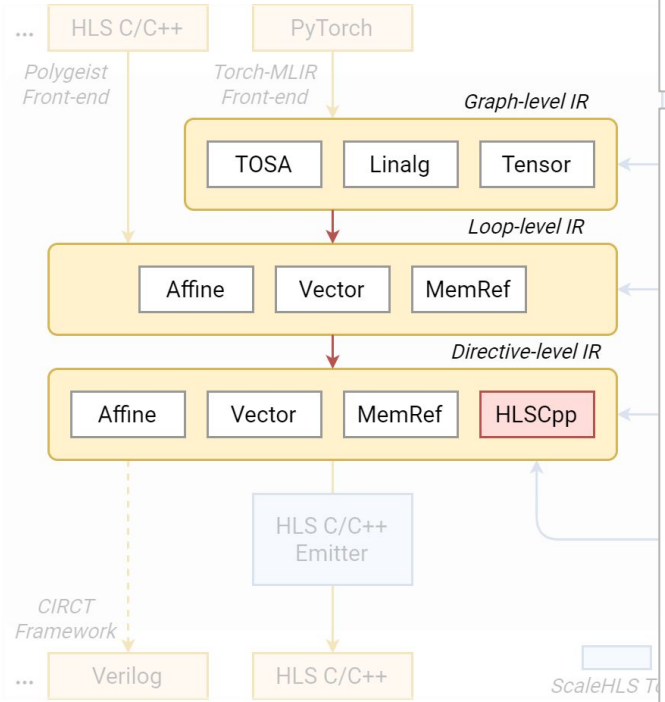

```
%O = "tosa.conv2d"(%I, %W, ...) {...} : (tensor<1x32x32x3xi8>, tensor<64x3x3x3xi8>, ...) ->
                                        tensor<1x30x30x64xi8>
```

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 (Cont'd)



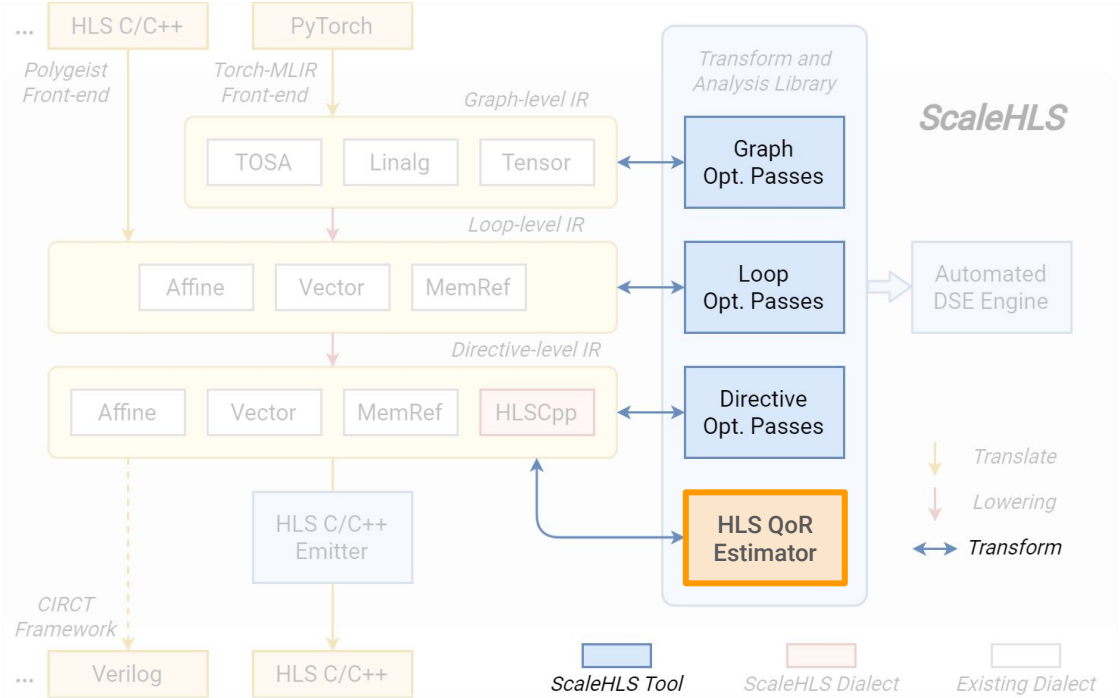

```
%O = "tosa.conv2d"(%I, %W, ...) {...} : (tensor<1x32x32x3xi8>, tensor<64x3x3x3xi8>, ...) ->
                                          tensor<1x30x30x64xi8>
```

