Harris Corner Detector

Team #8
陳昱銓 陳冠豪 楊竣佑
https://github.com/yqchenee/ACA_21S_final

Outline

- Introduction
- System & Algorithms
- Optimization & Experimental Result
- Conclusion

Outline

- Introduction
- System & Algorithms
- Optimization & Experimental Result
- Conclusion

Introduction

- Feature extraction is a common technique in computer vision and machine learning application.
- Harris corner detector is a method to extract features of corners in the image.
- In this project, we exploit parallel computation to accelerate the process of harris corner detector

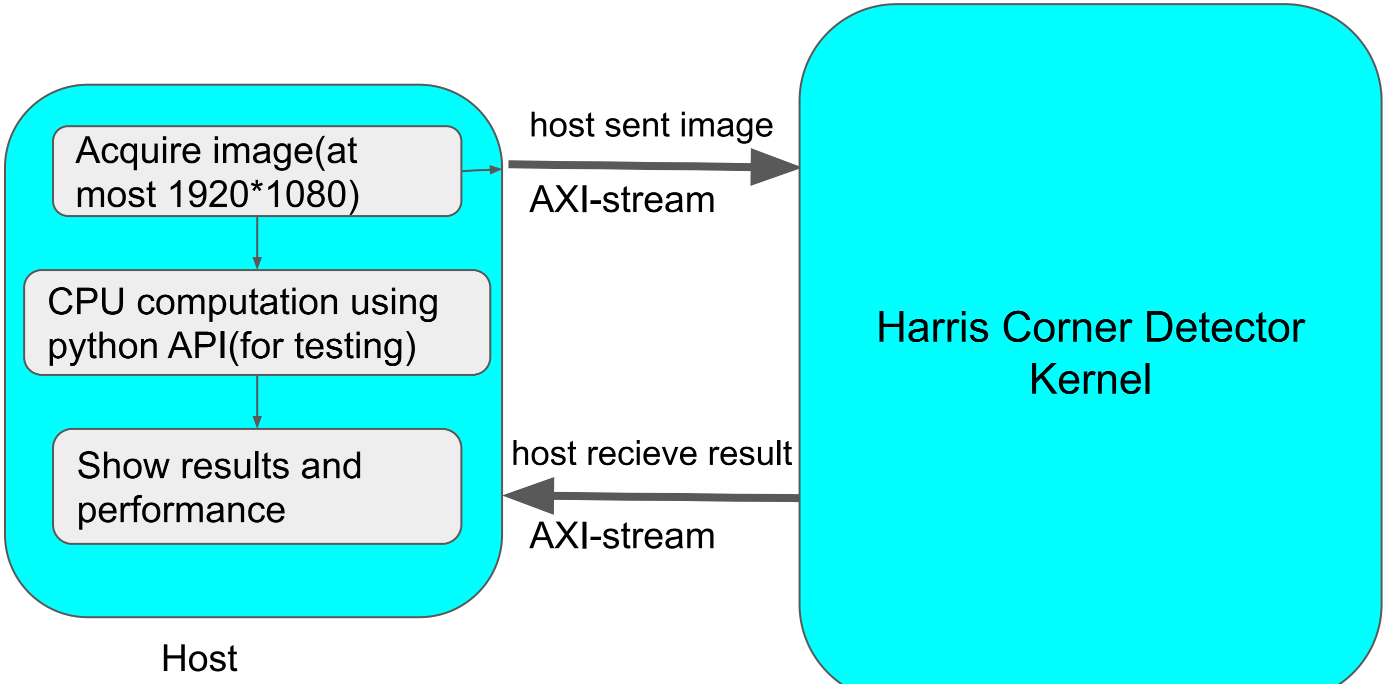
Introduction

- We use Xilinx Vivado HLS to implement hardware design.
- Hardware can be deployed on PYNQ-Z2 board.
- We explore some optimization methods.
- Our target spcification: performance(latency) and precision
    - We try to keep calculation error low enough and optimize performance as much as possible

