UCLA COMPUTER SCIENCE DEPARTMENT

Center for Domain Specific Computing

AutoDSE: Enabling Software Programmers to Design Efficient FPGA Accelerators

Atefeh Sohrabizadeh, Cody Hao Yu, Min Gao, Jason Cong

ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 27, No. 4, Article 32. Pub. date: January 2022.

FPGA'22 workshop – 2/27/2022

VLSI architecture, synthesis & technology

Introduction

Our Focus

- High-level synthesis optimization steps
  - 1) Loop transformations
    - e.g., for increasing data reuse and/or removing data dependency

      → Not specific to FPGAs
  - 2) Code transformations for general repetitive architectural optimizations
    - e.g., memory coalescing and/or memory burst

      → The Merlin Compiler can automatically apply them
  - 3) FPGA-specific architectural optimizations
    - e.g., deciding where to pipeline the computation, how many processing elements to use, etc.

      → Requires hardware knowledge

Target Example: One forward path of CNN

```
void CnnKernel(
    const float input[kNumIn][kImSizeIn][kImSizeIn],
    const float weight[kNumOut][kNumIn][kKernel][kKernel],
    const float bias[kNumOut],
    float output[kNumOut][kOutImSize][kOutImSize]) {

  float C[parOut][kImSize][kImSize];

// Initialization
  for (int i = 0; i < kNumOut / parOut; i++) {
    for (int h = 0; h < kImSize; ++h) {
      for (int w = 0; w < kImSize; ++w) {
        for (int po = 0; po < parOut; po++) {
          C[po][h][w] = 0.f; } } }
// Convolution
    conv: for (int j = 0; j < kNumIn; ++j) {
      loop_h: for (int h = 0; h < kImSize; ++h) {
        loop_w: for (int w = 0; w < kImSize; ++w) {
          loop_po: for (int po = 0; po < parOut; po++) {
            float tmp = (float )0;
            loop_p: for (int p = 0; p < kKernel; ++p) {
              Loop_q: for (int q = 0; q < kKernel; ++q) {
                tmp += weight[(i << shift) + po][j][p][q] *
input[j][h + p][w + q]; } }
            C[po][h][w] += tmp; } } } }
// Max pooling
    pooling:
    for (int h = 0; h < kOutImSize; ++h) {
      for (int w = 0; w < kOutImSize; ++w) {
        for (int po = 0; po < parOut; po++) {
          output[(i << shift) + po][h][w] = … } } } } }
```

- 80x slower than single core CPU!!
- Not every C program gives a good performance
- Solution:
  - Identify where to apply optimization pragmas to:
    - Create load/compute/store functions and double buffering
    - Create parallel coarse-grained processing elements by wrapping the inner loops as a function and setting proper array partition factors
    - Create parallel/pipelined units with proper partitioning of the arrays

Optimizing the Code Using Merlin Compiler

- Can exploit Merlin [1] pragmas to get to > 7,000x speedup without any code transformation
  - With only 4 pragmas! ☺

```
// Skip function header due to page limit
// Initialization
  for (int i = 0; i < kNumOut / parOut; i++) {
    for (int h = 0; h < kImSize; ++h) {
#pragma ACCEL PARALLEL FACTOR=4
      for (int w = 0; w < kImSize; ++w) {
        for (int po = 0; po < parOut; po++) {
          C[po][h][w] = 0.f;
     } } }
// Convolution
#pragma ACCEL PIPELINE
    for (int j = 0; j < kNumIn; ++j) {
      for (int h = 0; h < kImSize; ++h) {
#pragma ACCEL PARALLEL FACTOR=4
#pragma ACCEL PIPELINE flatten
        for (int w = 0; w < kImSize; ++w) {
          for (int po = 0; po < parOut; po++) {
            float tmp = (float )0;
            for (int p = 0; p < kKernel; ++p) {
              for (int q = 0; q < kKernel; ++q) {
                tmp += weight[(i << shift) + po][j][p][q] * input[j][h + p][w + q]; } }
            C[po][h][w] += tmp; } } } }
// Max pooling
    pooling:
    for (int h = 0; h < kOutImSize; ++h) {
      for (int w = 0; w < kOutImSize; ++w) {
        for (int po = 0; po < parOut; po++) {
          output[(i << shift) + po][h][w] = …
} } } } }
```


[1] https://github.com/Xilinx/merlin-compiler

Candidate Pragmas of the CNN Example

- However,
  - 26 candidate pragmas to search!
  - $> 10^{16}$ possible solutions!!