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



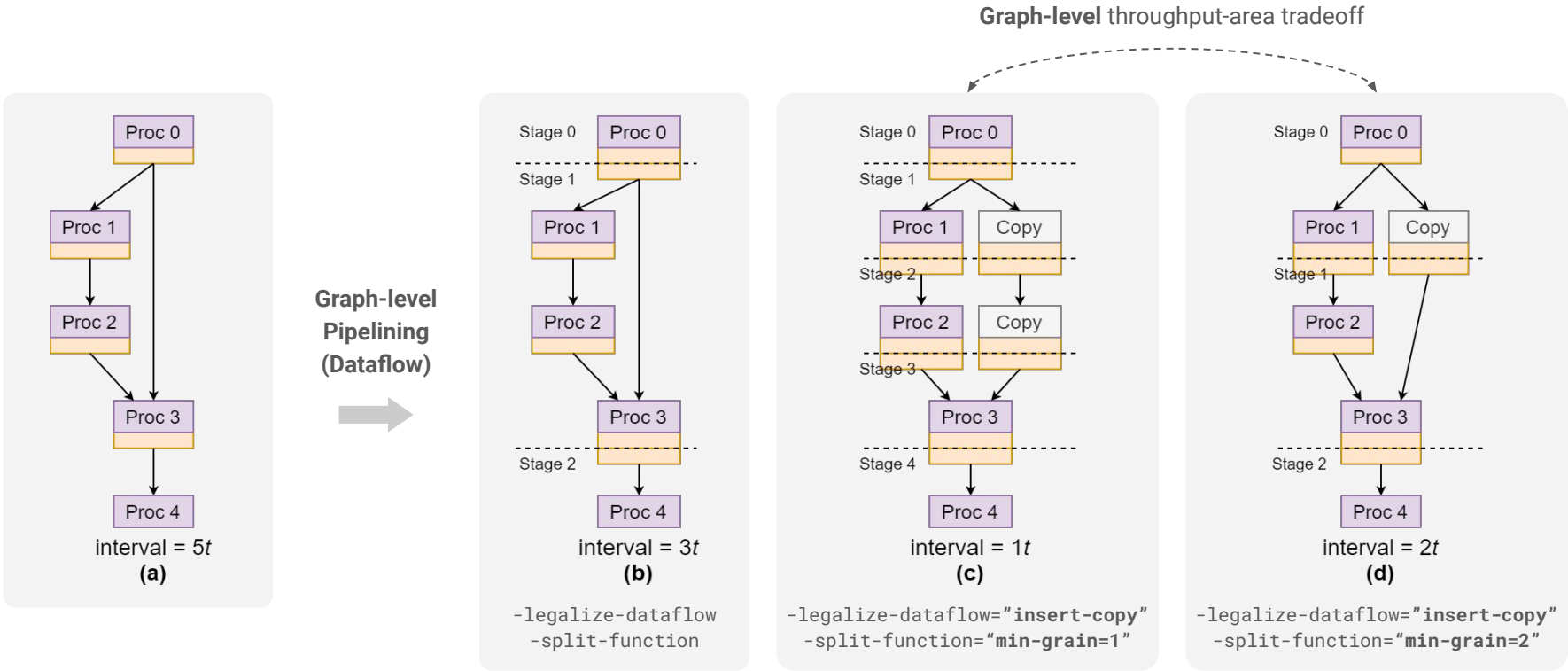
Level	ScaleHLS Passes

Graph	`-simplify-tosa-graph`
	`-legalize-dataflow`
	`-split-function`

Loop	`-affine-loop-perfectization`
	`-remove-variable-bound`
	`-affine-loop-tile`
	`-affine-loop-order-opt`
	`-affine-loop-unroll-jam`
	`-simplify-affine-if`
	`-affine-store-forward`
	`-simplify-memref-access`

Directive	`-loop-pipelining`
	`-function-pipelining`
	`-array-partition`
	`-create-hlscpp-primitive`
	`-qor-estimation`

