An Introduction to Vitis High Level Synthesis

Duc Tri Nguyen

Algorithm

$$s^2 = \sum_{i=0}^{n-1} x_i^2 = x_0^2 + x_1^2 + x_2^2 + \ldots + x_{n-1}^2 \quad (1)$$

Equation 1 compute l^2-norm of a vector $x = (x_0, x_1, \ldots, x_{n-1})$.

However, due to bit-width limitation, and the size of vector x is undefined, it can be arbitrary long or short. We use must use **saturation arithmetic** to perform add or subtract.

Algorithm

$$\mathrm{s}^2 = \sum_{i=0}^{n-1} x_i^2 = x_0^2 + x_1^2 + x_2^2 + \ldots + x_{n-1}^2 \quad (1)$$

Can we optimize the algorithm?

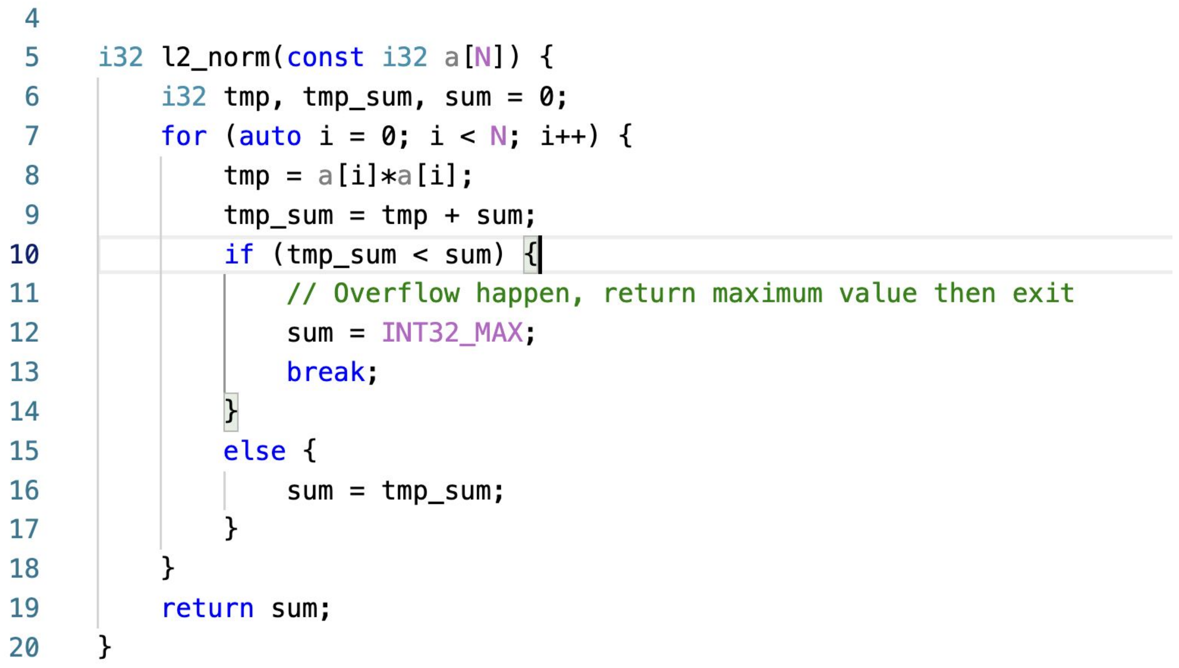
Algorithm

$$\mathrm{s}^2 = \sum_{i=0}^{n-1} x_i^2 = x_0^2 + x_1^2 + x_2^2 + \ldots + x_{n-1}^2 \quad (1)$$

Can we optimize the algorithm?

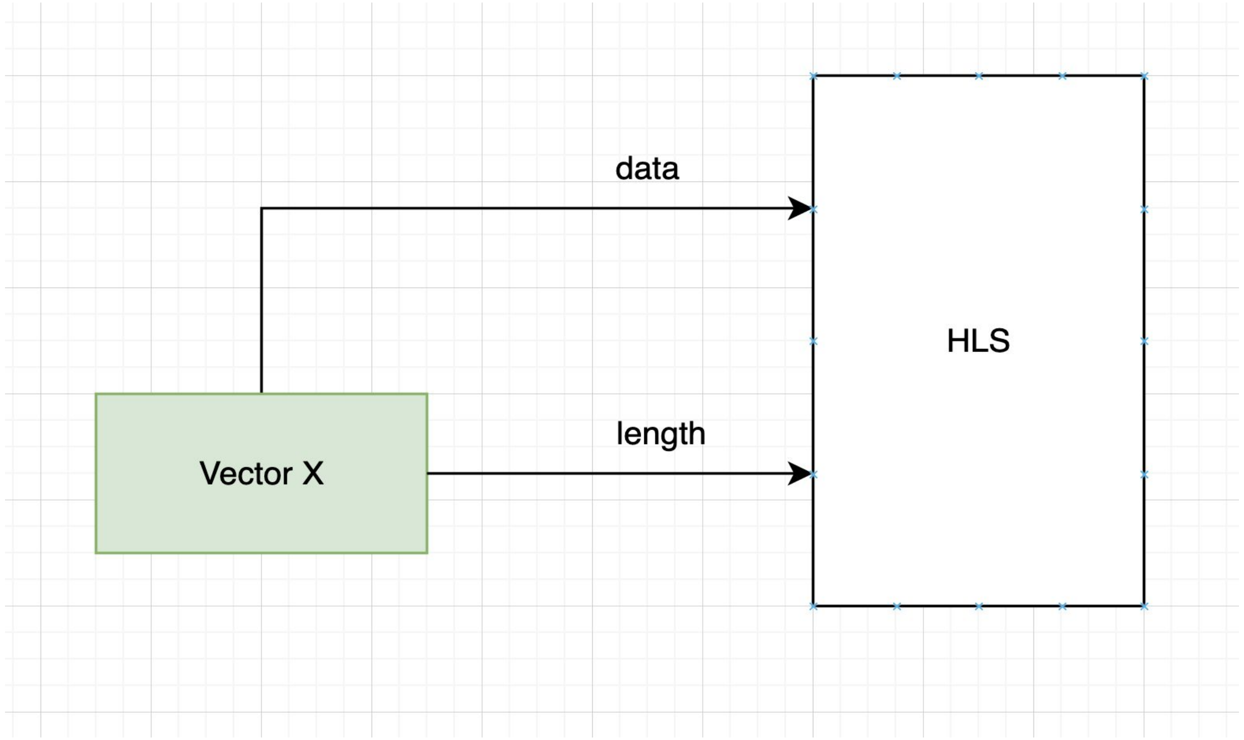
No possible algorithmic optimization!

Algorithm in C

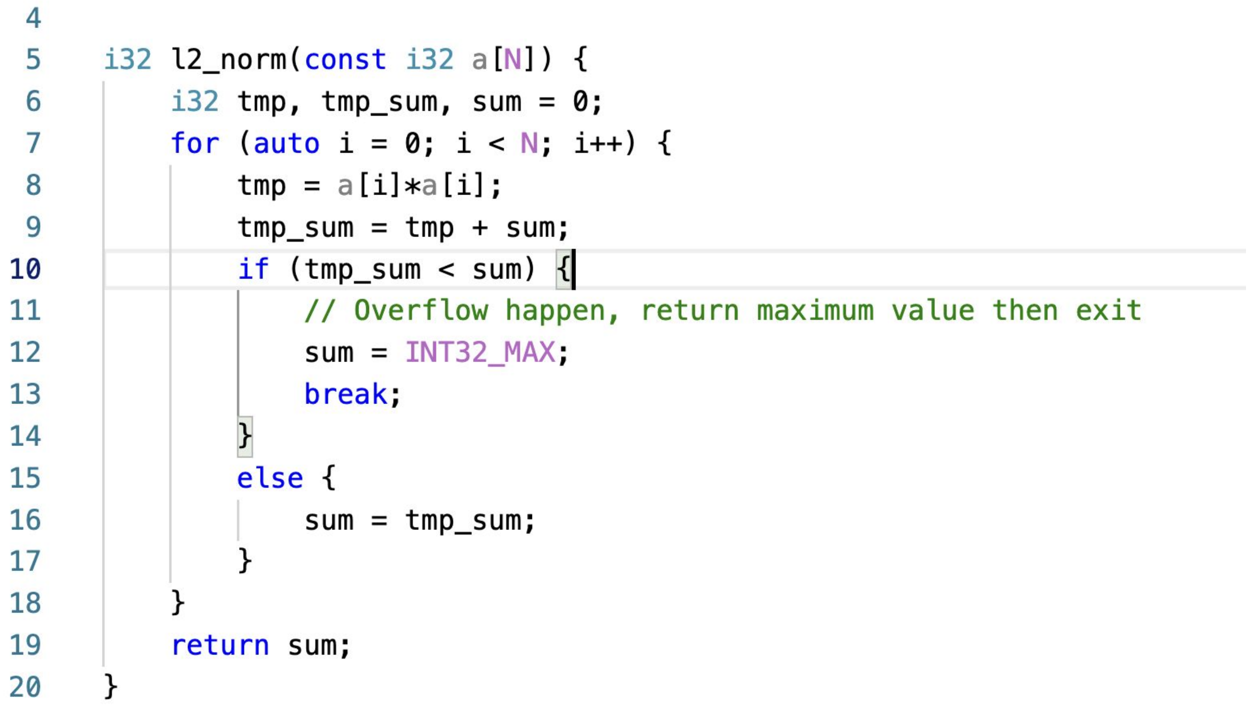


```c

i32 l2_norm(const i32 a[N]) {
    i32 tmp, tmp_sum, sum = 0;
    for (auto i = 0; i < N; i++) {
        tmp = a[i]*a[i];
        tmp_sum = tmp + sum;
        if (tmp_sum < sum) {
            // Overflow happen, return maximum value then exit
            sum = INT32_MAX;
            break;
        }
        else {
            sum = tmp_sum;
        }
    }
    return sum;
}
```

Input Flow



Reference C



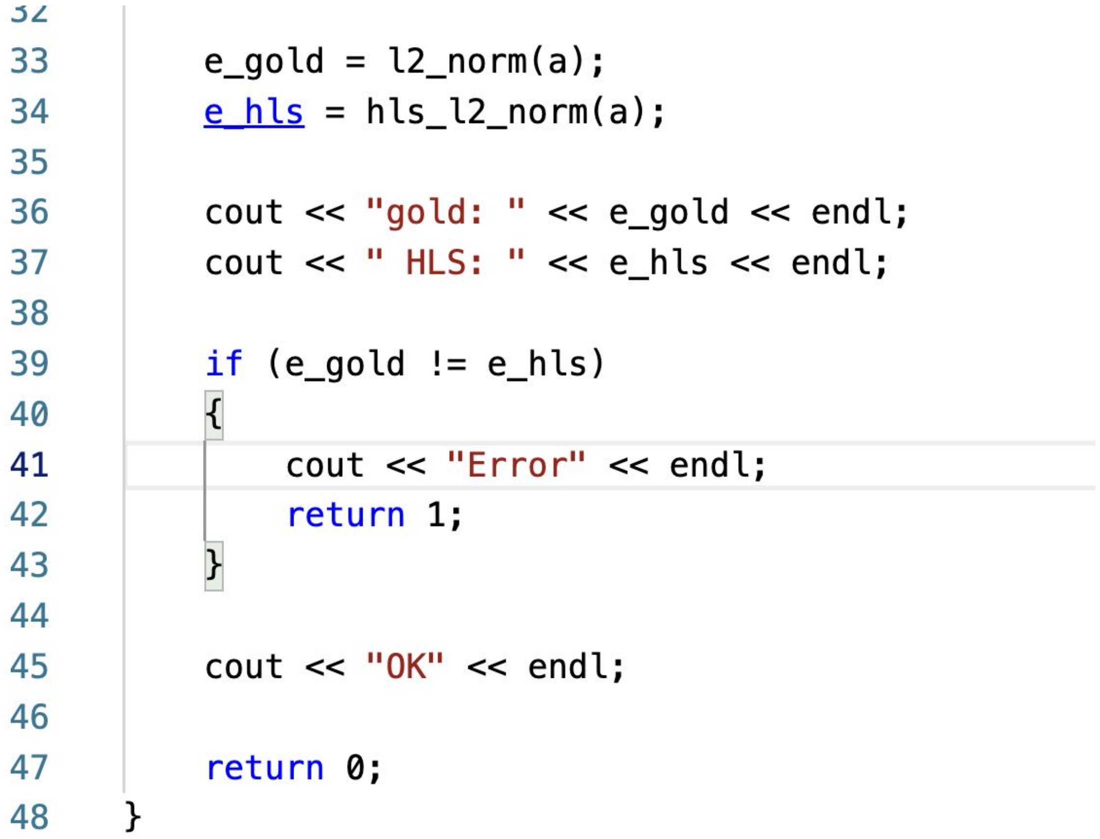
```c

i32 l2_norm(const i32 a[N]) {
    i32 tmp, tmp_sum, sum = 0;
    for (auto i = 0; i < N; i++) {
        tmp = a[i]*a[i];
        tmp_sum = tmp + sum;
        if (tmp_sum < sum) {
            // Overflow happen, return maximum value then exit
            sum = INT32_MAX;
            break;
        }
        else {
            sum = tmp_sum;
        }
    }
    return sum;
}
```