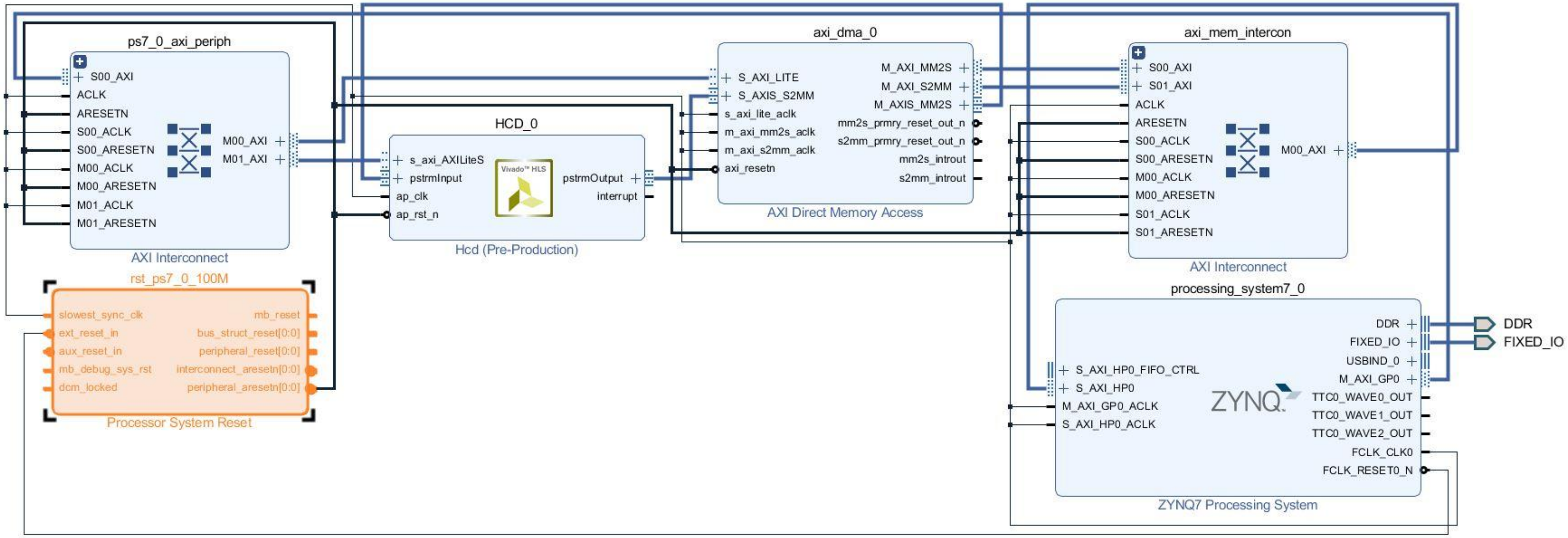
Outline

- Introduction
- System & Algorithms
- Optimization & Experimental Result
- Conclusion

System Overview



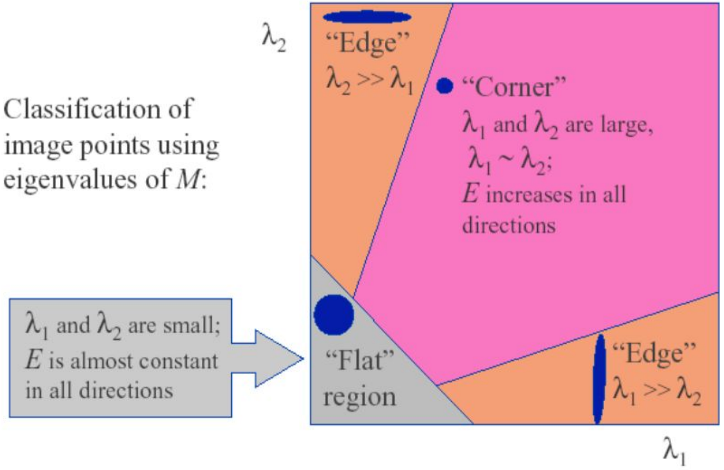
PL Block Diagram



Harris Corner Detector Algorithm

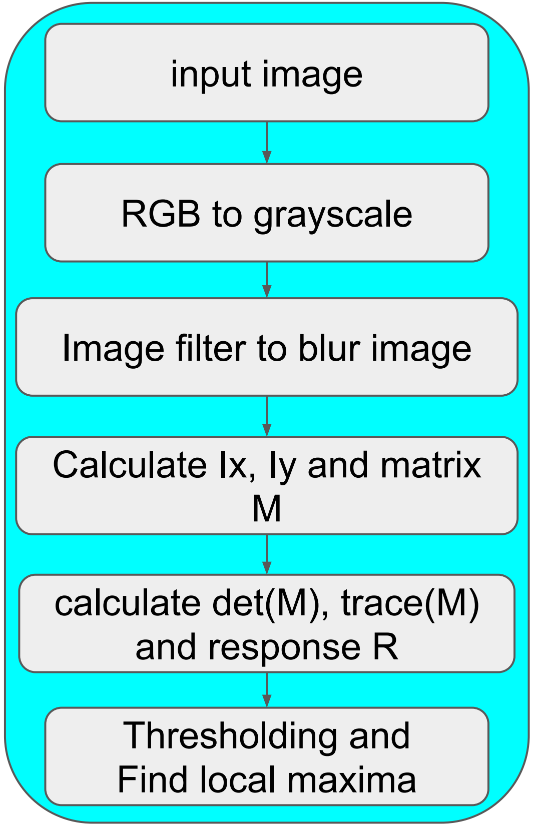
- Image filter
  - gaussianBlur
- Ix, Iy
  - gradients on x, y direction