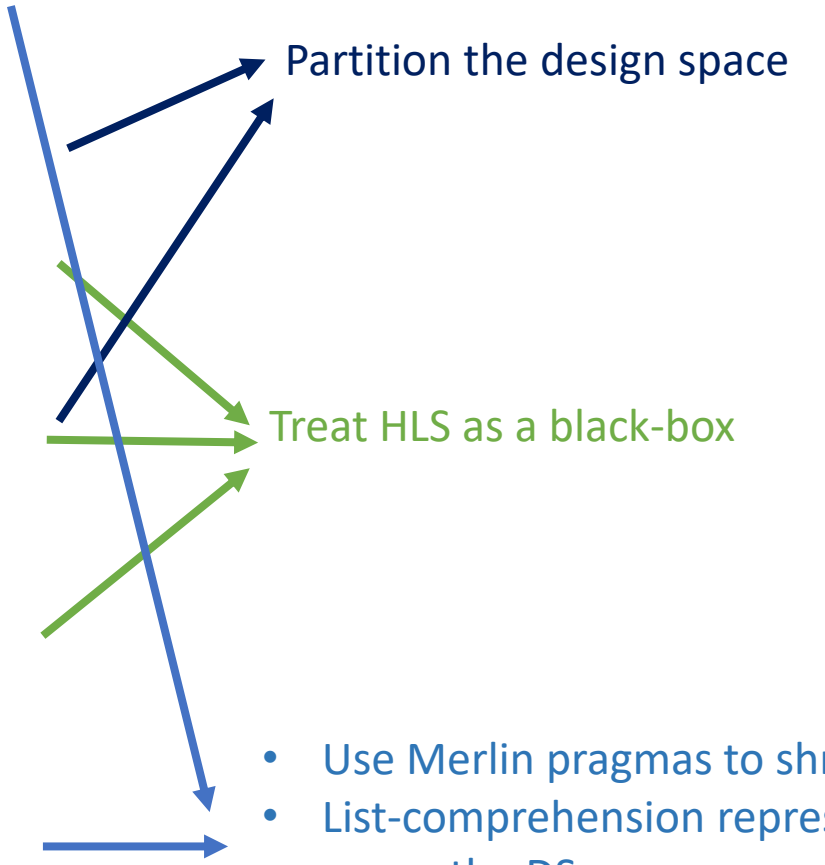
```
// Skip function header due to page limit
// Initialization
#pragma ACCEL PIPELINE auto{__PIPE__L0}
#pragma ACCEL TILE FACTOR=auto{__TILE__L0}
#pragma ACCEL PARALLEL FACTOR=auto{__PARA__L0}
  for (int i = 0; i < kNumOut / parOut; i++) {
#pragma ACCEL PIPELINE auto{__PIPE__L1}
#pragma ACCEL TILE FACTOR=auto{__TILE__L1}
#pragma ACCEL PARALLEL FACTOR=auto{__PARA__L1}
    for (int h = 0; h < kImSize; ++h) {
#pragma ACCEL PIPELINE auto{__PIPE__L4}
#pragma ACCEL TILE FACTOR=auto{__TILE__L4}
#pragma ACCEL PARALLEL FACTOR=auto{__PARA__L4}
      for (int w = 0; w < kImSize; ++w) {
        for (int po = 0; po < parOut; po++) {
          C[po][h][w] = 0.f;
    } } }

// Convolution
#pragma ACCEL PIPELINE auto{__PIPE__L2}
#pragma ACCEL TILE FACTOR=auto{__TILE__L2}
#pragma ACCEL PARALLEL FACTOR=auto{__PARA__L2}
    for (int j = 0; j < kNumIn; ++j) {
#pragma ACCEL PIPELINE auto{__PIPE__L5}
#pragma ACCEL TILE FACTOR=auto{__TILE__L5}
#pragma ACCEL PARALLEL FACTOR=auto{__PARA__L5}
      for (int h = 0; h < kImSize; ++h) {
```

```
#pragma ACCEL PIPELINE auto{__PIPE__L8}
#pragma ACCEL TILE FACTOR=auto{__TILE__L8}
#pragma ACCEL PARALLEL FACTOR=auto{__PARA__L8}
        for (int w = 0; w < kImSize; ++w) {
#pragma ACCEL PIPELINE auto{__PIPE__L10}
#pragma ACCEL TILE FACTOR=auto{__TILE__L10}
          for (int po = 0; po < parOut; po++) {
            float tmp = (float )0;
            for (int p = 0; p < kKernel; ++p) {
              for (int q = 0; q < kKernel; ++q) {
                tmp += weight[(i << shift) +
po][j][p][q] * input[j][h + p][w + q]; } }
            C[po][h][w] += tmp; } } } }
// Max pooling
#pragma ACCEL PIPELINE auto{__PIPE__L3}
#pragma ACCEL TILE FACTOR=auto{__TILE__L3}
#pragma ACCEL PARALLEL FACTOR=auto{__PARA__L3}
    for (int h = 0; h < kOutImSize; ++h) {
#pragma ACCEL PIPELINE auto{__PIPE__L6}
#pragma ACCEL TILE FACTOR=auto{__TILE__L6}
#pragma ACCEL PARALLEL FACTOR=auto{__PARA__L6}
      for (int w = 0; w < kOutImSize; ++w) {
        for (int po = 0; po < parOut; po++) {
          output[(i << shift) + po][h][w] = …
} } } } }
```