Setup

1. Setup Testbench

   Return 1 if ERROR

   Return 0 if CORRECT

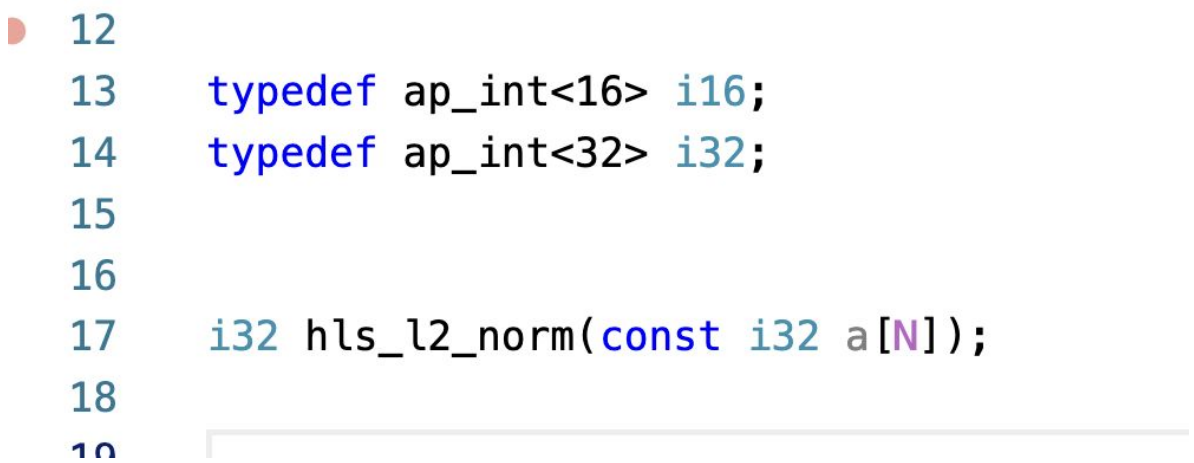


```
    e_gold = l2_norm(a);
    e_hls = hls_l2_norm(a);

    cout << "gold: " << e_gold << endl;
    cout << " HLS: " << e_hls << endl;

    if (e_gold != e_hls)
    {
        cout << "Error" << endl;
        return 1;
    }

    cout << "OK" << endl;

    return 0;
}
```

Setup

2. Setup Header

Header is used to link the testbench and the HLS code.

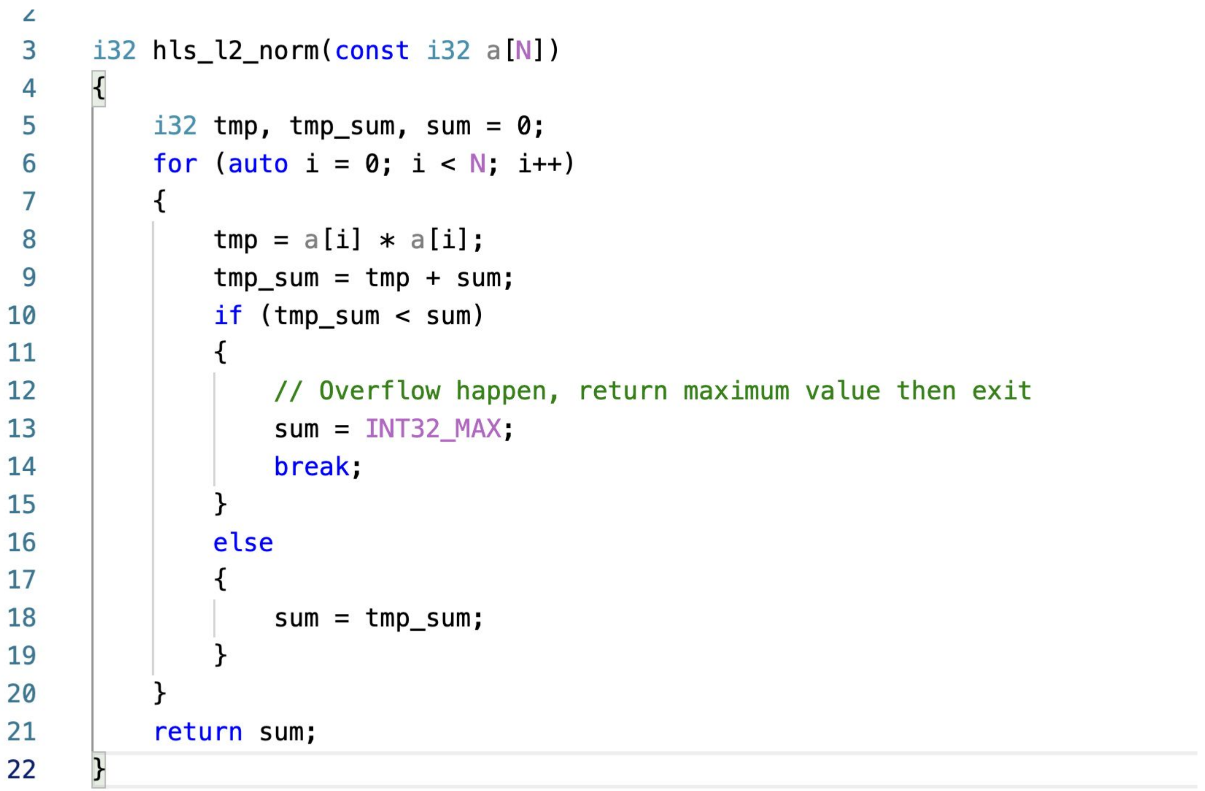


```

typedef ap_int<16> i16;
typedef ap_int<32> i32;


i32 hls_l2_norm(const i32 a[N]);

```

Setup

3. HLS function

HLS function is
same as C reference code



```
i32 hls_l2_norm(const i32 a[N])
{
    i32 tmp, tmp_sum, sum = 0;
    for (auto i = 0; i < N; i++)
    {
        tmp = a[i] * a[i];
        tmp_sum = tmp + sum;
        if (tmp_sum < sum)
        {
            // Overflow happen, return maximum value then exit
            sum = INT32_MAX;
            break;
        }
        else
        {
            sum = tmp_sum;
        }
    }
    return sum;
}
```

Baseline

The baseline is from the C reference implementation

1. Does not have Hardware Interface
2. Is not accelerated
3. Is a start point

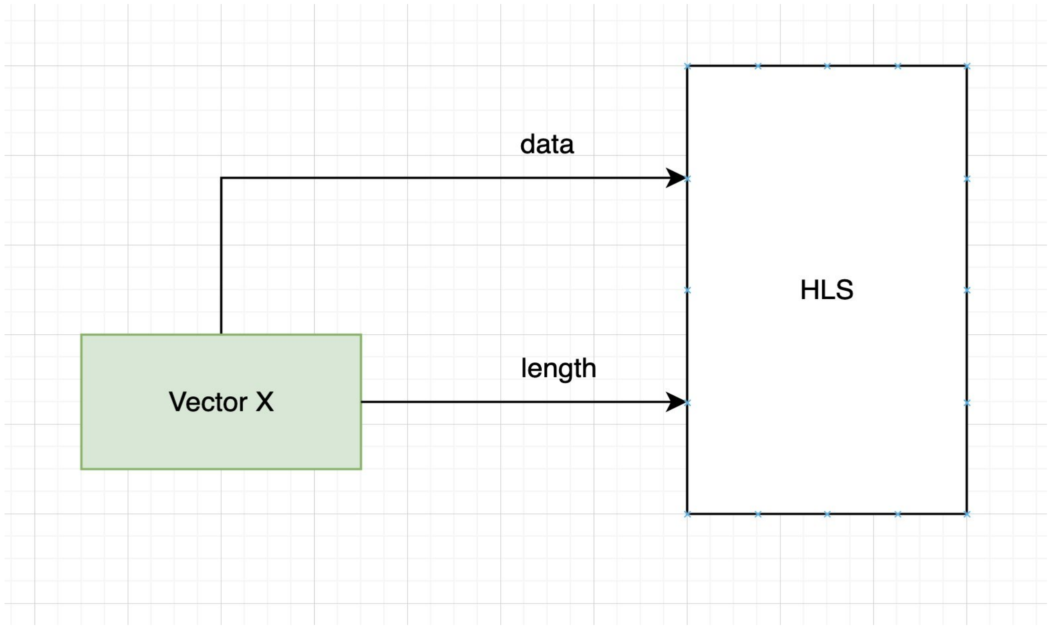
Task: Improve performance of Function in High Level Synthesis

Optimization Strategy

Constrains:
(1) No memory storage

Goal:
(2) Computer as fast as possible

Demo: Read C code

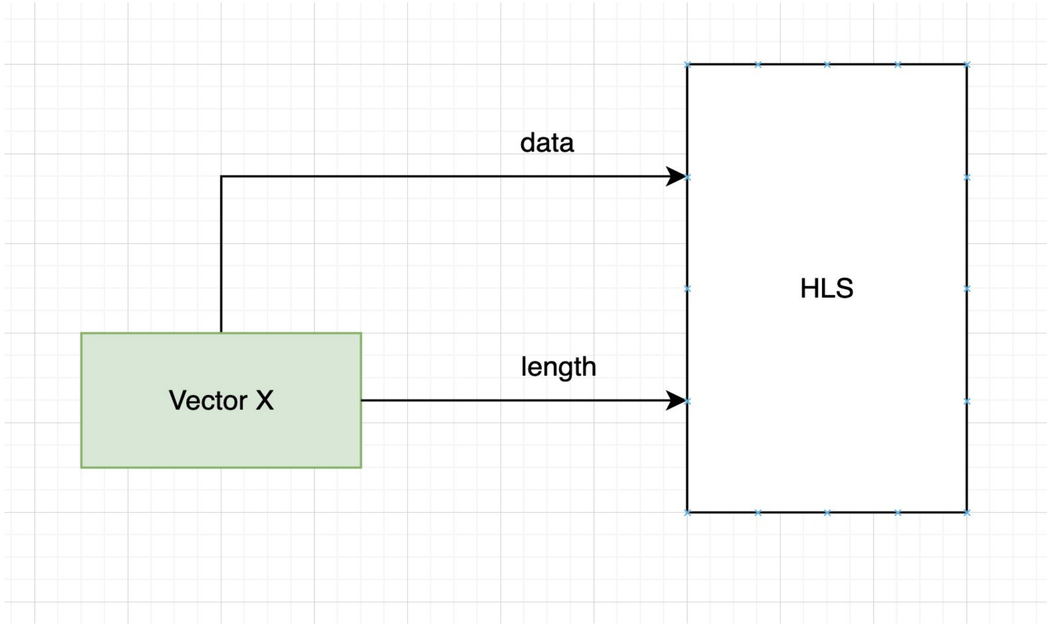


Optimization Strategy

Constrains:
(1) No memory storage

Goal:
(2) Computer as fast as possible

=> We can only increase bus-width



Increase from 32-bit to 512-bit

- Use maximum bus-width available

- In theoretical, if we can proceed 512-bit, we will improve performance 16 times faster