ScaleHLS Framework: Graph Optimization



ScaleHLS Framework: Loop Optimization

```
void syrk(int alpha, int beta, int C[32][32], int A[32][32]) {
  for (int i = 0; i < 32; i++) {
    for (int j = 0; j <= i; j++) {
      C[i][j] *= beta;
      for (int k = 0; k < 32; k++) {
        C[i][j] += alpha * A[i][k] * A[j][k];
      }
    }
  }
}
```

Loop perfectization

Baseline C

Loop-level Opts in MLIR

Loop permutation

Loop unrolling

Remove variable bound

```
void syrk(int alpha, int beta, int C[32][32], int A[32][32]) {
  for (int k = 0; k < 32; k += 2) {
    for (int i = 0; i < 32; i += 1) {
      for (int j = 0; j < 32; j += 1) {
        if ((i - j) >= 0) {
          int v7 = C[i][j];
          int v8 = beta * v7;
          int v9 = A[i][k];
          int v10 = A[j][k];
          int v11 = (k == 0) ? v8 : v7;
          int v12 = alpha * v9;
          int v13 = v12 * v10;
          int v14 = v11 + v13;
          int v15 = A[i][(k + 1)];
          int v16 = A[j][(k + 1)];
          int v17 = alpha * v15;
          int v18 = v17 * v16;
          int v19 = v14 + v18;
          C[i][j] = v19;
        }
      }
    }
  }
}
```

ScaleHLS Emitted C

ScaleHLS Framework: Directive Optimization

```
void syrk(int alpha, int beta, int C[32][32], int A[32][32]) {
  for (int k = 0; k < 32; k += 2) {
    for (int i = 0; i < 32; i += 1) {
      for (int j = 0; j < 32; j += 1) {
        if ((i - j) >= 0) {
          int v7 = C[i][j];
          int v8 = beta * v7;
          int v9 = A[i][k];
          int v10 = A[j][k];
          int v11 = (k == 0) ? v8 : v7;
          int v12 = alpha * v9;
          int v13 = v12 * v10;
          int v14 = v11 + v13;
          int v15 = A[i][(k + 1)];
          int v16 = A[j][(k + 1)];
          int v17 = alpha * v15;
          int v18 = v17 * v16;
          int v19 = v14 + v18;
          C[i][j] = v19;
        }
      }
    }
  }
}
```

ScaleHLS Emitted C

Directive-level Opts in MLIR

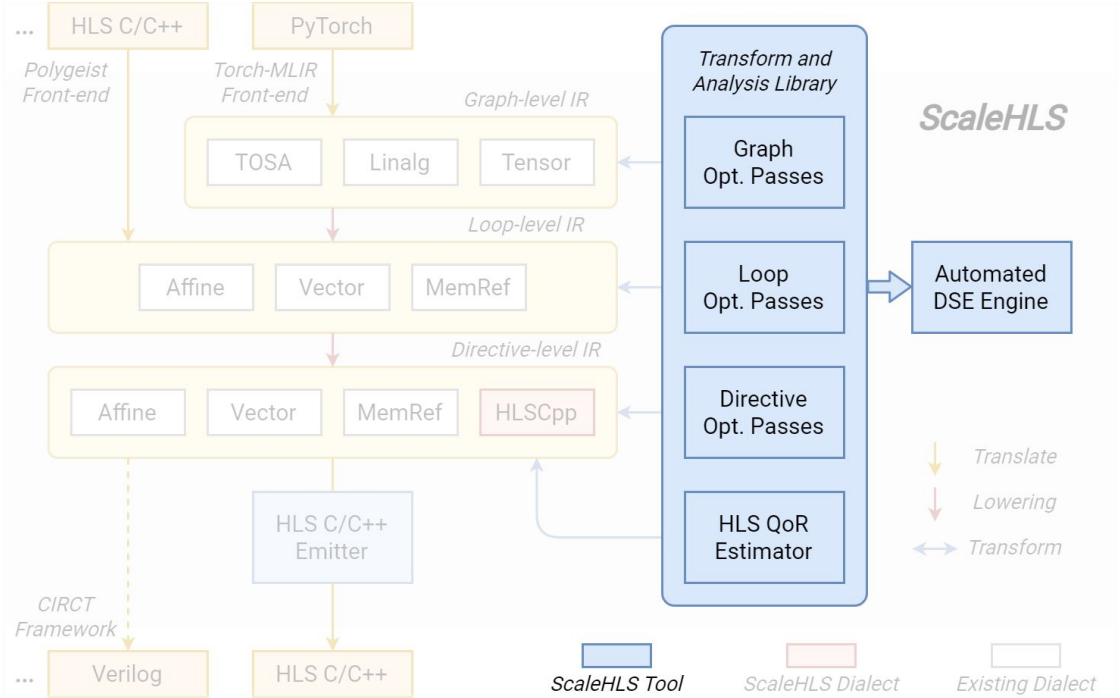

```
void syrk(int alpha, int beta, int C[32][32], int A[32][32]) {
#pragma HLS interface bram port=C
#pragma HLS interface bram port=A

#pragma HLS resource variable=C core=ram_s2p_bram
#pragma HLS resource variable=A core=ram_s2p_bram
#pragma HLS array_partition variable=A cyclic factor=2 dim=2

  for (int k = 0; k < 32; k += 2) {
    for (int i = 0; i < 32; i += 1) {
      for (int j = 0; j < 32; j += 1) {
#pragma HLS pipeline II = 3
        if ((i - j) >= 0) {
          int v7 = C[i][j];
          int v8 = beta * v7;
          int v9 = A[i][k];
          int v10 = A[j][k];
          int v11 = (k == 0) ? v8 : v7;
          int v12 = alpha * v9;
          int v13 = v12 * v10;
          int v14 = v11 + v13;
          int v15 = A[i][(k + 1)];
          int v16 = A[j][(k + 1)];
          int v17 = alpha * v15;
          int v18 = v17 * v16;
          int v19 = v14 + v18;
          C[i][j] = v19;
  } } } }
}
```

