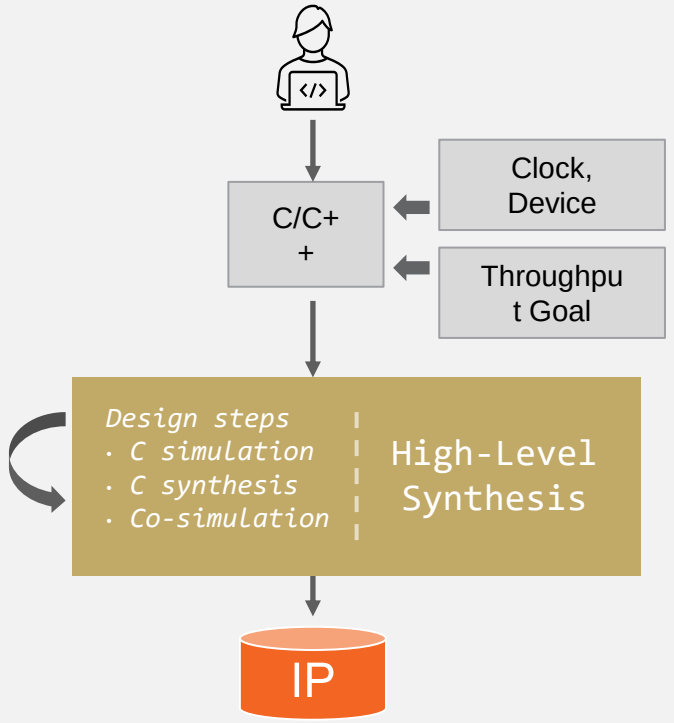
AMD Vitis™ HLS Performance Pragma

AMD
together we advance_

Performance Pragma – Methodology

Challenges of Traditional HLS Optimization



AMD Vitis™ HLS compiles C++ to RTL

Pragmas optimize design metrics:

- Latency
- Throughput
- Resources

- In the classic workflow, achieving performance goals **requires expertise in choosing the right combination of pragmas**
- Pragma-based designs can be **inflexible and sensitive to changes**
  - Vision library color detection uses over 40+ classic pragmas
    - unroll, pipeline, flatten…
  - Any changes in throughput could affect these 40+ pragmas...

Choosing the Right Combination of Pragmas can be a Challenge!!!

Performance Pragma: Simplifying HLS Optimization

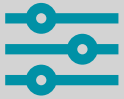
AMD Performance Pragma simplifies HLS optimization

Allows users define a high-level throughput goal

Shifts manual pragma selection optimization burden to the compiler

Enables Automatic Pragma Generation: No more manual pragma guessing!

Intelligently infers and applies optimizations (pipelining, unrolling, etc.)



Offers flexible throughput control via target specification

Tool automatically determines optimal pragma configuration

Represents a new, higher-level way to constrain design throughput

Provides a more intuitive and efficient path to desired performance

Performance Pragma

Performance Pragma can be applied to a top-level function or individual loops

Top-Level Pragma

- Defines a design-wide throughput goal
- Guides the compiler to optimize the entire design
- Automatically infers and applies loop-level pragmas based on analysis

Loop-Level Pragma

- Targets specific loops for local control
- Can be automatically inferred based on top-level performance pragma or manually applied
- Enables fine-grain optimization, infers classic pragmas (pipeline, unroll, etc...)

Benefits

Precise Control of Loop Behavior:

Optimizes critical loop throughput

Support for Top-Down Goals:

Helps achieve system-level performance targets

Performance Pragma Methodology

Offers a streamlined approach to guide the HLS optimization process to achieve optimal performance in the user's hardware implementation

Calculate the Performance Target Based on Throughput Goal

- Consider a video application aiming for a frame rate of 60 frames per second (60 fps) as an example