Increase from 32-bit to 512-bit

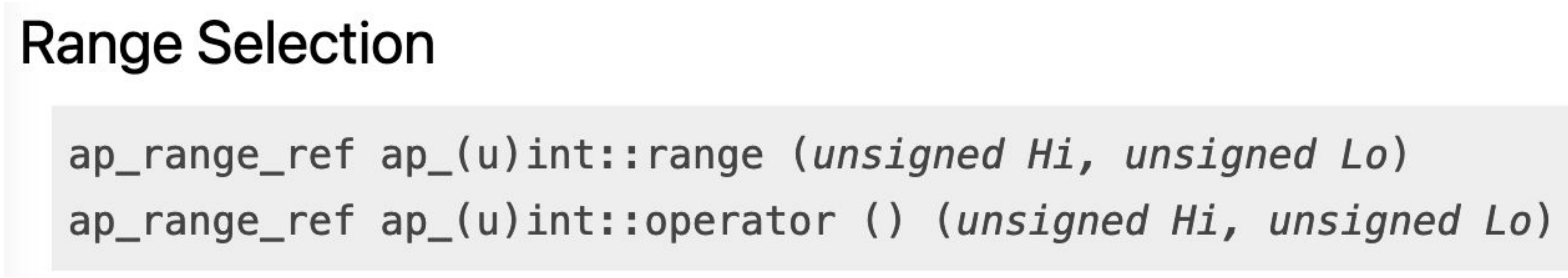
- Use maximum bus-width available

- In theoretical, if we can proceed 512-bit, we will improve performance 16 times faster

Demo

Increase from 32-bit to 512-bit

In summary, we can use two syntaxes to read data:



Range Selection

```
ap_range_ref ap_(u)int::range (unsigned Hi, unsigned Lo)
ap_range_ref ap_(u)int::operator () (unsigned Hi, unsigned Lo)
```

Increase from 32-bit to 512-bit

We can select bit range by () or range():

```
ap_uint<4> Rslt;

ap_uint<8> Val1 = 0x5f;
ap_uint<8> Val2 = 0xaa;

Rslt = Val1.range(3, 0); // Yields: 0xF
Val1(3,0) = Val2(3, 0); // Yields: 0x5A
Val1(3,0) = Val2(4, 1); // Yields: 0x55
Rslt = Val1.range(4, 7); // Yields: 0xA; bit-reversed!
```

Increase from 32-bit to 512-bit

- Handle bit-range selection is sometimes confusing

- We can use two dimensional array to represent wider bus-width with #pragma ARRAY_RESHAPE

Increase from 32-bit to 512-bit

- Handle bit-range selection is sometimes confusing

- We can use two dimensional array to represent wider bus-width with #pragma ARRAY_RESHAPE

Multi-dimensional Array wider bus-width

Demo

#pragma HLS ARRAY_RESHAPE

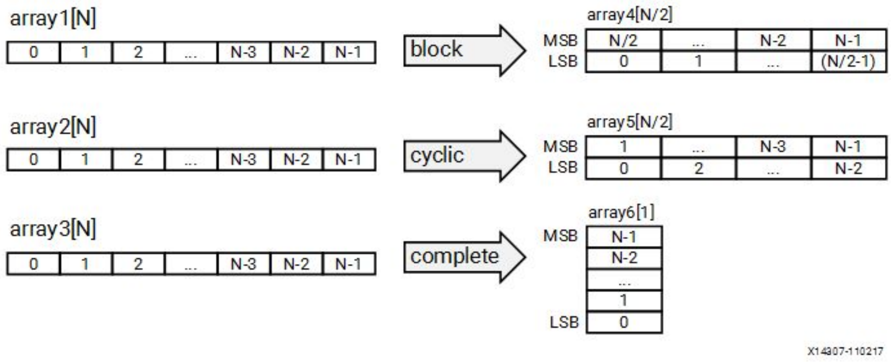
factor=2

Left side:

- Spent N clock cycles to read N elements in array

Right side:

- **Block**: 1 clock cycle read 2 elements **distance by N/2**



#pragma HLS ARRAY_RESHAPE

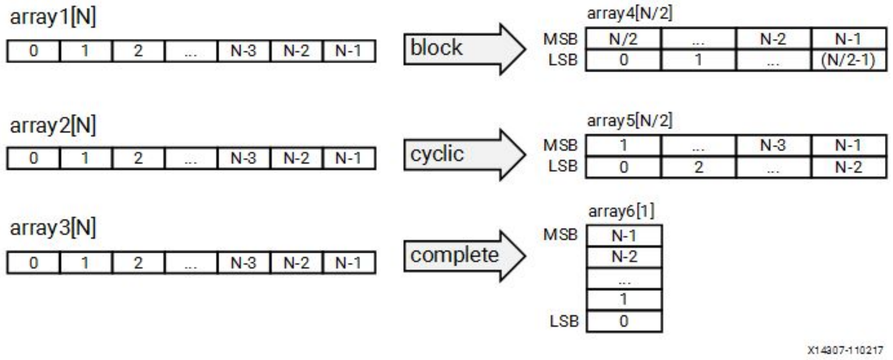
factor=2

Left side:

- Spent N clock cycles to read N elements in array

Right side:

- **Cyclic**: 1 clock cycle read 2 elements **distance by 1**



#pragma HLS ARRAY_RESHAPE

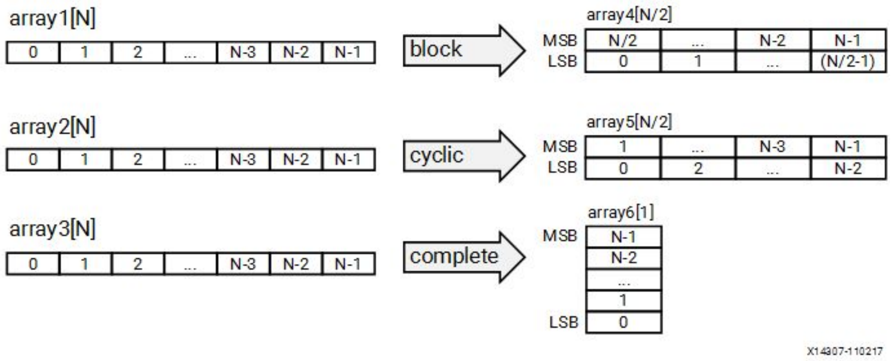
factor=2

Left side:

- Spent N clock cycles to read N elements in array

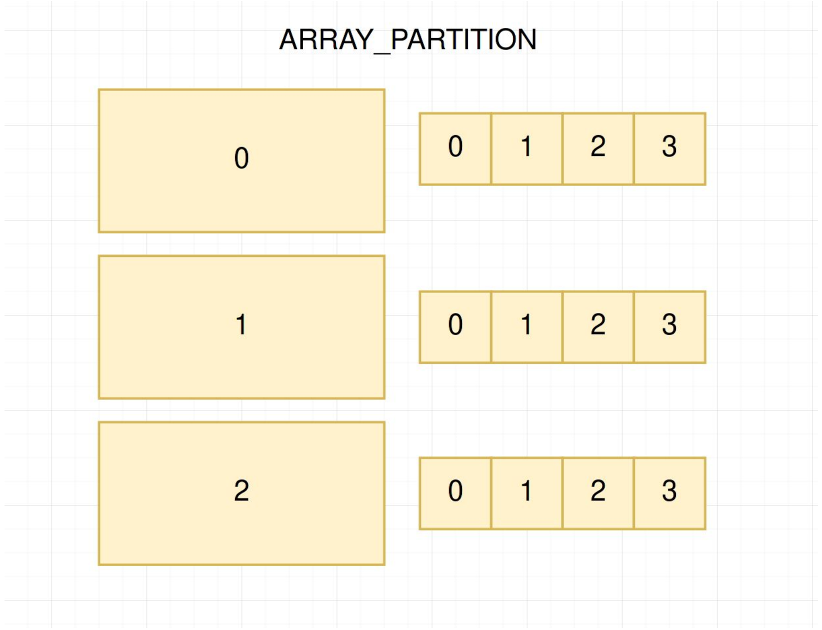
Right side:

- **Complete**: 1 clock cycle read **all** elements



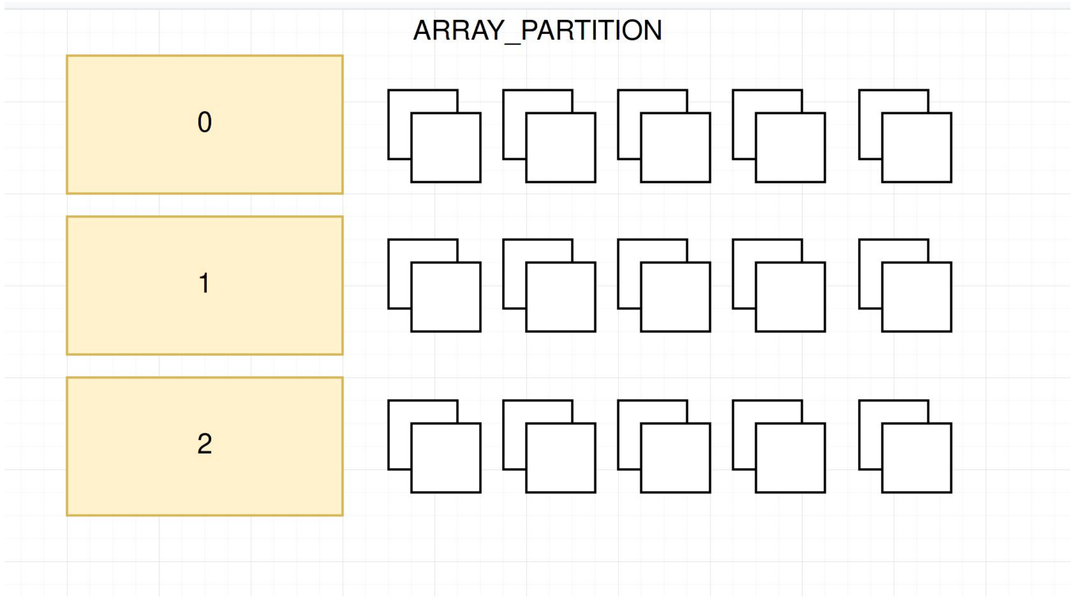
#pragma HLS ARRAY_PARITION

- Similar to ARRAY_RESHAPE
- Easier to control multi-dimensional array



#pragma HLS ARRAY_PARITION

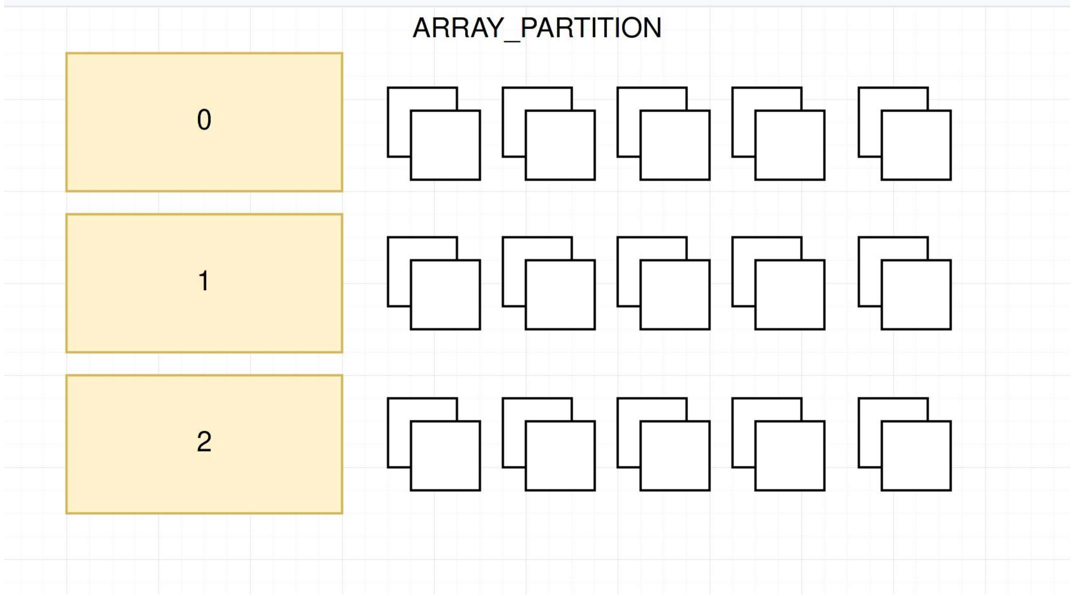
- Similar to ARRAY_RESHAPE
- Easier to control multi-dimensional array



ARRAY_PARITION vs ARRAY_RESHAPE

Two pragma are very similar, when to use what ?