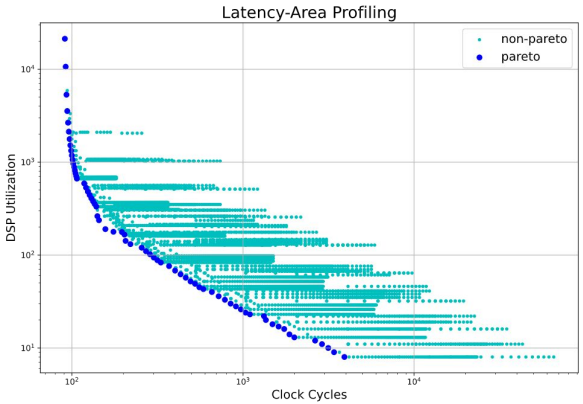
Array Partitioning

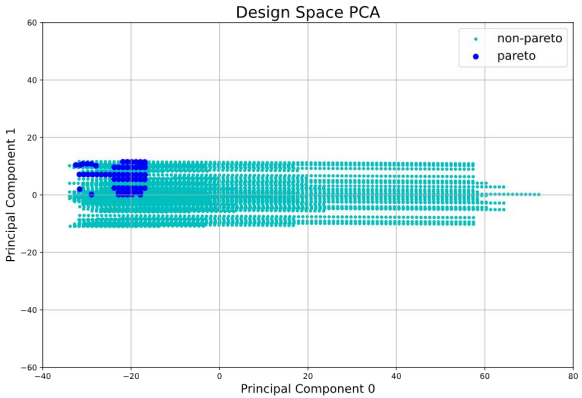
Loop Pipelining

ScaleHLS Emitted C

ScaleHLS Framework: Exploration





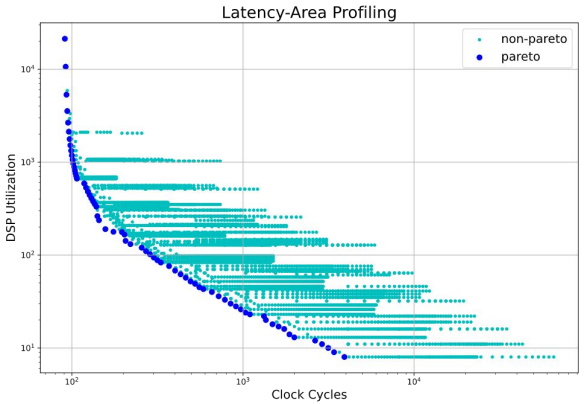


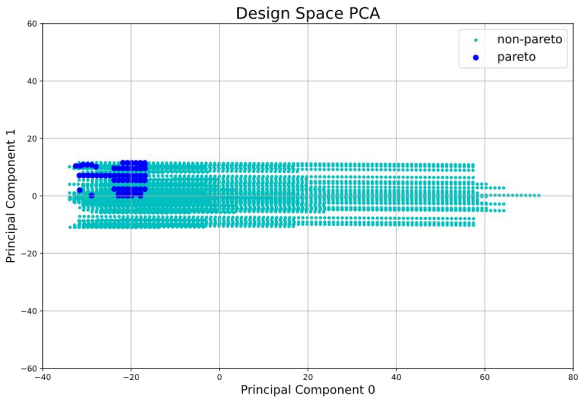
ScaleHLS Framework: Exploration (Cont'd)