Our Solution

- **Challenge 1:** Large solution space
- **Challenge 2:** Non-monotonic effect of design parameters on performance/area
- **Challenge 3:** Correlation of different characteristics of a design
- **Challenge 4:** Implementation disparity of HLS tools
- **Challenge 5:** Long synthesis time of HLS tools



- Use Merlin pragmas to shrink the DS
- List-comprehension representation to prune the DS
- Use bottleneck analyzer to systematically find better design points

AutoDSE Methodology

The Pragmas We Search For

- Built on top of the Merlin Compiler
  - Benefits:
    - Source-to-Source code transformation
    - Small set of pragmas

Keyword	Available Options	Architecture Structure
parallel	factor=<int>	CG & FG parallelism
tile	factor=<int>	Loop tiling
pipeline	mode=cg/fg	CG or FG pipeline

CG: Coarse-Grained, FG: Fine-Grained

    - Focus on high-level architectural changes

- Target HLS pragmas

Optimization Pragmas	Non-Optimization Pragmas
DATAFLOW	INTERFACE
STREAM	LOOP_TRIPCOUNT
PIPELINE	
UNROLL	
ARRAY_PARTITION	
DEPENDENCE	
LOOP_FLATTEN	
INLINE	

AutoDSE Overview

- Objectives can be
  - Performance
  - Area
  - Trade-off of the two
    - Assess the quality of design with respect to a reference point using:

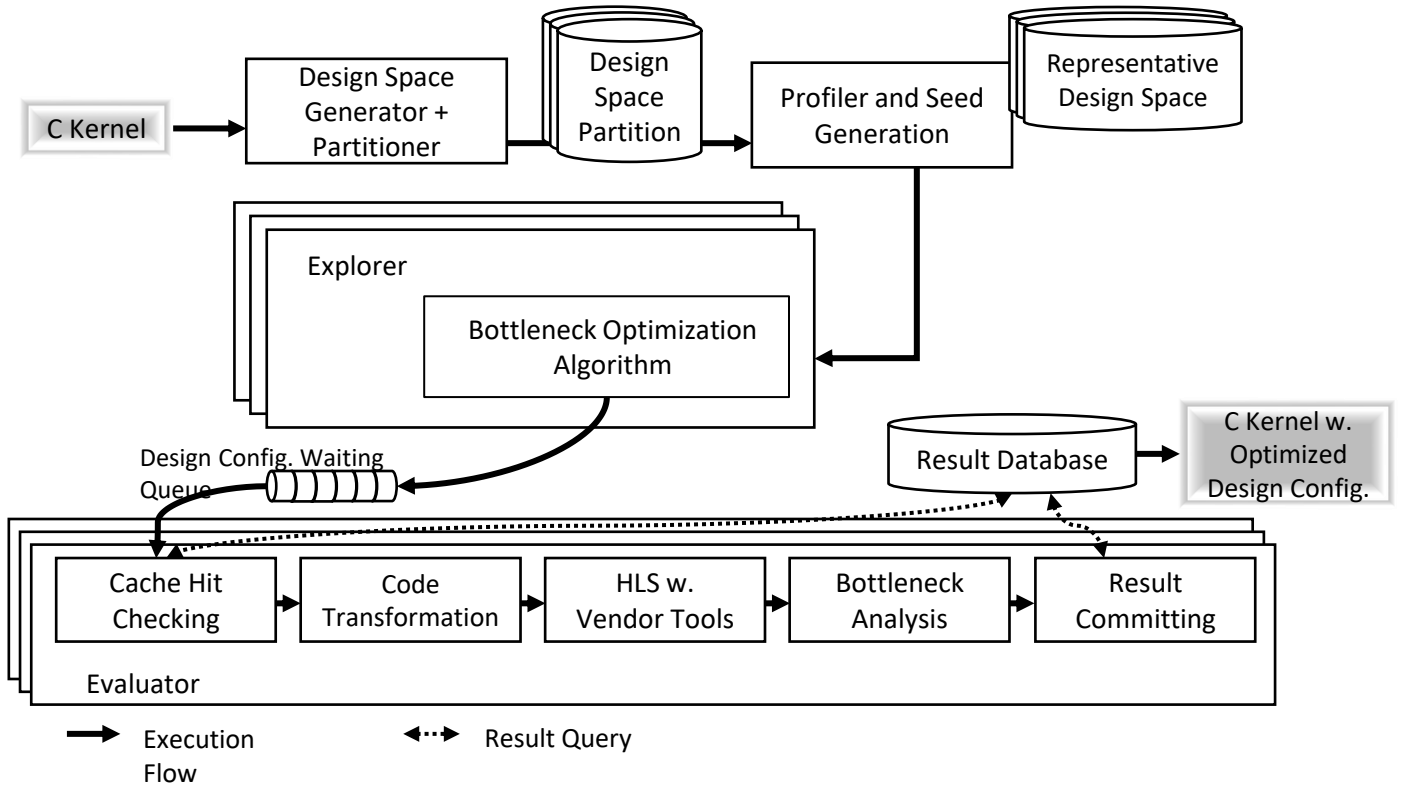
$$g(\theta_j, \theta_i) \sim \frac{Cycle(H, \mathcal{P}(\theta_j)) - Cycle(H, \mathcal{P}(\theta_i))}{Util(H, \mathcal{P}(\theta_j)) - Util(H, \mathcal{P}(\theta_i))}$$