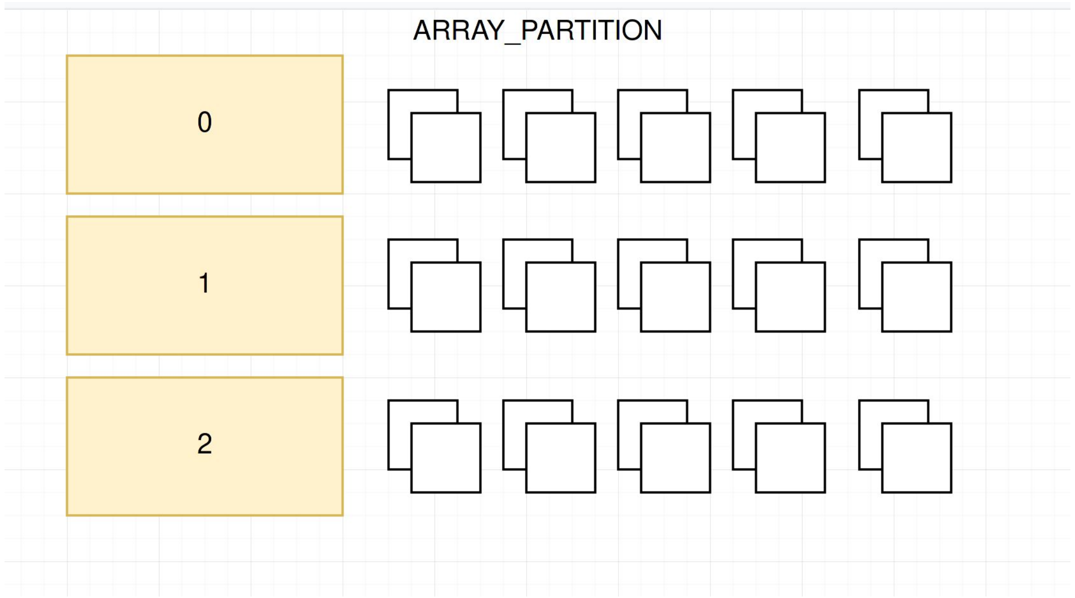
- ARRAY_RESHAPE for **single** dimensional array



ARRAY_PARITION vs ARRAY_RESHAPE

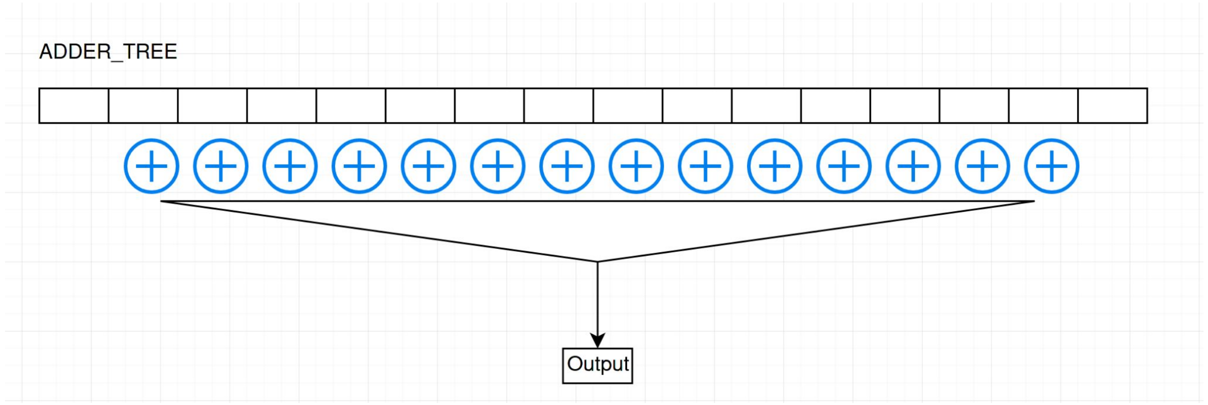
Two pragma are very similar, when to use what ?

- ARRAY_RESHAPE for **single** dimensional array
- ARRAY_PARITION for **multi-**dimensional array