Design Space Exploration (DSE) algorithm

1. Sample the whole design space and evaluate each sampled design point with the QoR estimator
2. Extract the Pareto frontier from all evaluated design points
3. Evaluate the closest neighbor of a randomly selected design point in the current Pareto frontier
4. Repeat step (2) and (3) to update the discovered Pareto frontier
5. Stop when no eligible neighbor can be found or meeting the early-termination criteria

With the **Transform and Analysis Library** of ScaleHLS, the exploration engine can be extended to support other optimization algorithms in the future.



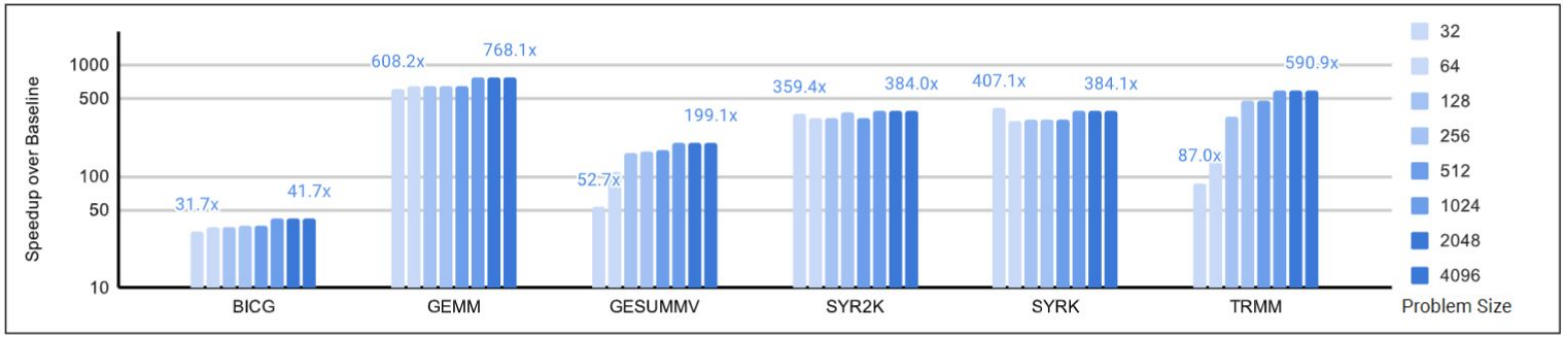


Evaluation of C/C++ Kernels: Design Space Exploration

Kernel	Prob. Size	Speedup	LP	RVB	Perm. Map	Tiling Sizes	Pipeline II	Array Partition Factors
BICG	4096	41.7×	No	No	[1, 0]	[16, 8]	43	A:[8, 16], s:[16], q:[8], p:[16], r:[8]
GEMM	4096	768.1×	Yes	No	[1, 2, 0]	[8, 1, 16]	3	C:[1, 16], A:[1, 8], B:[8, 16]
GESUMMV	4096	199.1×	Yes	No	[1, 0]	[8, 16]	9	A:[16, 8], B:[16, 8], tmp:[16], x:[8], y:[16]
SYR2K	4096	384.0×	Yes	Yes	[1, 2, 0]	[8, 4, 4]	8	C:[4, 4], A:[4, 8], B:[4, 8]
SYRK	4096	384.1×	Yes	Yes	[1, 2, 0]	[64, 1, 1]	3	C:[1, 1], A:[1, 64]
TRMM	4096	590.9×	Yes	Yes	[1, 2, 0]	[4, 4, 32]	13	A:[4, 4], B:[4, 32]

- The target platform is Xilinx XC7Z020 FPGA. Benchmarks are from PolyBench-C.
- 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 C/C++ Kernels: Scalability Study