$$\theta_{i+1} = \underset{\theta_j \in \theta_{cand}}{\operatorname{argmin}}\ g(\theta_j, \theta_i)$$

    - Benefit:

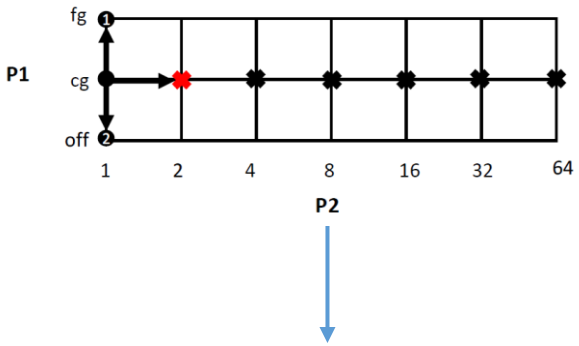
Design Point (θ_i)	Area compared to θ_0	Latency compared to θ_0	g(θ_i, θ_0)
θ_1	+30%	-10%	-0.33
θ_2	+10%	-5%	-0.5

Has more potential to get to better design since has more resources left



Peeking Into AutoDSE Framework

- List-based design space representation to prune the design space while keeping it smooth
  - Some pragmas are mutually exclusive
  - Make a lattice of the points and invalidate the conflicting points




```
// Skip the rest due to page limit
#pragma ACCEL PIPELINE auto{
  options: P1 = [x for x in [off, cg, fg]];
  default: 'off' }
#pragma ACCEL PARALLEL factor=auto{
  options: P2 = [x for x in [1, 2, 4, 8, 16, 32, 64] if P1!=cg];
  default: 1 }
for (int j = 0; j < NumIn; ++i) {
// Skip the rest due to page limit
```

- Parameter prioritization
  - An expert
    - Can analyze the cycle breakdown and find the bottleneck
    - Knows which parameter may be the killer parameter
  - Proposed method:
    - Mimic an expert's approach using a bottleneck analyzer

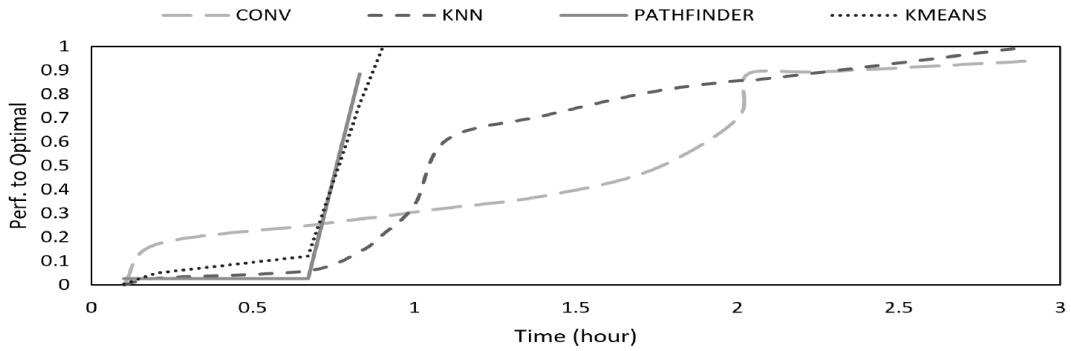
Hierarchy	TC	AC	CPC
kernel_mvt (mvt.c:4)		137701 (100.0%)	137701
auto memory burst for 'x1'(read)		120 (0.1%)	120
auto memory burst for 'A'(read)		7200 (5.2%)	7200
auto memory burst for 'y_1'(read)		120 (0.1%)	120
loop i (mvt.c:15)	120	727 (0.5%)	727
loop j (mvt.c:18)	120	-	-
auto memory burst for 'y_2'(read)		5 (0.0%)	5
auto memory burst for 'x1'(write)		120 (0.1%)	120
auto memory burst for 'A'(read)		246 (0.2%)	246
auto memory burst for 'x2'(read)		12 (0.0%)	12
loop i (mvt.c:28)	120	127200 (92.4%)	127200
loop j (mvt.c:31)	120	-	-
auto memory burst for 'x2'(write)		12 (0.0%)	12