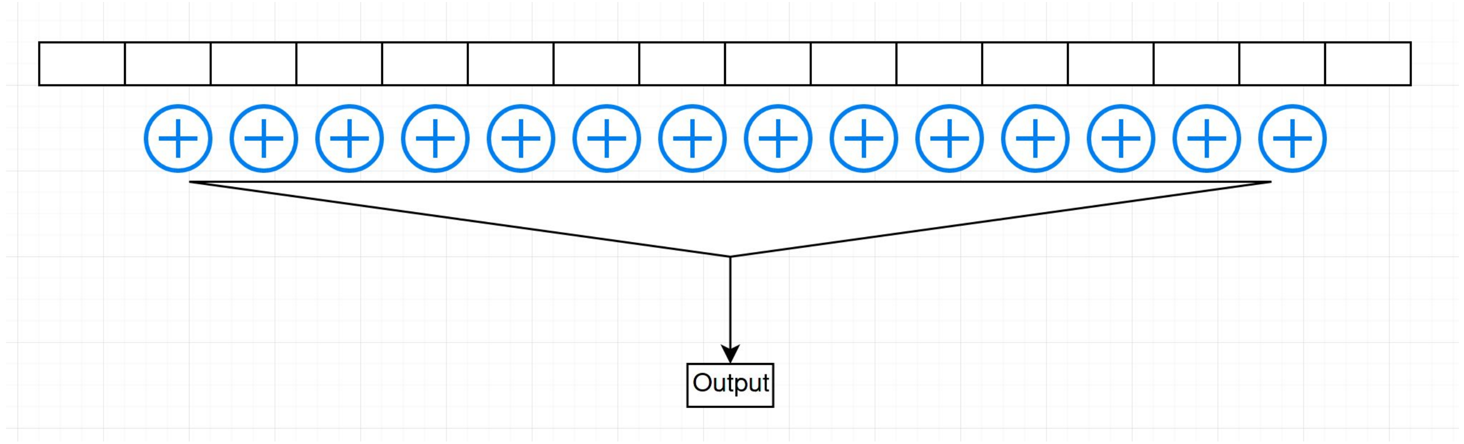


Adder Tree

What we have



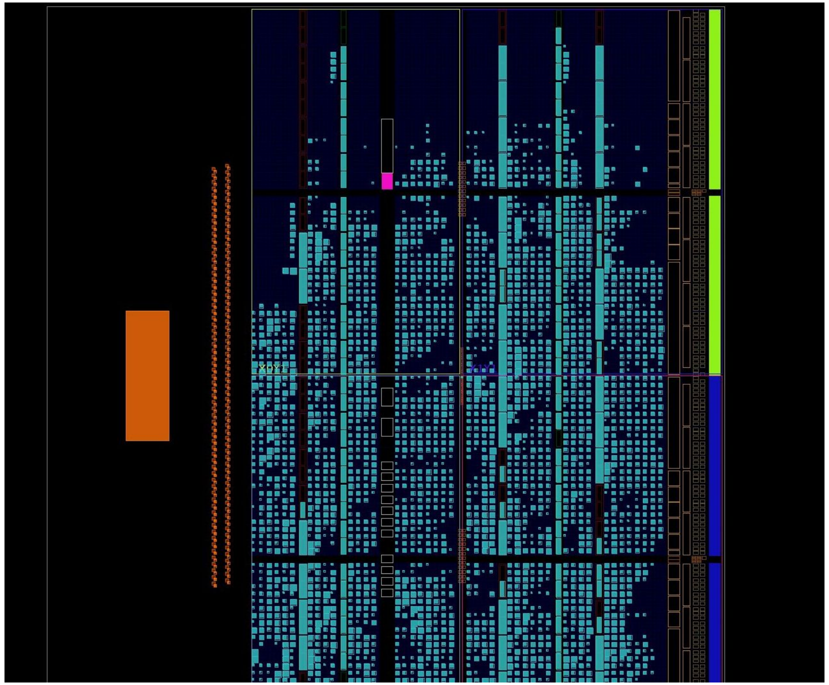
Fan-out Adder Tree



Huge adder tree decrease performance

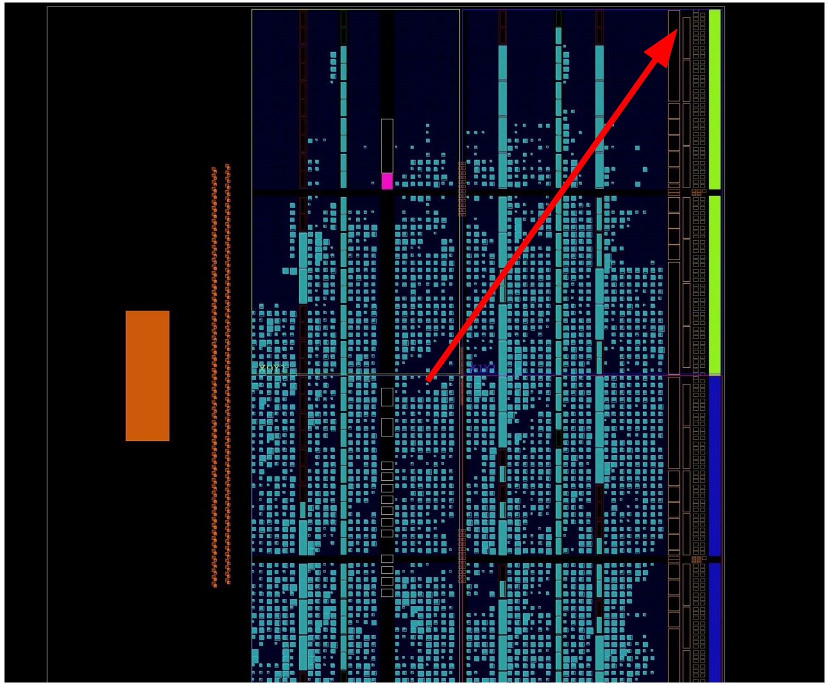
Fanout Problem

- Long critical path



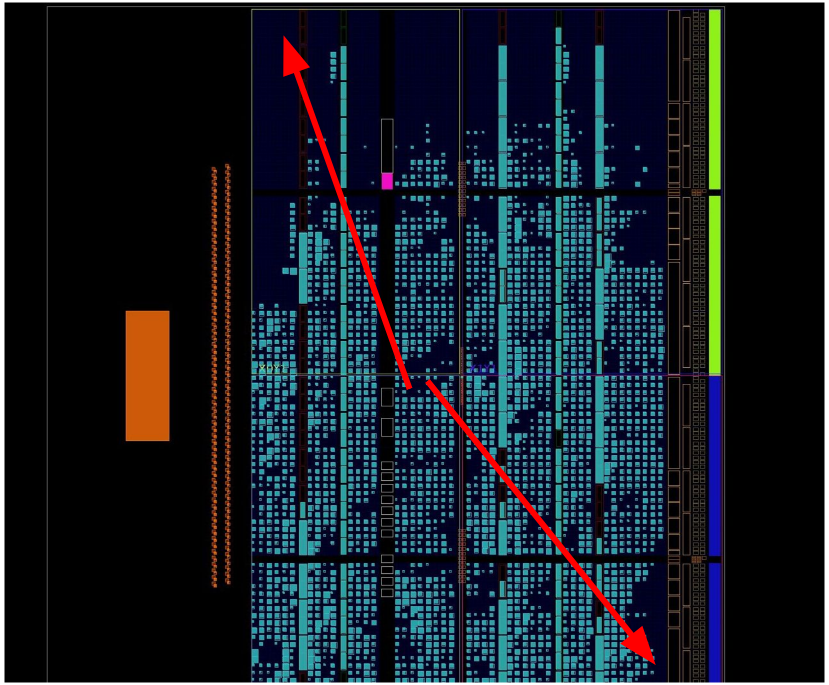
Fanout Problem

- Long critical path



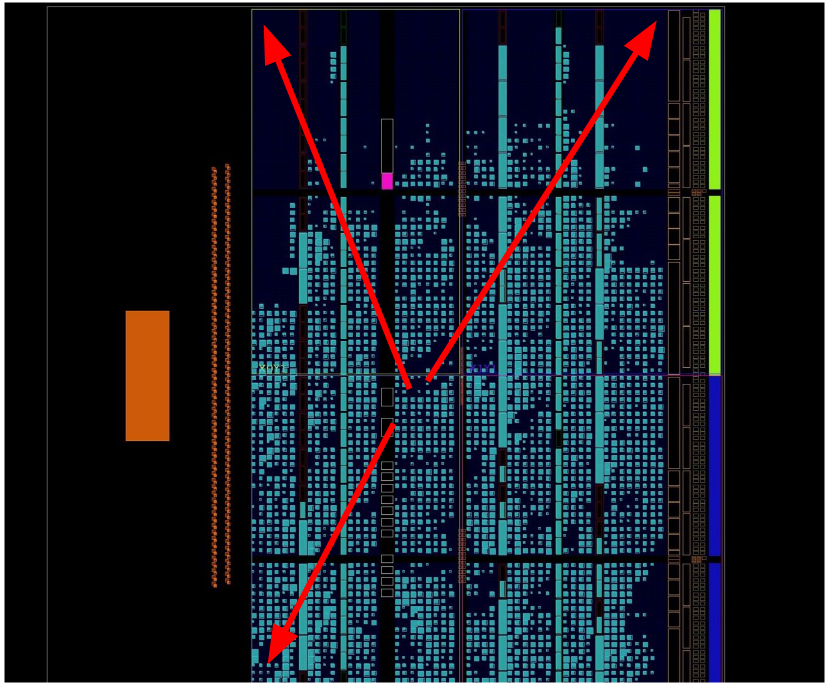
Fanout Problem

- Long critical path



Fanout Problem

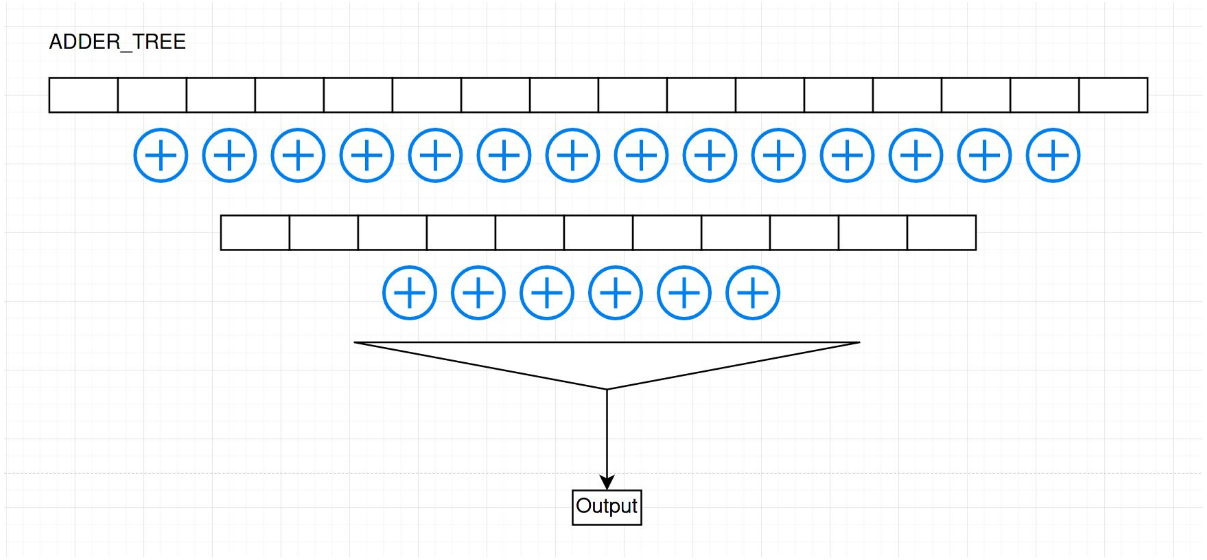
- Long critical path



Adder Tree

What we want:

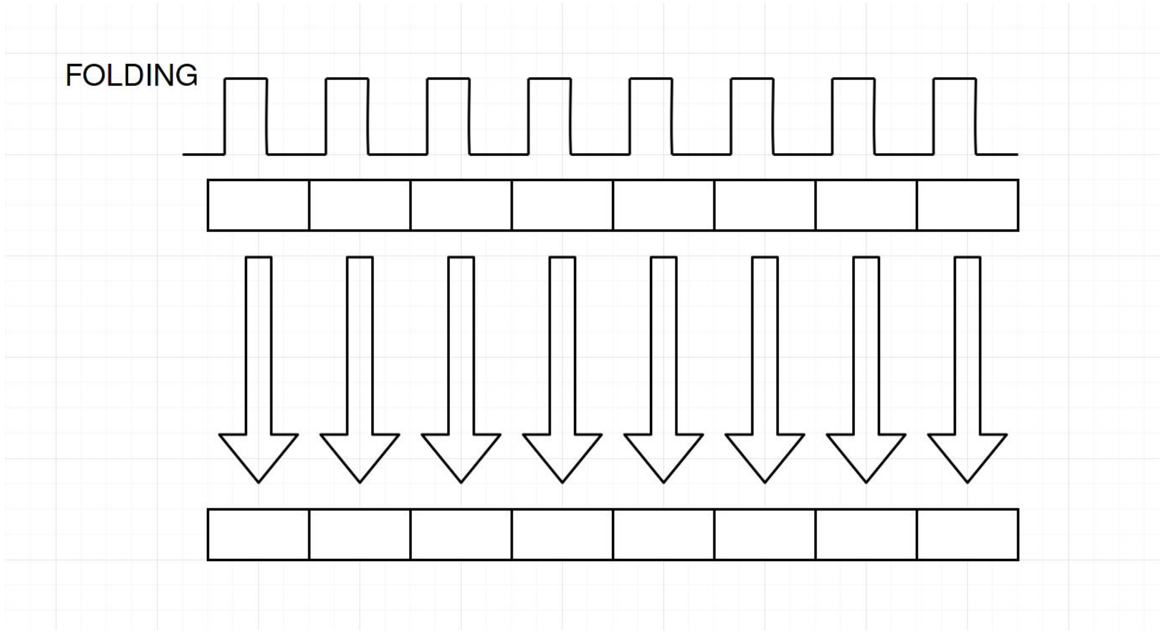
Multiple layer of adder tree



Partial UNROLL or “Folding”

Simple explanation:

1. Chop big loop to smaller loop



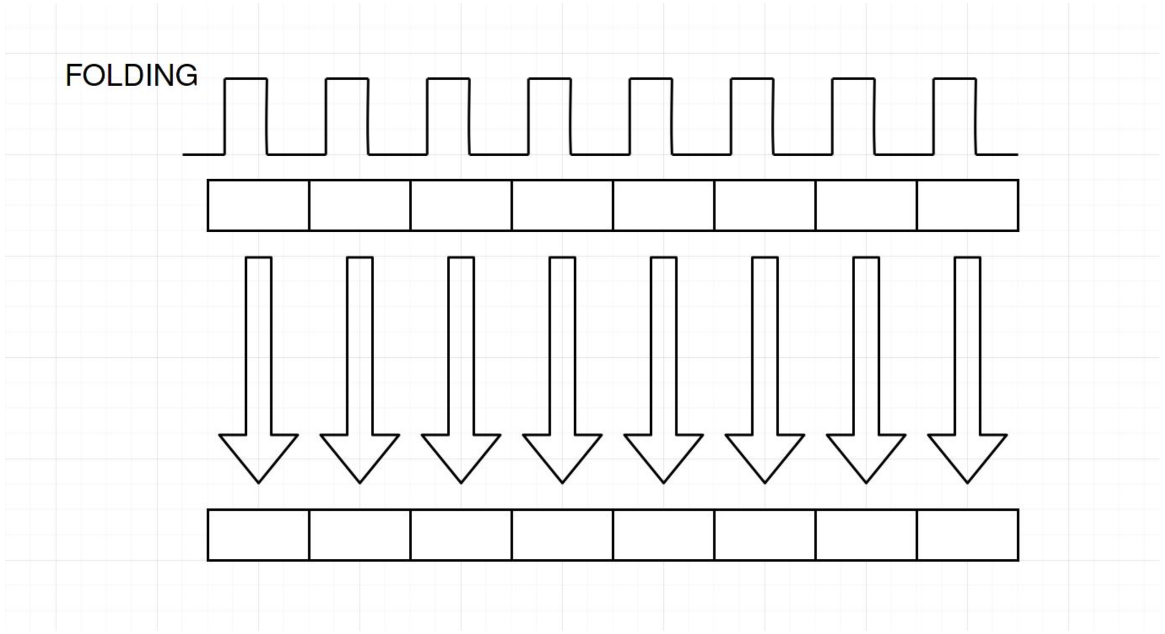
Partial UNROLL or “Folding”

Simple explanation:

1. Chop big loop to smaller loop

eh…?