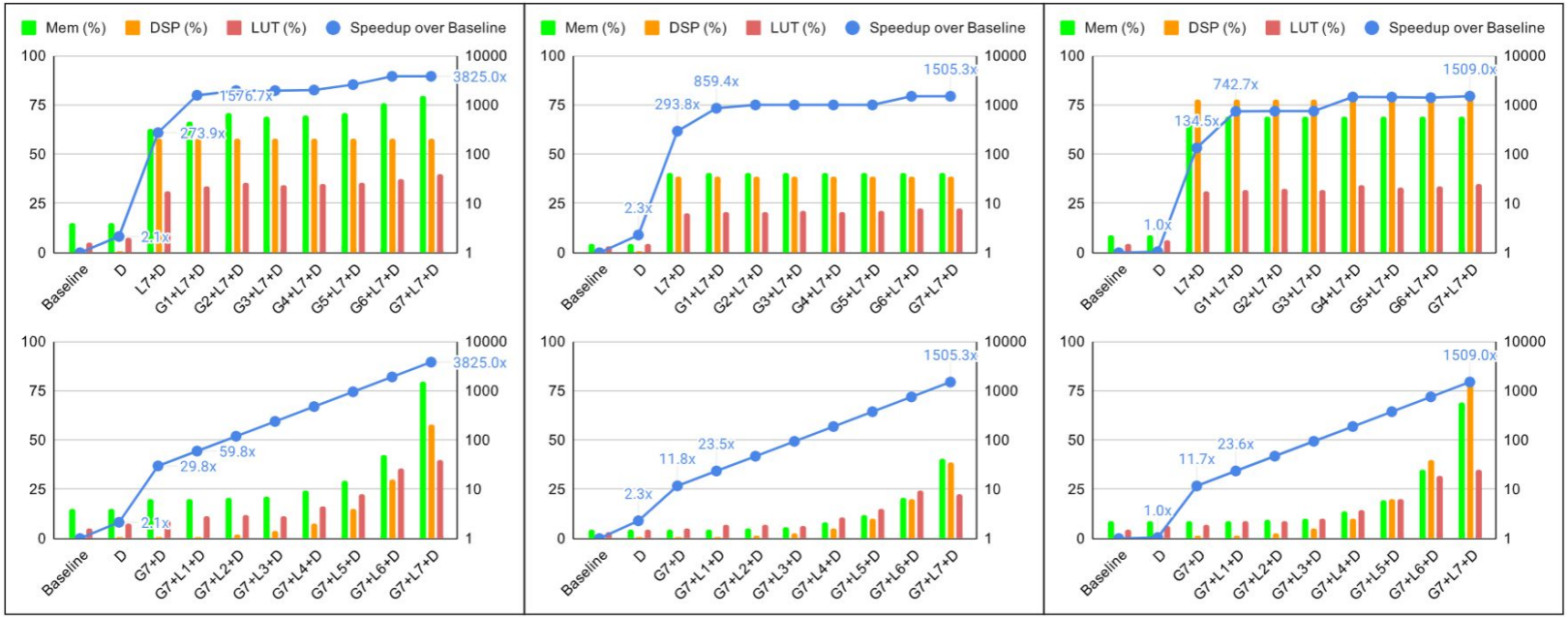
The problem sizes of PolyBench-C benchmarks are scaled from from 32 to 4096 and the DSE engine is used to search for the optimized solution under each setting.

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.

ScaleHLS is Open-Sourced!

ScaleHLS GitHub Repository

https://github.com/hanchenye/scalehls

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

Conclusion

1. We presented ScaleHLS, a new MLIR-based HLS compilation flow, which features multi-level representation and optimization of HLS designs and supports a transform and analysis library dedicated for HLS.
2. ScaleHLS enables an end-to-end compilation pipeline supporting both C/C++ and PyTorch as input.
3. An automated and extensible DSE engine is developed to search for optimal solutions in the multi-dimensional design spaces.
4. Experimental results demonstrate that ScaleHLS has a strong scalability to optimize large-scale and sophisticated HLS designs and achieves significant performance and productivity improvements on a set of benchmarks.

Acknowledgement

We thank Eric Cheng of Laboratory for Physical Sciences (LPS) and Samuel Bayliss of Xilinx for insightful discussions. This work is supported in part by Xilinx Center of Excellence at UIUC, Xilinx Adaptive Compute Cluster (XACC) initiative, and BAH HT 15-1158 contract.

Questions?

Email: hanchen8@illinois.edu

Website: hanchenye.com