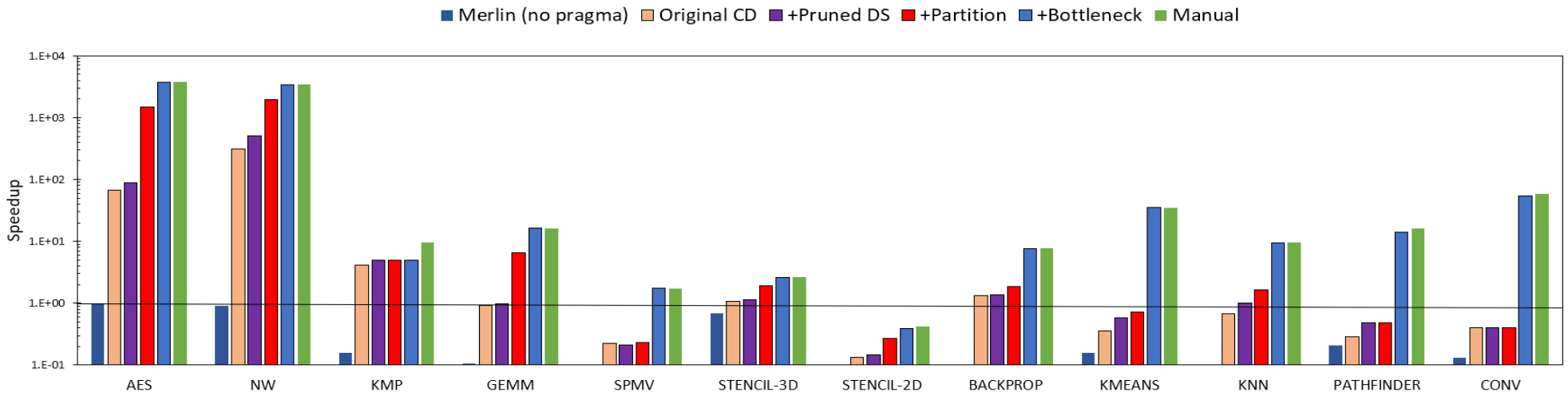
bottleneck

- Design space partitioning
  - Recap:
    - Performance gain is not smooth
    - There will be locally optimum points
  - Proposed solution
    - Partition based on the *pipeline mode* of a loop

Evaluation (1/2)