Performance Pragma Methodology

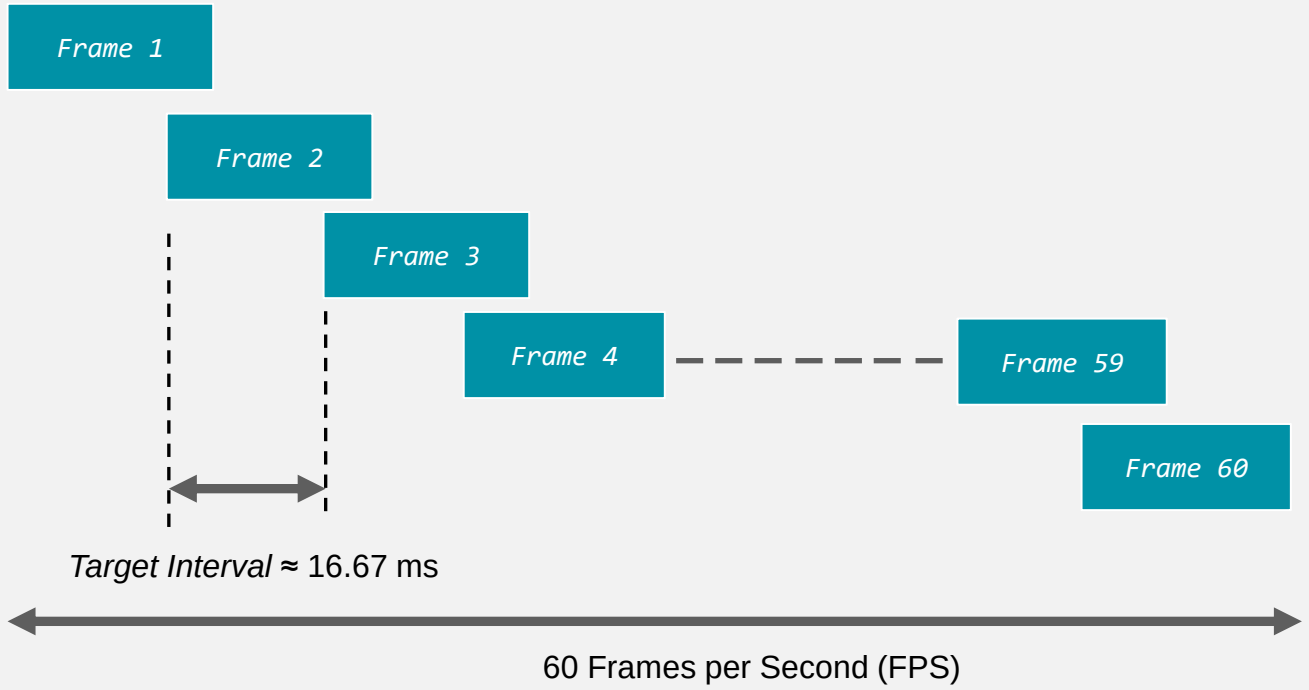
Offers a streamlined approach to guide the HLS optimization process to achieve optimal performance…

Step 1 → Step 2 → Step 3 → Step 4 → Step 5 → Step 6 → Step 7

Determine the Top-Level Performance Target (`target_ti`)

- To achieve the 60 FPS target, the top-level function must be ready to process a new frame within 1/60th of a second
- This yields the top-level performance target:

target_ti = 1 / FPS = 1 / 60 seconds ≈ 16.67 ms



Performance Pragma Methodology

Offers a streamlined approach to guide the HLS optimization process to achieve optimal performance in the user's hardware implementation

Re-architect the Code for Dataflow

- Design necessitates a re-architecture into the load-compute-store (LCS) paradigm and employ the dataflow pragma
- Allows AMD Vitis™ HLS to effectively:
  - Optimize the code
  - Exploit potential parallelism

Performance Pragma Methodology

Offers a streamlined approach to guide the HLS optimization process to achieve optimal performance in the user's hardware implementation

Run C Simulation and Determine Loop Trip Counts

- Metric is vital as the performance pragma algorithm requires precise loop budgeting
- By default, variable loop bounds as "1024," which can lead to inaccurate performance estimations
- For variable loops, users should provide dynamic trip count information using the pragma HLS loop_tripcount max=N

Performance Pragma Methodology

Offers a streamlined approach to guide the HLS optimization process to achieve optimal performance in the user's hardware implementation