$$M = \sum_{(x,y)\in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

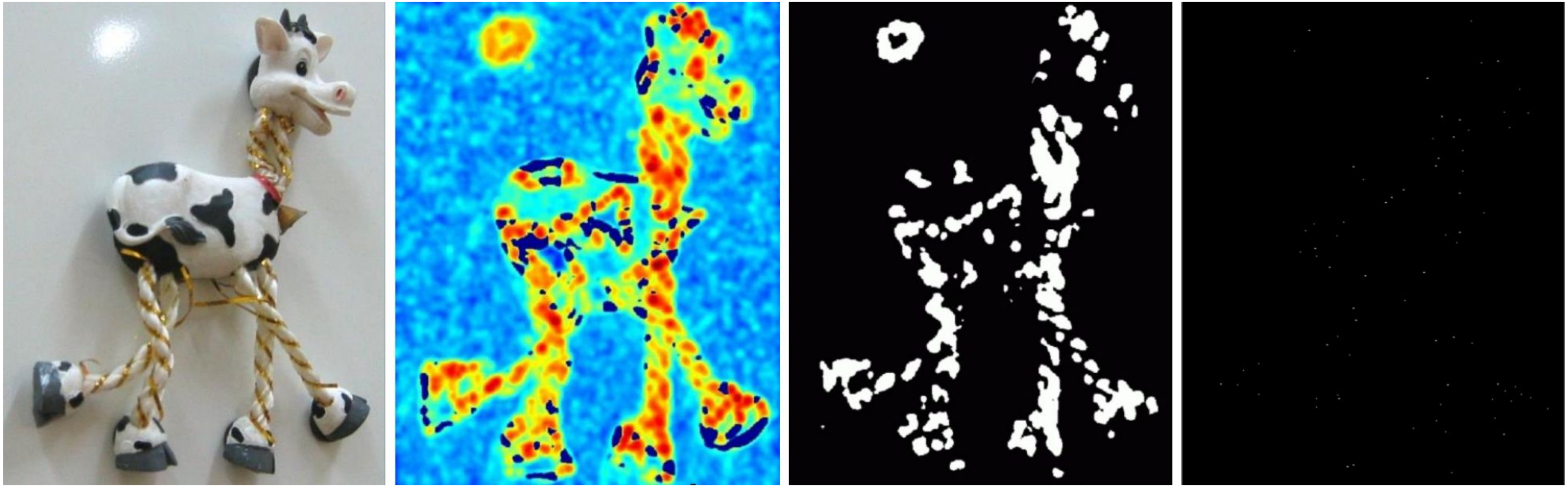


$$R = det(M) - k(trace(M))^2 = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$

Harris Corner Detector Kernel



Harris Corner Detector Example



original image → response

image filter
& matrix operation

thresholding

local maxima
(final result)

Outline

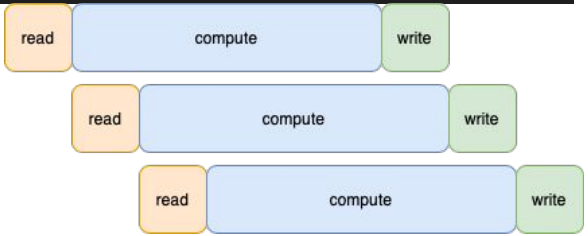
- Introduction
- System & Algorithms
- Optimization & Experimental Result
- Conclusion