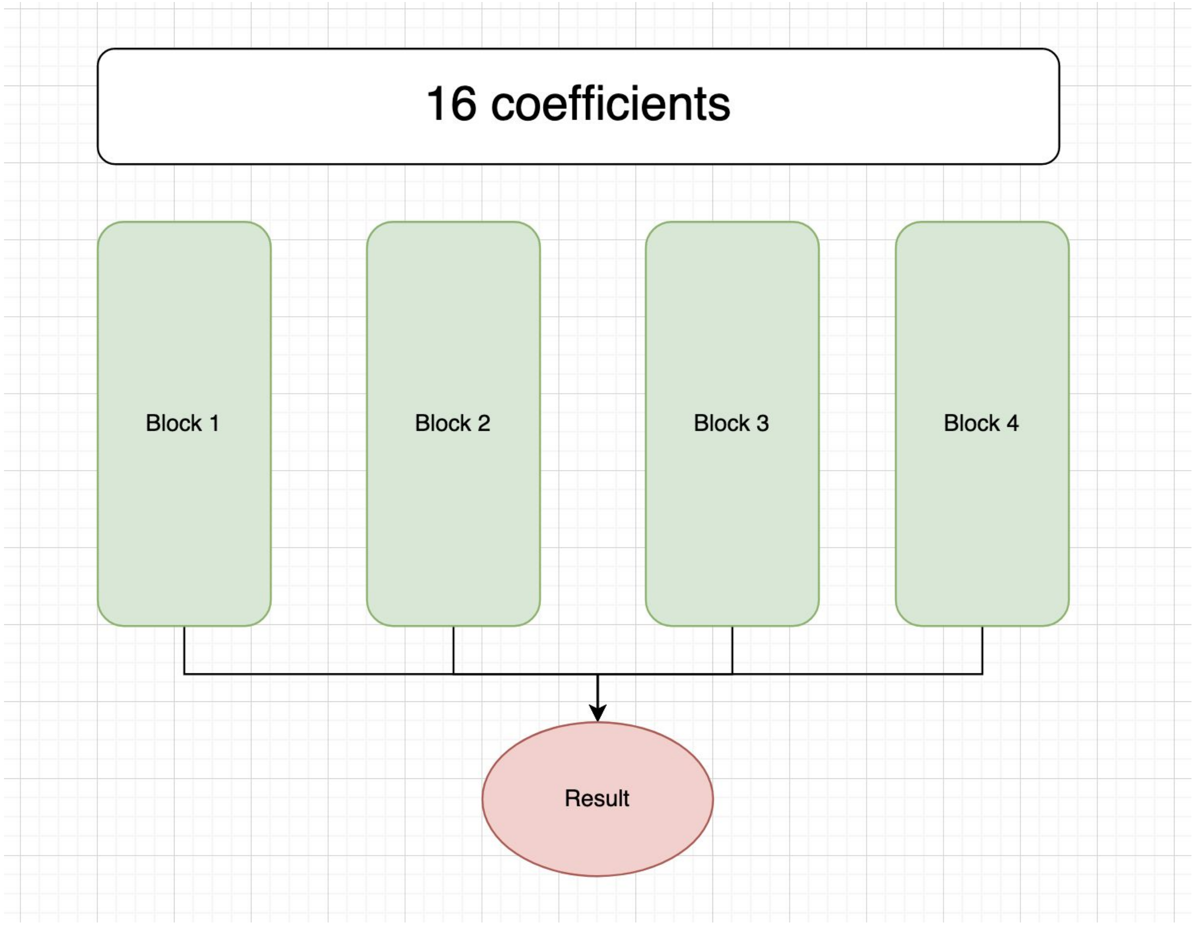
2. Chop Chop Chop



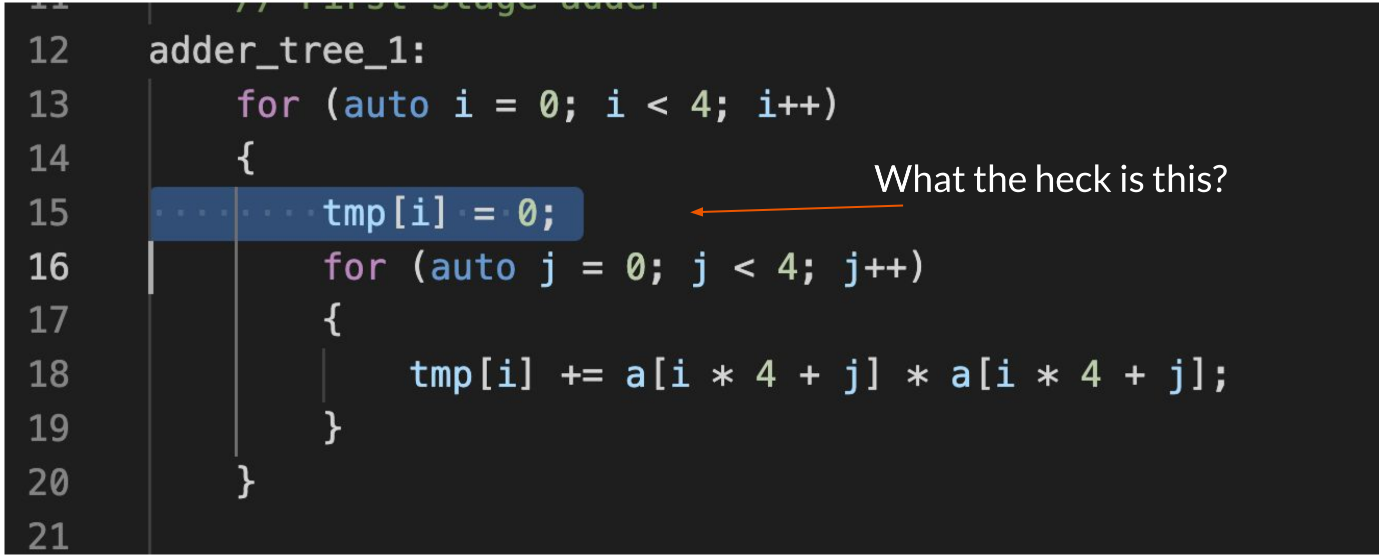
Folding Demo

Demo

-



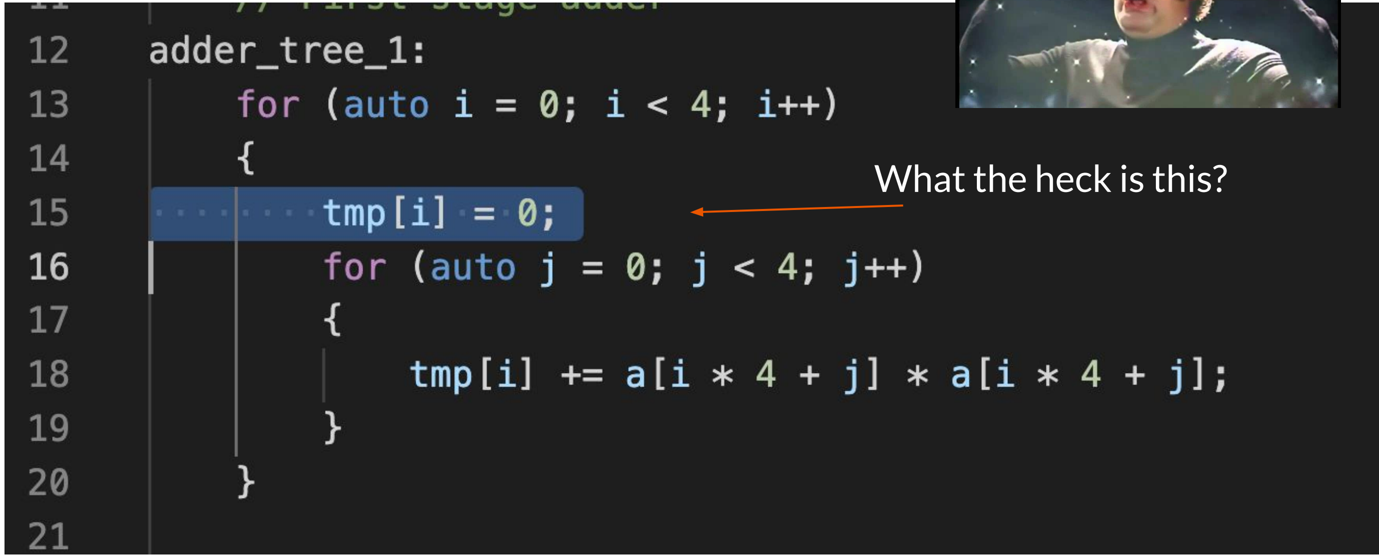
Decrease loop performance



```
    // First stage adder
adder_tree_1:
    for (auto i = 0; i < 4; i++)
    {
        tmp[i] = 0;
        for (auto j = 0; j < 4; j++)
        {
            tmp[i] += a[i * 4 + j] * a[i * 4 + j];
        }
    }
```

What the heck is this?

Decrease loop performance



```
adder_tree_1:
    for (auto i = 0; i < 4; i++)
    {
        tmp[i] = 0;
        for (auto j = 0; j < 4; j++)
        {
            tmp[i] += a[i * 4 + j] * a[i * 4 + j];
        }
    }
```

What the heck is this?

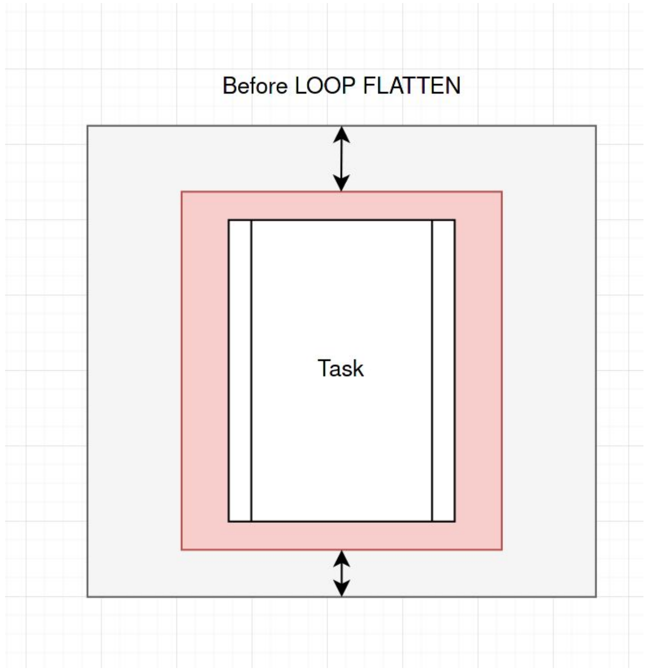
#pragma HLS LOOP_FLATTEN

Either:

- In every iteration of the **outer loop**, it spends 1 cycles to jump into **inner loop**

Or

- After **inner loop** finished, it spends 1 cycles to jump to **outer loop**



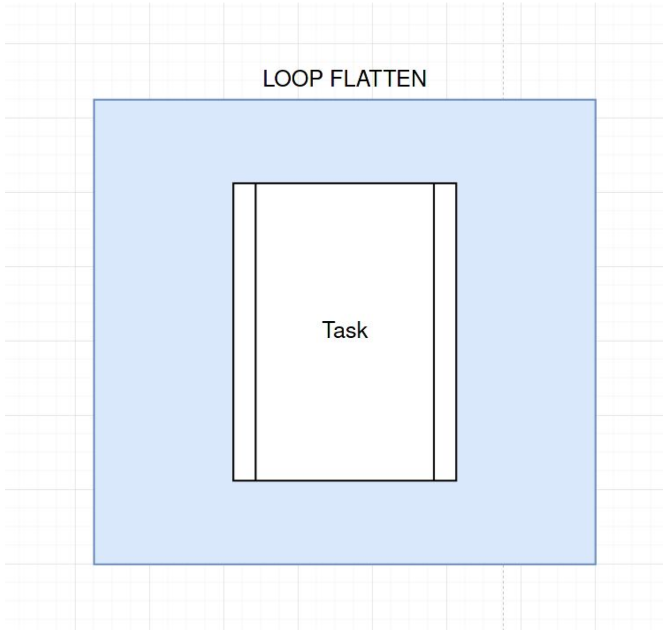
#pragma HLS LOOP_FLATTEN

Either:

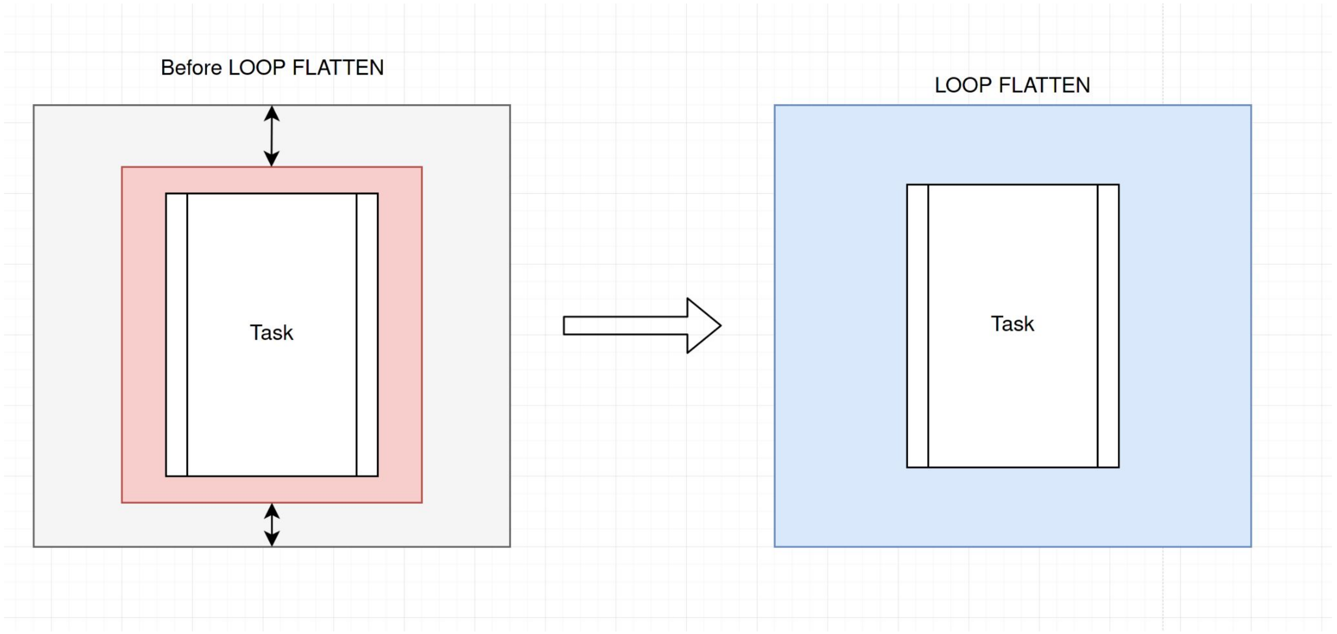
- In every iteration of the **outer loop**, it spends 1 cycles to jump into **inner loop**

Or

- After **inner loop** finished, it spends 1 cycles to jump to **outer loop**



#pragma HLS LOOP_FLATTEN



Should we assign value outside the loop?

- If you assign value at the outer loop, you have to spend **1 cycles** to get to the inner loop.
- If you assign value at the initialization, you rely on **HLS compiler** to merge the initialization to the loop.