Add the Top-Level Performance Target

- Apply the desired performance goal using the top-level performance pragma

  #pragma HLS performance target_ti = 16.67 ms/cycle

Performance Pragma Methodology

Offers a streamlined approach to guide the HLS optimization process to achieve optimal performance in the user's hardware implementation

Identify Bottleneck Loops (if any)

- Run C synthesis and analyze the C synthesis report to identify any loops or functions that fail to meet the specified target_ti requirement
- **Note:** Even if the performance targets are not fully achieved, the pragma will ensure that the design meets its timing requirements

Performance Pragma Methodology

Offers a streamlined approach to guide the HLS optimization process to achieve optimal performance in the user's hardware implementation

Add/Update Local Performance Targets

- For critical loops identified as bottlenecks, specify loop-level performance targets
- Rerun C synthesis and analyze the updated reports
- Continue this iterative process of refining loop-level targets until the overall design meets the desired performance goal

Note: If the desired performance targets are still not met, it is recommended to use more granular, classic pragmas to further enhance performance without violating the established timing constraints

Performance Pragma Methodology: Key Differences

Key differences compared to the traditional approach of manual pragma insertion

Top-Level Throughput Constraint

Unlike starting optimization at individual loop level, you define a system-wide performance target first

Needs Trip Count for Dynamic Loops

Provides the tool with crucial information for accurate performance estimation, especially for loops with variable iterations

May Need Loop-Level Performance Pragma Too

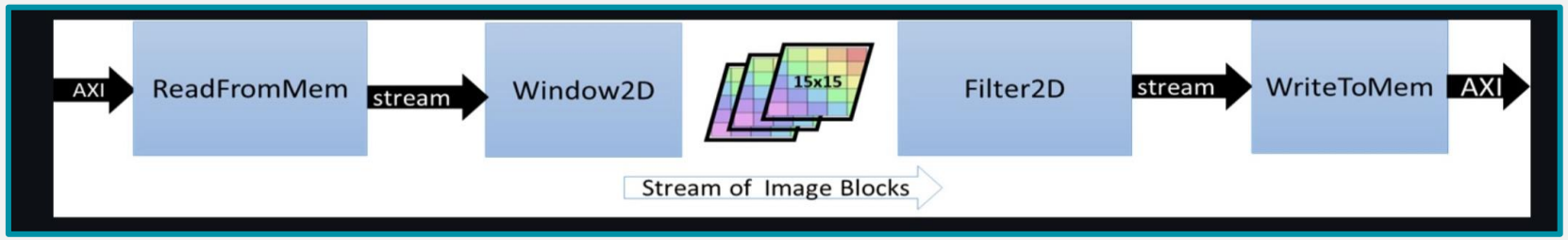
While the tool automates, you can still fine-tune specific bottlenecks for more granular control

Performance Pragma in Action: Convolution Design

Convolution 2D: Calculate the Performance Target

Calculate the Performance Target Based on Throughput Goal

- Convolution function process an HD 140 frames per second @ 300 MHz clock…



Convolution 2D: Determine the Top-Level Performance Target

Determine the Top-Level Performance Target (target_ti)

Target Interval (target_ti) can be expressed in time (ms) *or* number of cycles:

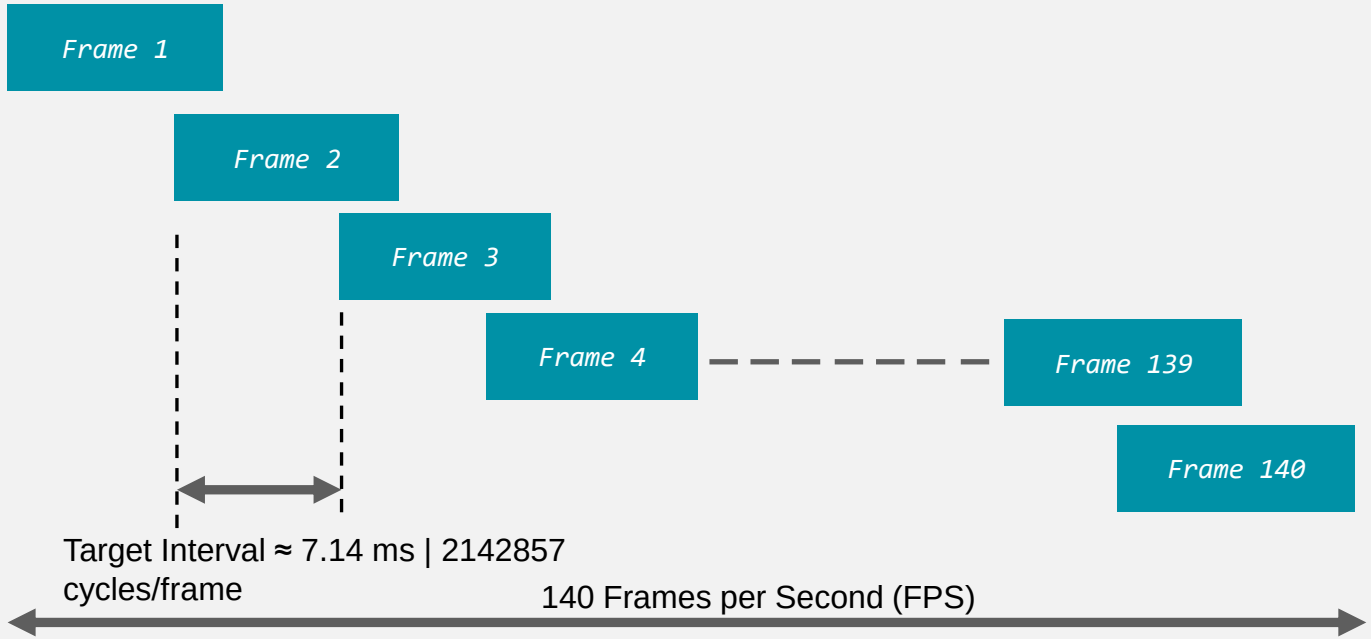
- Time: 140 frames/sec hence: `target_ti`

  (milliseconds/frame) = 1000 /140 = `7.14` ms

- Cycles: 140 frames/sec at 300 MHz hence:
  - `target_ti` = (kernel freq.) / (throughput) =

    (300 * 10^6 cycles/second) /140 =

    `2,142,857.14` cycles/frame



Convolution 2D: Run C Simulation with Code Analyzer

Run C Simulation and Determine Loop Trip Counts

- Run C simulation with Code Analyzer
- Add trip counts for dynamic variable loops

```
        for (int x = 0; x < width; x++)
        {
#pragma HLS LOOP_TRIPCOUNT max=1920
            // Read a 2D window of pixels
            window w = window_stream.read();
```

Convolution 2D: Re-architect the Code

Re-architect the Code for Dataflow

- Code for load-compute-store…

```
void Filter2DKernel(
        const char           coeffs[256],
        float                factor,
        short                bias,
        unsigned short       width,
        unsigned short       height,
        unsigned short       stride,
        const unsigned char  src[MAX_IMAGE_WIDTH*MAX_IMAGE_HEIGHT],
        unsigned char        dst[MAX_IMAGE_WIDTH*MAX_IMAGE_HEIGHT])
  {

#ifdef STEP1
#pragma HLS performance target_ti = 2142857
#endif
#pragma HLS dataflow
```

Convolution 2D: Add the Top-Level Performance Target

Add the Top-Level Performance Target

- Apply the top-level performance pragma using target_ti
- Enable performance pragma via
  - TCL: config_dse -enable=true
  - Config: syn.dse.enabled=1