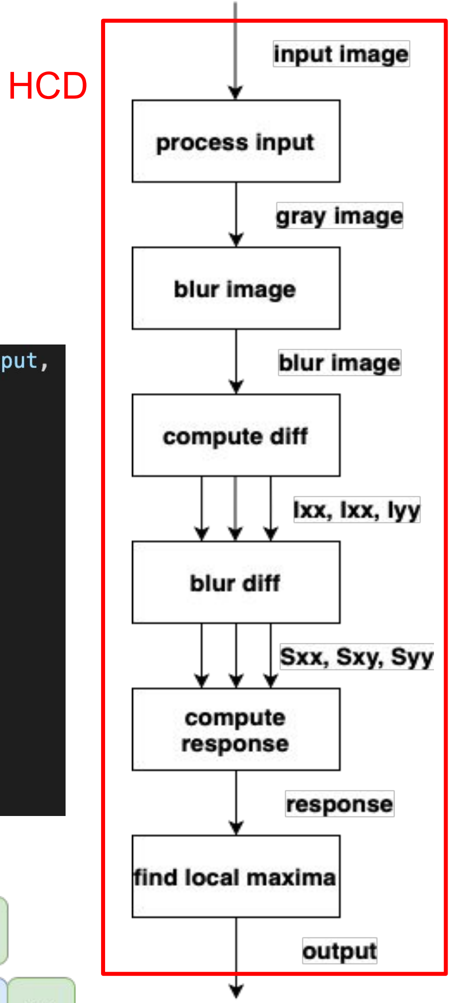
Optimize top function (HCD kernel)

- HLS pragma Dataflow
  - Make the functions operate in parallel to achieve high throughput

```
void HCD(stream_t* pstrmInput, stream_t* pstrmOutput,
        reg32_t row, reg32_t col)
{
#pragma HLS DATAFLOW
    process_input(pstrmInput, &stream_gray, row, col);

    blur_img(&stream_gray, &stream_blur, row, col);

    compute_dif(&stream_blur, &stream_Ixx, &stream_Iyy,
            &stream_Ixy, row, col);

    blur_diff(&stream_Ixx, &stream_Iyy, &stream_Ixy,
            &stream_Sxx, &stream_Syy, &stream_Sxy, row, col);

    compute_det_trace(&stream_Sxx, &stream_Syy, &stream_Sxy,
            &stream_response, row, col);

    find_local_maxima(&stream_response, pstrmOutput, row, col);
}
```

```
void sub_function(stream* input, stream* output,
                    int row, int col)
{
    for (i in row) {
        for (j in col) {
            tmp = input-> read()

            if (condition) {
                operation on tmp
            }

            output-> write(tmp);
        }
    }
}
```





Optimization inside sub-functions

- Reduce resources usage
  - Merge all loops into one loop to reduce the number of unexpectable states in finite-state machine

```
template<typename T>
void blur_img(hls::stream<T>& stream_gray, hls::str
{
    // initial read
    for (i = 0 ; i < 2 ; i++) {
        for (j = 0 ; j < MAX_WIDTH ; j++) {…
    }

    // i = 0
    for (j = 1; j < MAX_WIDTH; ++j) {…


    // i = 1 ~ MAX_HEIGHT-2
    for (i = 1; i < MAX_HEIGHT - 1; ++i) {
        for (j = 1; j < MAX_WIDTH; ++j) {…
    }

    // i = MAX_HEIGHT
    for (j = 1; j < MAX_WIDTH; ++j) {…

}
```

* Multiplexer:

Name	LUT
ColIndex_assign_5_reg_6519	9
ColIndex_assign_6_reg_6316	9
ColIndex_assign_reg_6159	9
ap_NS_fsm	1761
buf_M_0_V_address0	597
buf_M_0_V_address1	597
buf_M_0_V_d0	585
buf_M_0_V_d1	581
buf_M_1_V_address0	1185
buf_M_1_V_address1	1189
buf_M_1_V_d0	589
buf_M_1_V_d1	545
buf_M_2_V_address0	585
buf_M_2_V_address1	585

```
template<typename T>
&amp; stream_gray, hls::stream
GHT+1; i++) {
WIDTH+1; j++) {
H) {…
HT & j < MAX_WIDTH) {…
H) {…
X_WIDTH) {…
X_HEIGHT) {…
filter_1<T, ap_window<T, 3
e(blur);
}
```