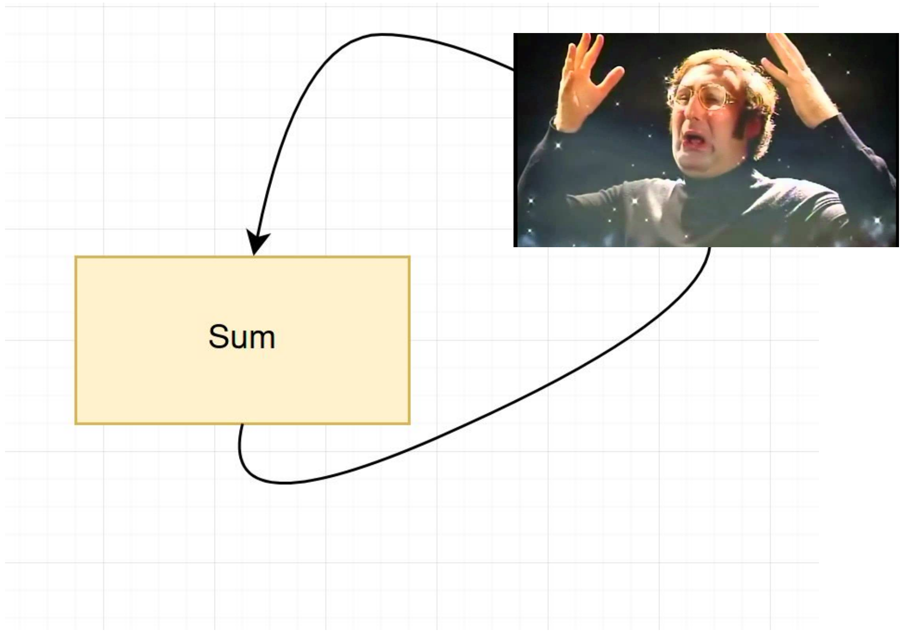
We can do better.

Demo

Violation

Resolve ADD Violation

Demo

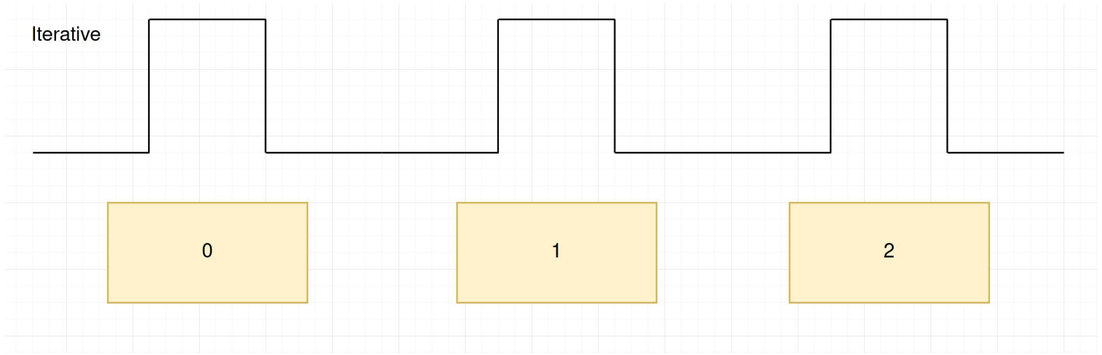


We forgot to handle overflow !!!

Demo

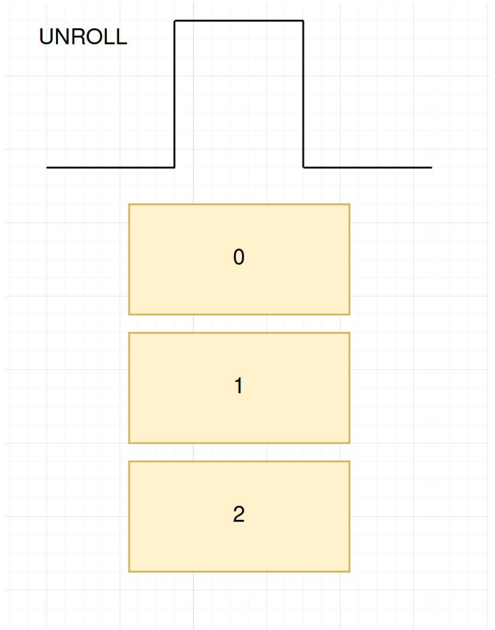
#pragma HLS UNROLL

Motivation



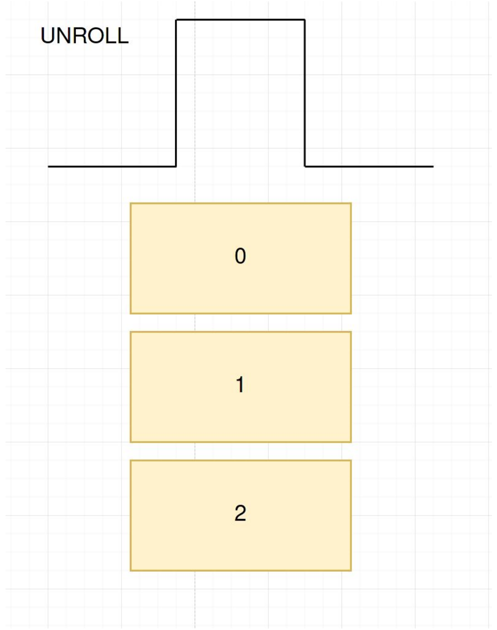
#pragma HLS UNROLL

- Parallel execution in **single** clock cycle
- Maximum performance



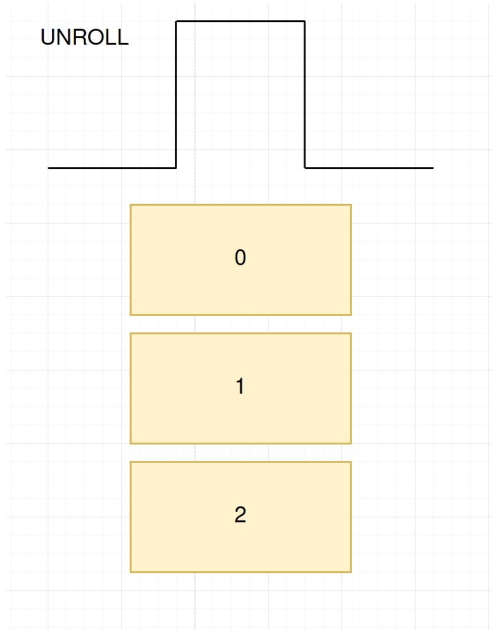
Problem with #pragma HLS UNROLL

- Fan-out
- Decrease frequency significantly
- Resources blow up

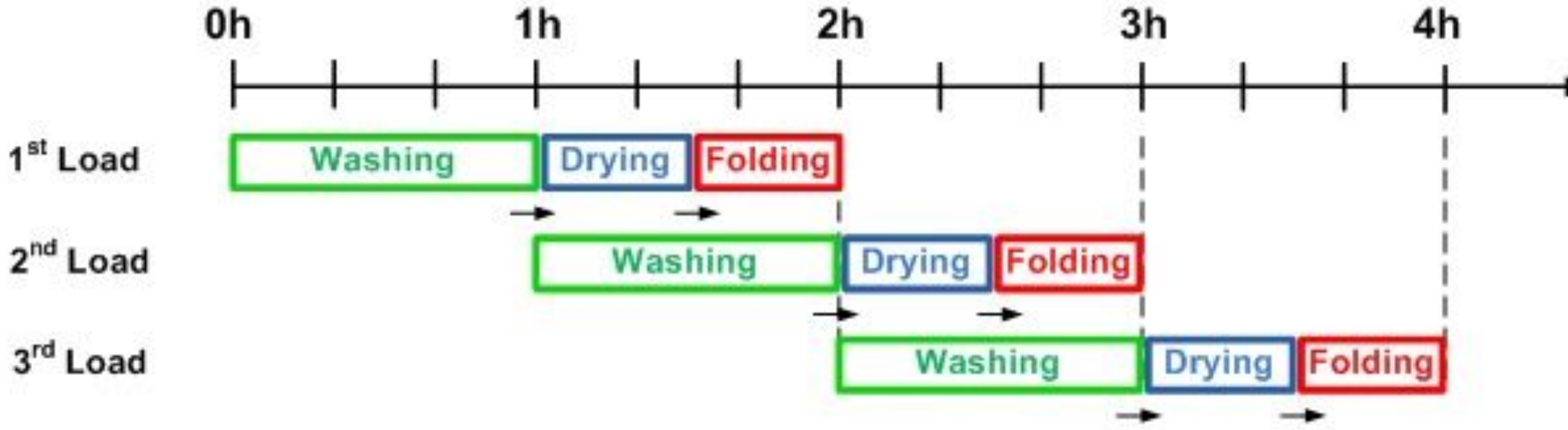


#pragma HLS UNROLL

Demo

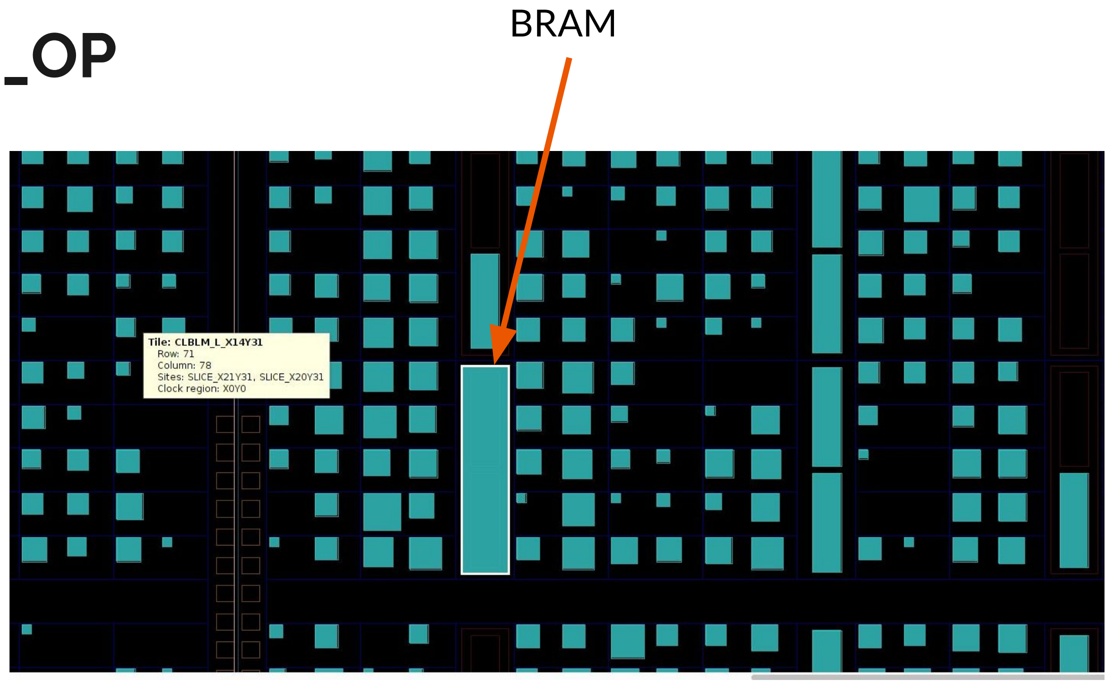


#pragma HLS PIPELINE



#pragma HLS BIND_OP

Pin specified operation to premade fabric: DSP, BRAM, URAM, etc...