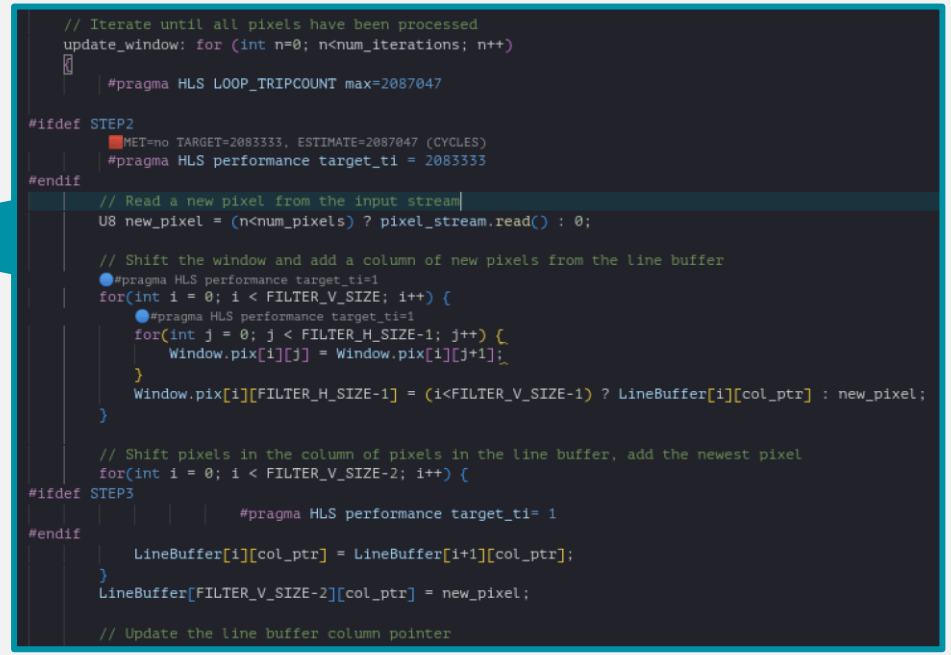
```
void Filter2DKernel(
        const char           coeffs[256],
        float                factor,
        short                bias,
        unsigned short       width,
        unsigned short       height,
        unsigned short       stride,
        const unsigned char  src[MAX_IMAGE_WIDTH*MAX_IMAGE_HEIGHT],
        unsigned char        dst[MAX_IMAGE_WIDTH*MAX_IMAGE_HEIGHT])
  {

#ifdef STEP1
#pragma HLS performance target_ti = 2142857
#endif
#pragma HLS dataflow
```

Convolution 2D: Detect Performance Bottlenecks

Identify Bottleneck Loops (if any)

- Identify underperforming process
  - Here Window2D does not meet the performance target of 45,915,037 cycles per frame

MODULES & LOOPS	ISSUE	TARGET TI(CYCLES)	ESTIMATED TI(CYCLES)	PERFORMANCE CONSTRAINT
Filter2DKernel (5)		2,142,857	45,915,038	pragma
entry_proc				
ReadFromMem (2)		2,073,873	2,074,007	
Window2D (1)		532,196,988	45,915,037	
update_window (2)		532,196,985	45,915,034	inferred
Outline_VITIS_LOOP_141_1		1	1	
Window2D_Pipeline_VITIS_LOOP_149_3 (1)				
Filter2D (2)		575,394	2,073,858	
WriteToMem (1)		2,073,612	2,073,676	



```
    // Iterate until all pixels have been processed
    update_window: for (int n=0; n<num_iterations; n++)
    {
        #pragma HLS LOOP_TRIPCOUNT max=2087047

#ifdef STEP2
        MET=no TARGET=2083333, ESTIMATE=2087047 (CYCLES)
        #pragma HLS performance target_ti = 2083333
#endif
        // Read a new pixel from the input stream
        U8 new_pixel = (n<num_pixels) ? pixel_stream.read() : 0;

        // Shift the window and add a column of new pixels from the line buffer
        #pragma HLS performance target_ti=1
        for(int i = 0; i < FILTER_V_SIZE; i++) {
            #pragma HLS performance target_ti=1
            for(int j = 0; j < FILTER_H_SIZE-1; j++) {
                Window.pix[i][j] = Window.pix[i][j+1];
            }
            Window.pix[i][FILTER_H_SIZE-1] = (i<FILTER_V_SIZE-1) ? LineBuffer[i][col_ptr] : new_pixel;
        }

        // Shift pixels in the column of pixels in the line buffer, add the newest pixel
        for(int i = 0; i < FILTER_V_SIZE-2; i++) {
#ifdef STEP3
                #pragma HLS performance target_ti= 1
#endif
            LineBuffer[i][col_ptr] = LineBuffer[i+1][col_ptr];
        }
        LineBuffer[FILTER_V_SIZE-2][col_ptr] = new_pixel;

        // Update the line buffer column pointer
```