Optimization inside sub-functions

- Using **ap_fixed** instead of floating point
    - Improve both utilization and timing
        - float operations consume a lot of time and resources
    - Inside functions
        - blur image (gaussian blur)
        - compute response

	latency (cycle)	FF	LUT
float	159	960	1803
ap_fixed	33	60	245

```cpp
template<typename P, typename W>
P Gaussian_filter_1(W* window)
{
    float sum = 0;
    P pixel =0;

    float op[3][3] =
    {…

    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            sum += window->getval(i,j) * op[i][j];
        }
    }
    pixel = P(sum);
    return pixel;
}
```

```cpp
template<typename P, typename W>
P Gaussian_filter_1(W* window)
{
    ap_fixed<22, 16> sum = 0;
    P pixel =0;

    ap_fixed<8, 2> op[3][3] =
    {…

    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            sum += window->getval(i,j) * op[i][j];
        }
    }
    pixel = P(sum);
    return pixel;
}
```

Array Partition- Gaussian Blur (Compute weighted sum of a window)

- Complete partition window and const array
- Entire loop can be unrolled

```
ap_fixed<12, 1> op[3][3] =
{
    {0.0751136, 0.123841, 0.0751136},
    {0.123841, 0.20418, 0.123841},
    {0.0751136, 0.123841, 0.0751136}
};
#pragma HLS array_partition variable=op complete

for (i = 0; i < 3; i++) {
    #pragma HLS unroll
    for (j = 0; j < 3; j++) {
        #pragma HLS unroll
        sum += window->getval(i,j) * op[i][j];
    }
}
```

- Expand entire loop
- Reduce one cycle compared to unrolled loop

```
ap_fixed<12, 1> op[3][3] =
{
    {0.0751136, 0.123841, 0.0751136},
    {0.123841, 0.20418, 0.123841},
    {0.0751136, 0.123841, 0.0751136}
};
#pragma HLS array_partition variable=op complete
#pragma HLS expression_balance

sum = window->getval(0,0) * op[0][0] +
      window->getval(0,1) * op[0][1] +
      window->getval(0,2) * op[0][2] +
      window->getval(1,0) * op[1][0] +
      window->getval(1,1) * op[1][1] +
      window->getval(1,2) * op[1][2] +
      window->getval(2,0) * op[2][0] +
      window->getval(2,1) * op[2][1] +
      window->getval(2,2) * op[2][2] ;
```

Loop Unrolling - local maxima (Find local maximum on a 5*5 region)

- Condiser border condiction
- Origin version that can't be unrolled

```
if (center_pixel != 0) {
    input.data =1;
    d_bound = (i-4 < 0) ? i : 4;
    l_bound = (j-4 < 0) ? 0 : j-4;
    r_bound = (j >= col) ? col-1 : j;
    for(si = 0 ; si <= d_bound; si++) {
        for(sj = l_bound; sj <= r_bound; sj++) {
            if(response_buf.getval(si, sj) > center_pixel)
                input.data = 0;
                break;
            }
        }
    }
}
```

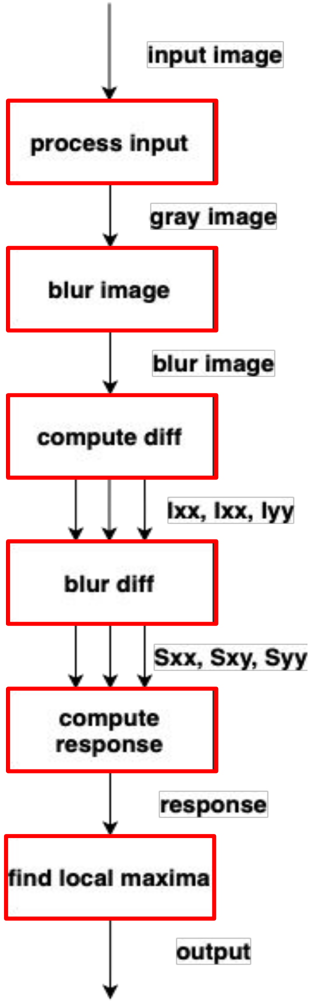
- Revised version that can be unrolled

```
if (center_pixel != 0) {
    input.data =1;
    for(sj = 0; sj < 5; sj++) {
    #pragma HLS unroll
        for(si = 0 ; si < 5; si++) {
        #pragma HLS unroll
            // handle bound at getval function
            if(response_buf.getval(si, j-sj) > center_pixel) {
                input.data = 0;
                break;
            }
        }
    }
}
```

Optimization inside sub-functions

- Improve timing in all sub-functions
  - all functions have the following structures
  - use pipeline to enhance the throughput