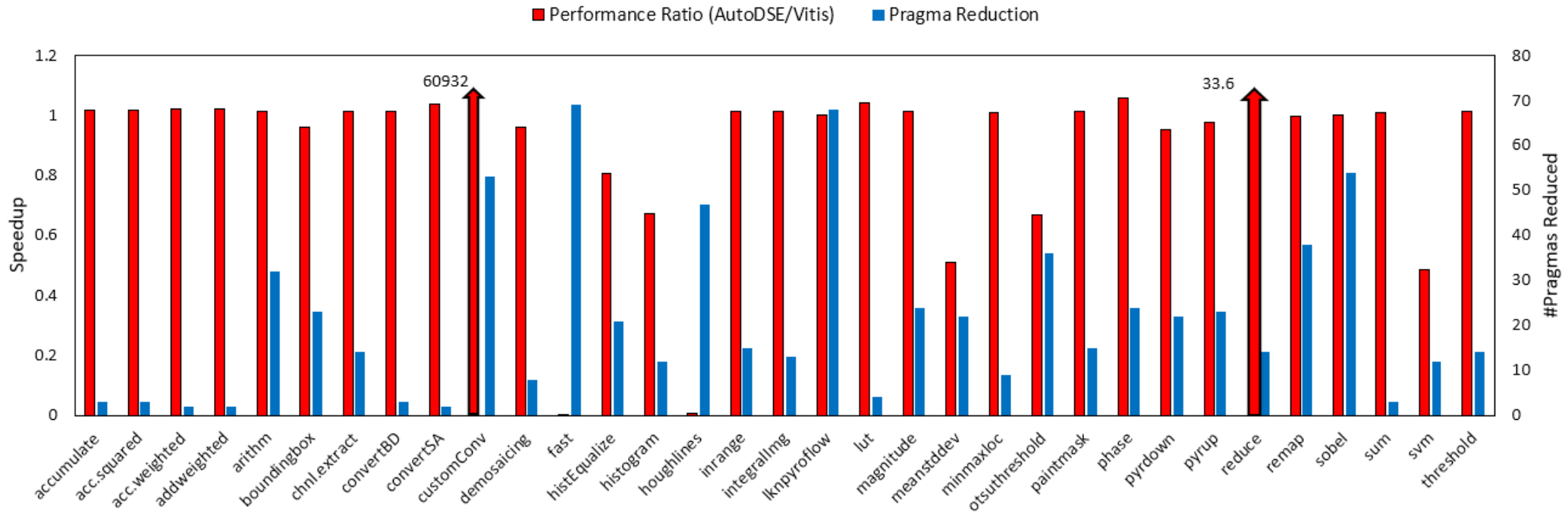
- Experimental Setup
  - Target FPGA: Xilinx Virtex UltraScale+TM VU9P FPGA
- Evaluation on MachSuite and Rodinia Benchmark
  - Speedups (on geometric mean)
    - 19.9x with respect to single core CPU
    - 3.6x with respect to S2FA [Yu, et al. DAC'18] and
      4.3x compared to lattice-based DSE [Ferretti, et al. ICCD'18]
      17.9x over Bayesian Optimization [Sun, et al. DATE'21]
    - 182.9x with respect to Merlin without pragmas
    - 0.93x compared to manual desgins
  - 1.1 hours on the geometric mean to find the best design





Evaluation (2/2)

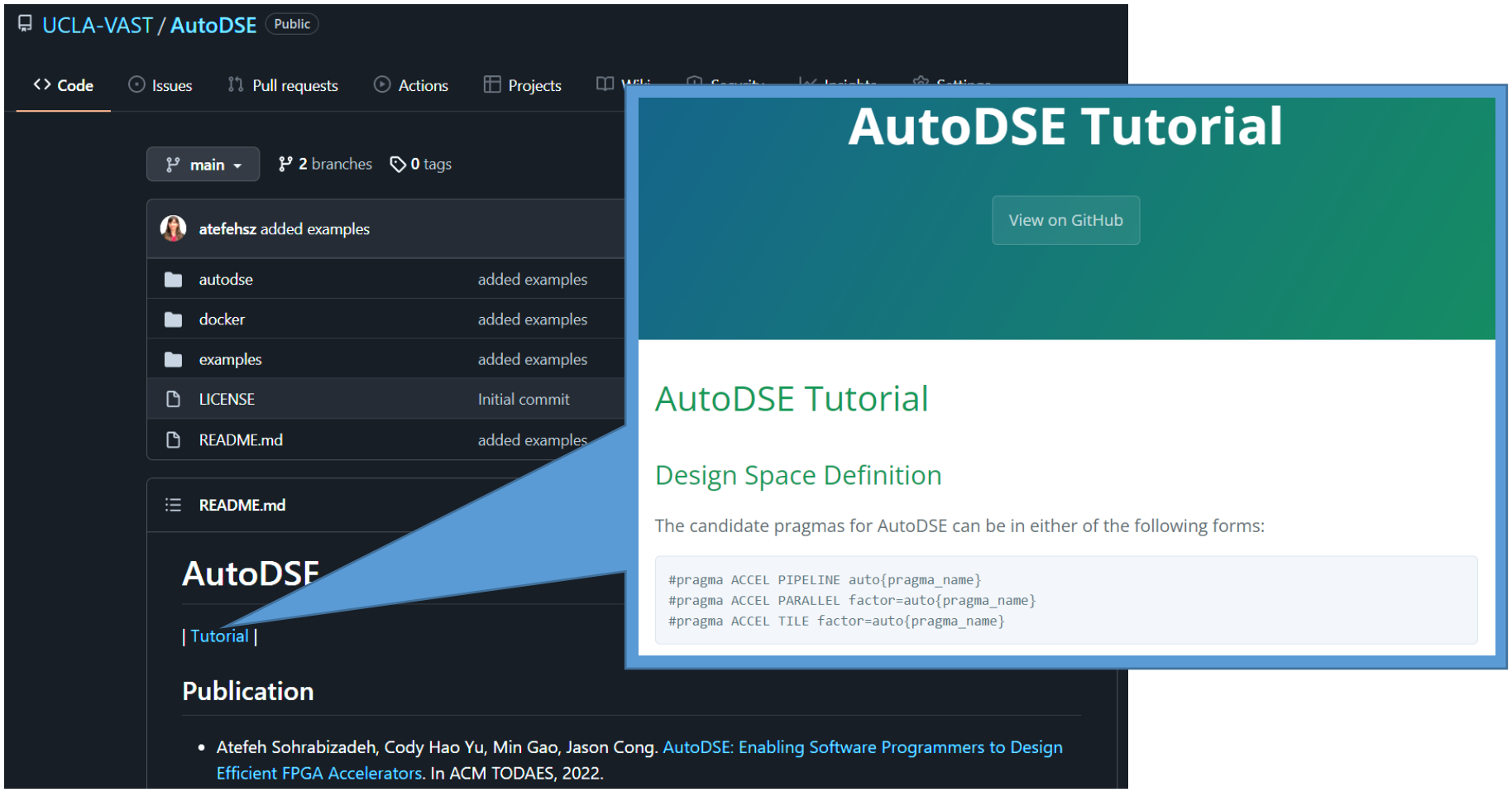
- Evaluation on Xilinx Vitis library
  - Tested on 33 of vision kernels
  - Removed the following optimization pragmas
    - UNROLL, PIPELINE, ARRAY_PARTITION, DEPENDENCE, LOOP_FLATTEN, and INLINE
    - Are used 13.5 times on geometric mean
  - Achieved 1.04x the performance of Xilinx in 0.3 hours on geometric mean