Convolution 2D: Add/Update Local Performance Targets (1/2)

Add/Update Local Performance Targets

- Apply loop-level performance pragmas
  - Specify loop-level performance pragmas for the Window2D function to achieve target_ti

```
    update_window: for (int n=0; n<num_iterations; n++)
    {
        #pragma HLS LOOP_TRIPCOUNT max=2087047

#ifdef STEP2
        MET=no TARGET=2142857, ESTIMATE=45915034 (CYCLES)
        #pragma HLS performance target_ti = 2142857
#endif
```

Convolution 2D: Add/Update Local Performance Targets (2/2)

Target Still Not Reached

MODULES & LOOPS	ISSUE	TARGET TI(CYCLES)	ESTIMATED TI(CYCLES)	PERFORMANCE CONSTRAINT
Filter2DKernel (5)		2,142,857	45,915,038	pragma
entry_proc				
ReadFromMem (2)		2,073,873	2,074,007	
Window2D (1)		532,196,988	45,915,037	
update_window (2)		2,142,857	45,915,034	pragma
Outline_VITIS_LOOP_141_1		1	1	
Window2D_Pipeline_VITIS_LOOP_149_3 (1)				
VITIS_LOOP_149_3		26	13	inferred
Filter2D (2)		575,394	2,073,858	
WriteToMem (1)		2,073,612	2,073,676	

Csynth report with top and loop-level Performance pragma

Solution

Set a target loop TI = 1 for this shift loop to nudge the compiler to execute the loop in a single cycle

```
        for(int i = 0; i < FILTER_V_SIZE-2; i++) {
#ifdef STEP3
                    #pragma HLS performance target_ti= 1
#endif
            LineBuffer[i][col_ptr] = LineBuffer[i+1][col_ptr];
        }
```

Voila!! Meets performance!

Convolution 2D: Results

Goal

2,142,857.14 cycles/frame
or 140 frames per second

Achieved

2,087,651 cycles/frame or
~144 frames per second

Category	Using Classic Pragmas	Using Performance Pragma
Target Interval	2,087,651	2,087,651
Optimizations Pragmas in the Design	8	3 (2.6X fewer pragmas)

Performance Pragma: Limitations

Limitations of Performance Pragma

Pragma Precedence ("OFF" pragmas prevent automatic inference)

PIPELINE OFF	UNROLL OFF	FLATTEN	ARRAY_PARTITION OFF
Disables automatic pipelining for a loop	Disables automatic unrolling for a loop	Disables automatic partitioning for a local array	Prevents automatic flattening for a loop

Interface Port Limitation	
	• Arrays at the top function interface are NOT auto-partitioned by default (potential bottleneck) • Enable with ***config_array_partition -throughput_driven=aggressive***

Limitations of Performance Pragma

Features	Behavior/Limitation
ap_cint	The tool exits with an explicit warning message
ap_(u)int / ap_(u)fixed	• No performance models are inaccurate
Big constant arrays of ap_int / ap_fixed	• Using them with macro NON_C99STRING will result in a compilation error
std::complex<ap_fixed>	Lead to a compiler error on Windows
HLS IP blocks (FFT, FIR, ...), hls_math.h, ap_wait, hls::vector, ap_axis/ap_axiu	• No performance models are inaccurate
hls::stream_of_blocks, hls::task, hls::split / hls::merge, hls::print, hls::half, ap_utils.h, hls_fpo.h, ap_float, RTL Blackboxes, OpenCL, hls::burst_maxi, hls::fence, hls::directio	The tool exits with an explicit warning message
Deprecated HLS pragmas	Assertion failure
Function pipeline with sub loop(s)	Assertion failure

Summary

Performance Pragma simplifies complex HLS optimization by enabling users to define a high-level throughput goal, shifting the optimization burden to the compiler for automatic pragma generation and application of key transformations, offering flexible throughput control and a more efficient path to desired hardware performance.

General Disclaimer and Attribution Statement

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18u.


AMD