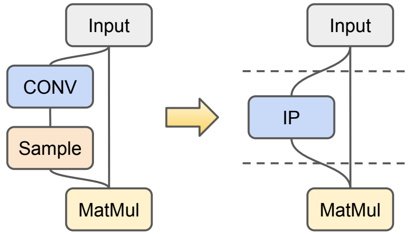
ScaleHLS: A New Scalable High-Level Synthesis Framework on Multi-Level Intermediate Representation

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.

Motivation: Marry HLS and MLIR



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

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

```
for (int k = 0; k < 32; k++) {
  for (int i = 0; i < 32; i++) {
    for (int j = 0; j < 32; j++) {
#pragma HLS pipeline
      if (k == 0)
        C[i][j] *= beta;
      C[i][j] += alpha * A[i][k] * B[k][j];
} } }
```

Graph Optimizations

Node fusion
IP integration
Task-level pipeline, etc.

Loop Optimizations

Loop interchange
Loop perfectization
Loop tile, skew, etc.

Directive Optimizations

Loop pipeline, unroll
Function pipeline, inline
Array partition, etc.

Multiple Abstraction Levels

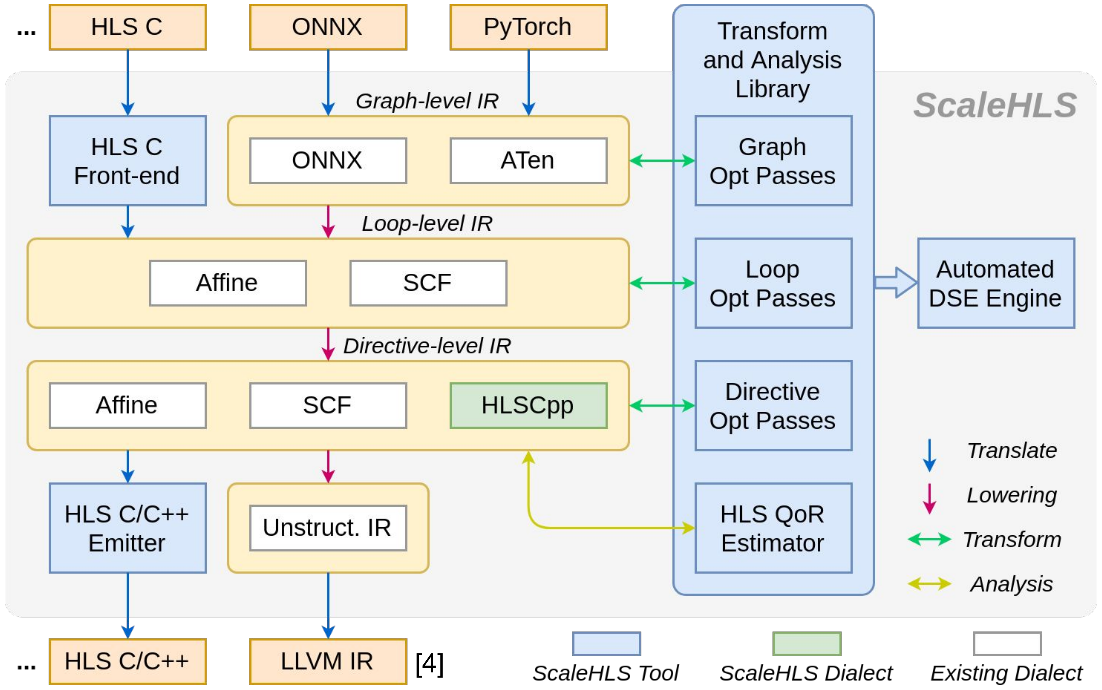
HLS designs come with intrinsic hierarchy

HLS optimizations are better solved at multiple different abstraction levels

MLIR

A compilation infrastructure with Multi-Level Intermediate Representation

ScaleHLS Framework



[1] ONNX-MLIR: Compiling ONNX Neural Network Models Using MLIR. https://github.com/onnx/onnx-mlir
[2] NPComp: MLIR based compiler toolkit for numerical python programs. https://github.com/llvm/mlir-npcomp
[3] MLIR: Multi-Level Intermediate Representation. https://github.com/llvm/llvm-project/tree/main/mlir
[4] Vitis HLS Front-end: https://github.com/Xilinx/HLS

Represent It!

Graph-level IR: ONNX [1] and ATen [2] dialect.

Loop-level IR: Affine [3] and SCF (structured control flow) [3] dialect. Can leverage the transformation and analysis libraries applicable in MLIR.