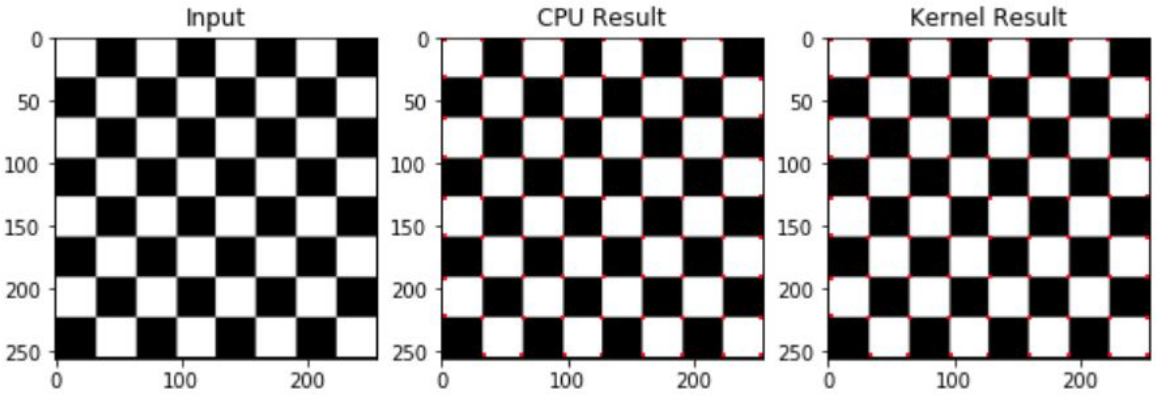
```
void sub_function(stream* input, stream* output,
                  int row, int col)
{
    for (i in row) {
        for (j in col) {
            # pragma HLS pipeline enable_flush
            tmp = input-> read()

            if (condition) {
                operation on tmp
            }

            output-> write(tmp);
        }
    }
}
```

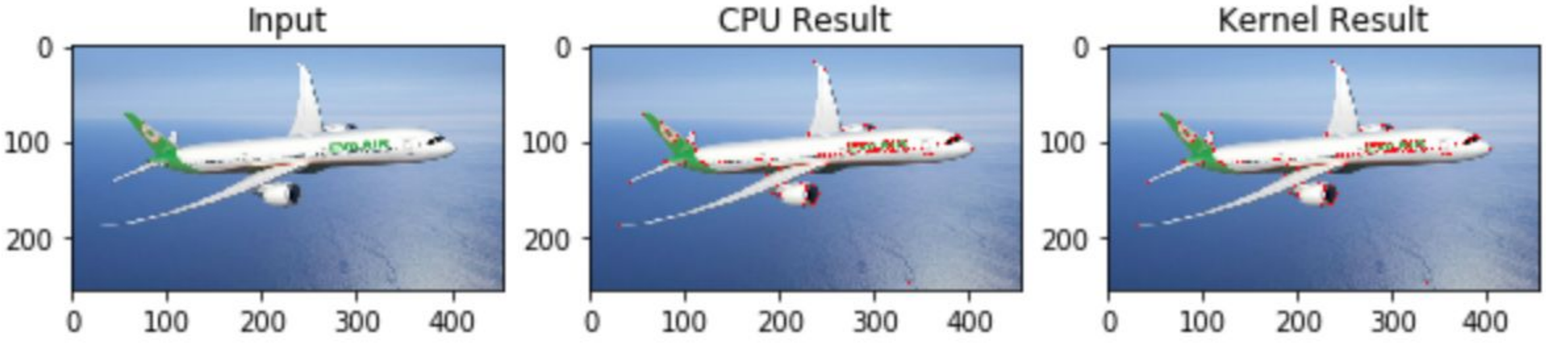


	latency			utilization			
	Synthesis report	Run time 1 (256*256)	Run time 2 (1920*1080)	BRAM_ 18k	DSP48 E	FF	LUT
no pragma	58 ms	46.96 ms [1]	1472 ms [1]	120	26	4433	11199
unroll	5.290 ms	5.860 ms [1]	171.5 ms [1]	80	92	9299	16640
pipeline + unroll	2.663 ms	3.316 ms [1]	83.51 ms [1]	80	127	16725	18057
CPU runtime		235 ms [2]	2425 ms [2]				

[1] run under pynq
[2] run with python under Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz

Experimental results





Outline

- Introduction
- System & Algorithms
- Optimization & Experimental Result
- Conclusion

Conclusion

- We use Xilinx tools to implement Harris Corner Detector into hardware design.
- Optimize the hls code to meet the resource constraint and shorten the latency.
- Deploy our design on PYNQ-Z2 board.
- Kernel function speed up over 30x compare to CPU.