How to use AutoDSE?

AutoDSE's repo

- https://github.com/UCLA-VAST/AutoDSE



Scan me!

Pragma Types and How to Define Them

- 3 types of pragmas:

```
#pragma ACCEL PIPELINE auto{__PIPE__L1}
#pragma ACCEL TILE FACTOR=auto{__TILE__L1}
#pragma ACCEL PARALLEL FACTOR=auto{__PARA__L1}
```

- Each one should be defined in the following form:

```
"__PARA__L1" :
        {
            "ds_type" : "PARALLEL",
            "options" : "[x for x in [1,2,4,7,8,14,16,32] if x*__TILE__L1<=32]",
            "default" : 1,
            "order" : "0 if x&(x-1)==0 else 1"
        }
```

AutoDSE Run Configurations

- Apart from the pragma descriptions, [DS_INFO].json should set the settings to run AutoDSE:

```
    "design-space.max-part-num" : 4,
    "evaluate.command.bitgen" : "make mcc_bitgen",
    "evaluate.command.hls" : "make mcc_estimate",
    "evaluate.command.transform" : "make mcc_acc",
    "evaluate.max-util.BRAM" : 0.8,
    "evaluate.max-util.DSP" : 0.8,
    "evaluate.max-util.FF" : 0.8,
    "evaluate.max-util.LUT" : 0.8,
    "evaluate.worker-per-part" : 2,
    "project.backup" : "BACKUP_ERROR",
    "project.fast-output-num" : 4,
    "project.name" : "dse_project",
    "search.algorithm.exhaustive.batch-size" : 2,
    "search.algorithm.gradient.fine-grained-first" : true,
    "search.algorithm.gradient.latency-threshold" : 64,
    "search.algorithm.gradient.quality-type" : "performance",
    "search.algorithm.name" : “bottleneck",
    "timeout.bitgen" : 480,
    "timeout.exploration" : 640,
    "timeout.hls" : 120,
    "timeout.transform" : 20
```

GEMM Kernel as an Example

- Either add your candidate pragmas yourself:
  - Define the pragmas in [DS_CONFIG].json
  - Run DSE using the following command:
    - `$ dse [PROJECT_DIR] work_dir [DS_INFO].json fast`

- Or, let AutoDSE do it:
  - Only generate the design space:
    - `$ ds_generator [-I<INCLUDE_DIR>] [file].c/cpp`
      - It will create two files:
        - rose_merlinkernel_[KERNEL_NAME].[c/cpp]
          - The kernel code with optimization pragmas added
        - ds_info.json
          - The options for the pragmas and running AutoDSE
  - Generate the design space and run DSE:
    - `$ autodse [PROJECT_DIR] work_dir [file].c/cpp fastgen`