Directive-level IR: HLSCpp, Affine, and SCF dialect.

Optimize It!

Optimization Passes: Cover the graph, loop, and directive levels. Solve optimization problems at the 'correct' abstraction level. 💪

QoR Estimator: Estimate the latency and resource utilization through IR analysis.

Explore It!

Transform and Analysis Library: Parameterized interfaces of all optimization passes and the QoR estimator. A playground of DSE. 🚀

Automated DSE Engine: Find the Pareto-frontier of the throughput-area trade-off design space.

Enable End-to-end Flow!

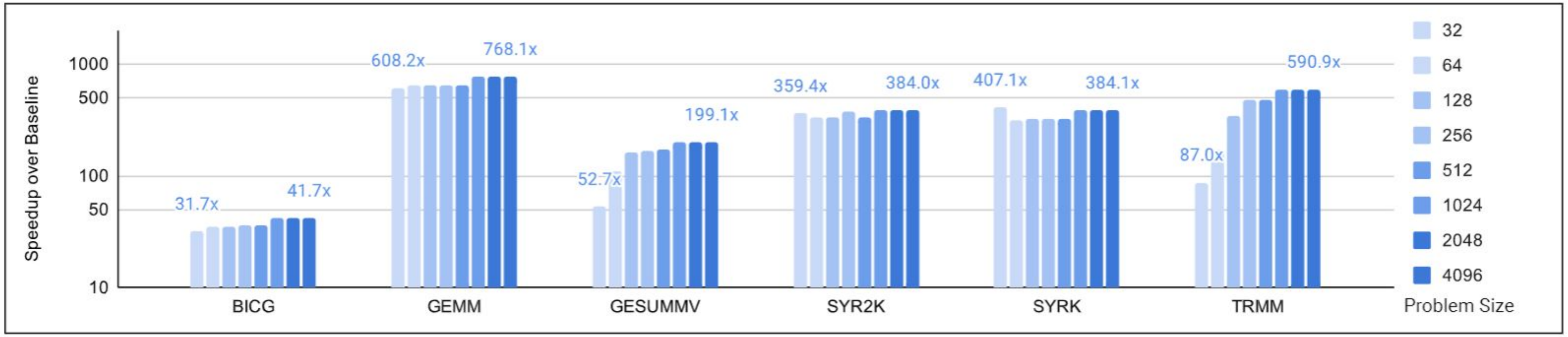
HLS C Front-end: Parse C programs into MLIR.

HLS C/C++ Emitter: Generate synthesizable HLS designs for downstream tools, such as Vivado HLS.

DSE Results of Computation Kernel

Kernel	Prob. Size	Speedup	LP	RVB	Perm. Map	Tiling Sizes	Pipeline II	Array Partition
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]

Evaluation results on Xilinx XC7Z020 FPGA. Speedup is with respect to the baseline designs only optimized by Xilinx Vivado HLS. LP and RVB denote Loop Perfectization and Remove Variable Bound, respectively. In the Loop Order Optimization,the i-th loop in the loop nest is permuted to location $PermMap[i]$, where locations are from the outermost loop to inner.