Sample output

```
#pragma ACCEL kernel
void gemm(double m1[4096],double m2[4096],double prod[4096]) {

#pragma ACCEL PIPELINE auto{__PIPE__L0}
#pragma ACCEL TILE FACTOR=auto{__TILE__L0}
#pragma ACCEL PARALLEL FACTOR=auto{__PARA__L0}
  outer: for (int i = 0; i < 64; i++) {
#pragma ACCEL PIPELINE auto{__PIPE__L1}
#pragma ACCEL TILE FACTOR=auto{__TILE__L1}
#pragma ACCEL PARALLEL FACTOR=auto{__PARA__L1}
    middle: for (int j = 0; j < 64; j++) {
      int i_col = i * 64;
      double sum = (double )0;
#pragma ACCEL PARALLEL FACTOR=auto{__PARA__L2}
      inner: for (int k = 0; k < 64; k++) {
        int k_col = k * 64;
        double mult = m1[i_col + k] * m2[k_col + j];
        sum += mult;
      }
      prod[i_col + j] = sum;
} } }
```

GEMM Kernel as an Example

```
[   0m]    INFO Main: Building the scope map
[   7m]    INFO Report: DSE Configure
[   7m]    INFO Report: +-------------------+--------------+
[   7m]    INFO Report: |       Config      |    Value     |
[   7m]    INFO Report: +===================+==============+
[   7m]    INFO Report: | Project           | dse_project  |
[   7m]    INFO Report: +-------------------+--------------+
[   7m]    INFO Report: | Backup mode       | BACKUP_ERROR |
[   7m]    INFO Report: +-------------------+--------------+
[   7m]    INFO Report: | Fast mode output # | 4            |
[   7m]    INFO Report: +-------------------+--------------+
[   7m]    INFO Report: | Execution mode    | fastgen-dse  |
[   7m]    INFO Report: +-------------------+--------------+
[   7m]    INFO Report: | Search approach   | bottleneck   |
[   7m]    INFO Report: +-------------------+--------------+
[   7m]    INFO Report: | DSE time          | 1200         |
[   7m]    INFO Report: +-------------------+--------------+
[   7m]    INFO Report: | HLS time          | 80           |
[   7m]    INFO Report: +-------------------+--------------+
[   7m]    INFO Report: The actual elapsed time may be over the set up exploration time
[   7m]    INFO Report: because we do not abandon the effort of running cases
[   7m]    INFO Main: Compiling design space
[   7m]    INFO DSProc: Design space contains 3415104 valid design points
[   7m]    INFO DSProc: Finished design space compilation
[   7m]    INFO Main: Partitioning the design space to at maximum 4 parts
[   7m]    INFO Main: 4 parts generated
[   7m]    INFO Main: Start the exploration
[   7m]    INFO Main: 4 explorers have been launched
[  10m]    INFO Report: Best result reporting...
[  10m]    INFO Report: +-------+-------+-------------------------------------------+
[  10m]    INFO Report: |Quality|Perf.  |Resource                                   |
[  10m]    INFO Report: +-------+-------+-------------------------------------------+
[  16m]    INFO Report: |2.9e-05|3.5e+04|BRAM:3.0%, DSP:0.0%, LUT:1.0%, FF:0.0%     |
[  16m]    INFO Report: +-------+-------+-------------------------------------------+
[  28m]    INFO Report: |5.3e-05|1.9e+04|BRAM:9.0%, DSP:19.0%, LUT:48.0%, FF:6.0%   |
[  28m]    INFO Report: +-------+-------+-------------------------------------------+
[  79m]    INFO Report: |7.7e-05|1.3e+04|BRAM:5.0%, DSP:1.0%, LUT:2.0%, FF:1.0%     |
[  79m]    INFO Report: +-------+-------+-------------------------------------------+
[ 171m]    INFO Report: |1.3e-04|7.9e+03|BRAM:3.0%, DSP:19.0%, LUT:52.0%, FF:7.0%   |
[ 171m]    INFO Report: +-------+-------+-------------------------------------------+
[ 196m] Explored 44 points, still working.../
```

Sample output

```
#pragma ACCEL kernel
void gemm(double m1[4096],double m2[4096],double prod[4096]) {

#pragma ACCEL PIPELINE auto{__PIPE__L0}
#pragma ACCEL TILE FACTOR=auto{__TILE__L0}
#pragma ACCEL PARALLEL FACTOR=auto{__PARA__L0}
  outer: for (int i = 0; i < 64; i++) {
#pragma ACCEL PIPELINE auto{__PIPE__L1}
#pragma ACCEL TILE FACTOR=auto{__TILE__L1}
#pragma ACCEL PARALLEL FACTOR=auto{__PARA__L1}
    middle: for (int j = 0; j < 64; j++) {
      int i_col = i * 64;
      double sum = (double )0;
#pragma ACCEL PARALLEL FACTOR=auto{__PARA__L2}
      inner: for (int k = 0; k < 64; k++) {
        int k_col = k * 64;
        double mult = m1[i_col + k] * m2[k_col + j];
        sum += mult;
      }
      prod[i_col + j] = sum;
} } }
```

Thank You!