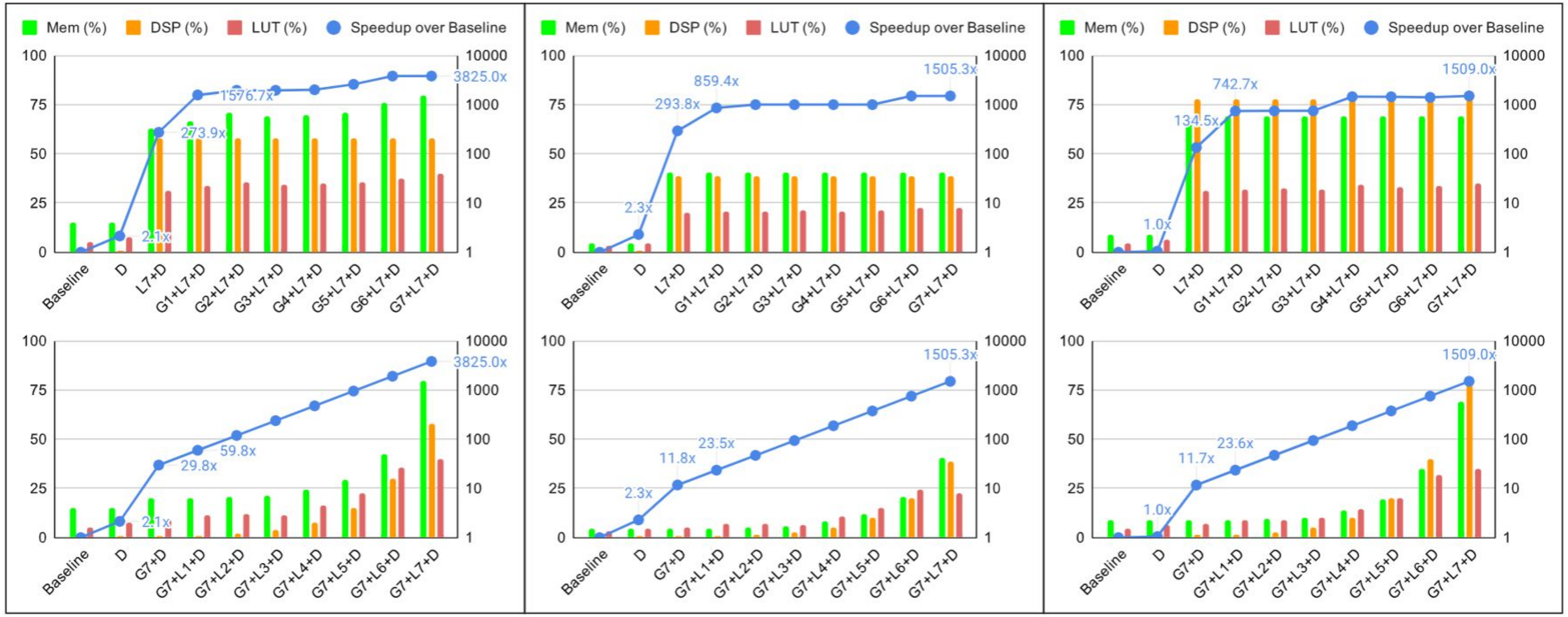


The problem sizes of computation kernels are scaled from 32 to 4096 and the DSE engine is launched to search for the optimal solutions under each problem size.

Optimization Results of DNN Models

Model	Speedup	Runtime (seconds)	Memory (SLR Util. %)	DSP (SLR Util. %)	LUT (SLR Util. %)	FF (SLR Util. %)	Our DSP Effi. (OPs/Cycle/DSP)	DSP Effi. of TVM-VTA [26]
ResNet-18	3825.0×	60.8	91.7Mb (79.5%)	1326 (58.2%)	157902 (40.1%)	54766 (6.9%)	1.343	0.344
VGG-16	1505.3×	37.3	46.7Mb (40.5%)	878 (38.5%)	88108 (22.4%)	31358 (4.0%)	0.744	0.296
MobileNet	1509.0×	38.1	79.4Mb (68.9%)	1774 (77.8%)	138060 (35.0%)	56680 (7.2%)	0.791	0.468

Evaluation results on Xilinx VU9P FPGA with graph, loop, and directive optimizations. *Speedup* is with respect to the baseline designs compiled from PyTorch to HLS C++ by ScaleHLS but without the multi-level optimization.



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

Ablation study of DNN models. 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!

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

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

HLS Researchers

1. Rapidly implement and evaluate new HLS optimization algorithms on top of our multi-level IR
2. Investigate new design space exploration algorithms on top of our transform and analysis library
3. Rapidly build end-to-end HLS optimization flow and demonstrate their wonderful work!

HLS Users

1. Optimize their HLS designs at multiple abstraction levels using our optimization passes in command line tool
2. Avoid premature design choices by using our QoR estimator to estimate the latency and utilization
3. Find optimized HLS designs with our automated DSE

Represent It!

Graph-level IR: ONNX [1] and ATen [2] dialect.

Loop-level IR: Affine [3] and SCF (structured control flow) [3] dialect. Can leverage the transformation and analysis libraries applicable in MLIR.

Directive-level IR: HLSCpp, Affine, and SCF dialect.

Optimize It!

Optimization Passes: Cover the graph, loop, and directive levels. Solve optimization problems at the 'correct' abstraction level. 💪

QoR Estimator: Estimate the latency and resource utilization through IR analysis.

Explore It!

Transform and Analysis Library: Parameterized interfaces of all optimization passes and the QoR estimator. A playground of DSE. 🚀

Automated DSE Engine: Find the Pareto-frontier of the throughput-area trade-off design space.

Enable End-to-end Flow!

HLS C Front-end: Parse C programs into MLIR.

HLS C/C++ Emitter: Generate synthesizable HLS designs for downstream tools, such as Vivado HLS.