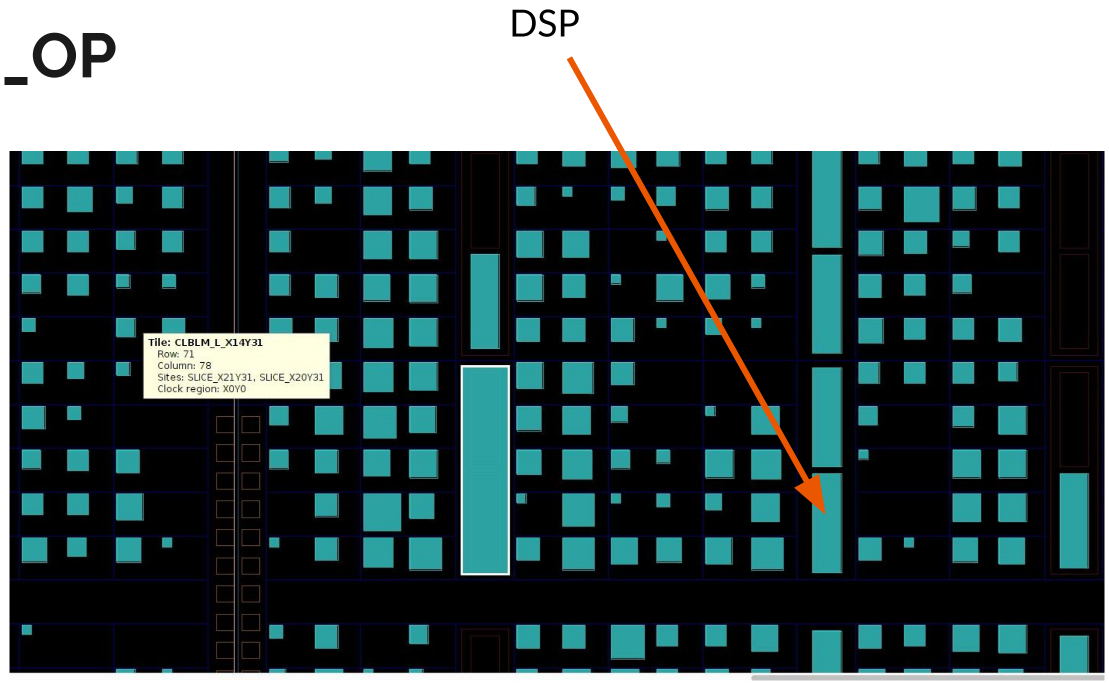


Tile: CLBLM_L_X14Y31
Row: 71
Column: 78
Sites: SLICE_X21Y31, SLICE_X20Y31
Clock region: X0Y0

#pragma HLS BIND_OP

Pin specified operation to premade fabric: DSP, BRAM, URAM, etc...

Force Pipeline inside fabric with specified latency (auto is good enough)



```
Tile: CLBLM_L_X14Y31
  Row: 71
  Column: 78
  Sites: SLICE_X21Y31, SLICE_X20Y31
  Clock region: X0Y0
```

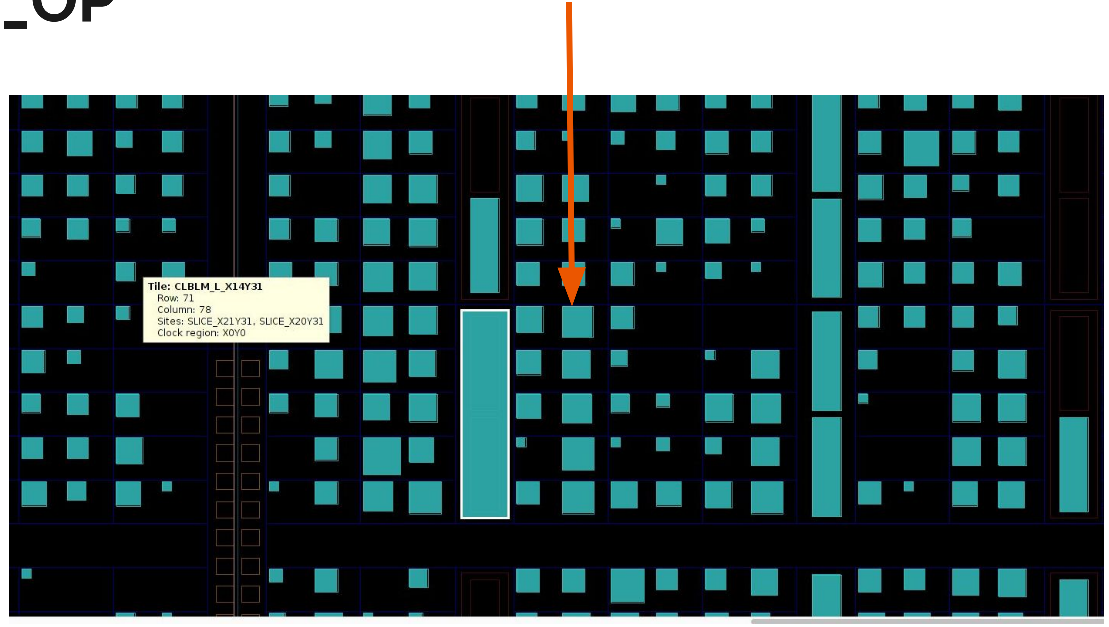
#pragma HLS BIND_OP

CLB ? Who cares

Pin specified operation to premade fabric: DSP, BRAM, URAM, etc…

Force Pipeline inside fabric with specified latency (auto is good enough)

Boost maximum frequency



Tile: CLBLM_L_X14Y31
Row: 71
Column: 78
Sites: SLICE_X21Y31, SLICE_X20Y31
Clock region: X0Y0

#pragma HLS BIND_OP

Pin specified operation to premade fabric: DSP, BRAM, URAM, etc…

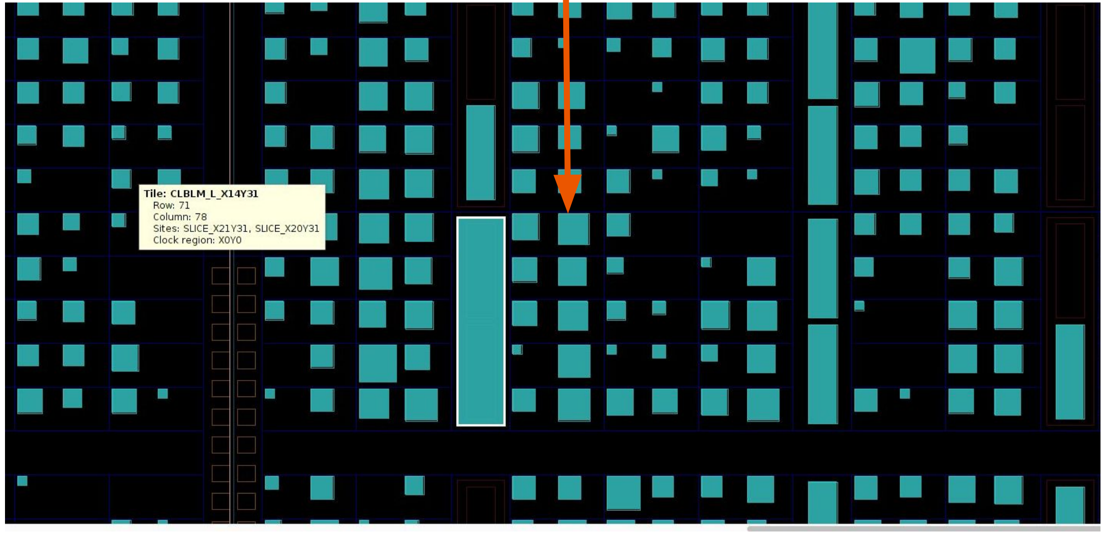
Force Pipeline inside fabric with specified latency (auto is good enough)

Boost maximum frequency

Demo

- C synthesis

CLB ? Who cares



Coding Conclusion

- If you are looking at lightweight implementation: choose HDL
- HLS is an abstract layer over HDL
- HLS can perform complex operation as HDL

- Pr with HLS is easy
- Testing with HLS is quick
- Co-simulation with HLS is quick

Performance Conclusion

- HLS can achieve good performance.
- CERG GMU demonstrates HLS can achieve similar latency, frequency as RTL.
  But HLS utilizes more resources compare to RTL.
- The problem with FPGA in HPC nowadays is not to minimize LUT, FF usage.
  It is to achieve **maximum performance**, utilize all available resources on FPGA.

Performance Conclusion

- HLS can achieve good performance.
- CERG GMU demonstrates HLS can achieve similar latency, frequency as RTL. HLS utilize more resources compare to RTL.
- The problem with FPGA in HPC nowadays is not to minimize LUT, FF usage. It is to achieve maximum performance, utilize all available resources on FPGA.
- Remember to take the transfer size into account.

- **Prepare input data can give ultimate solution!!!**

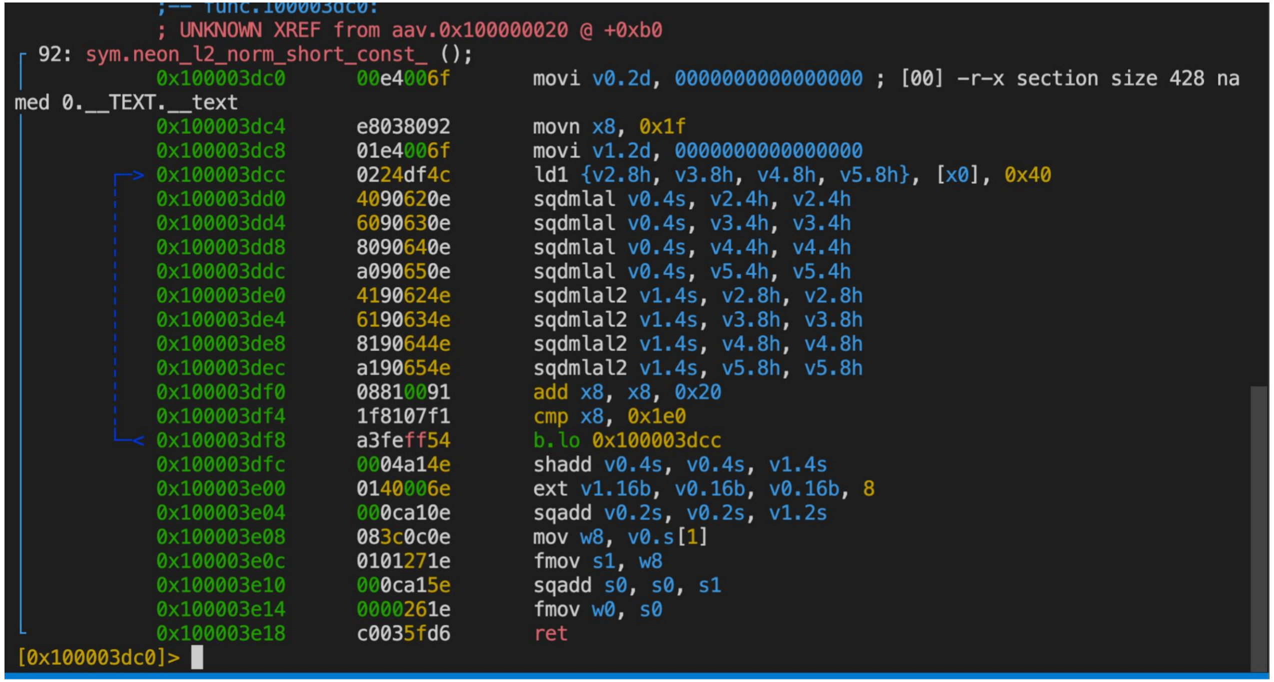
What the point of Hardware accelerator anyway?

- Not mentioned in this talk: **#pragma HLS DATAFLOW**

- Not mentioned in this talk: NEON vs HLS
- Load/Store architecture and Hardware Accelerator

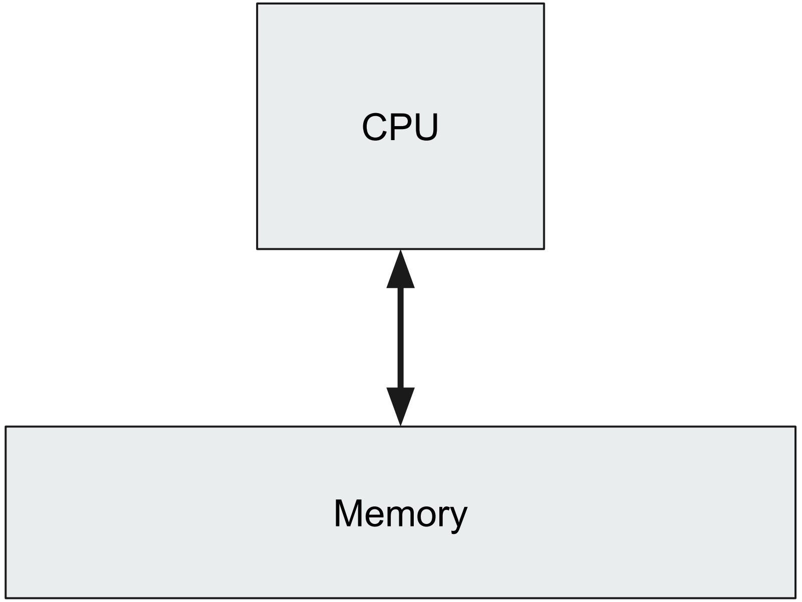
- The code for NEON is ready. It’s up to you to figure it out which one is faster in this case. And let the class know!!!

NEON Implementation

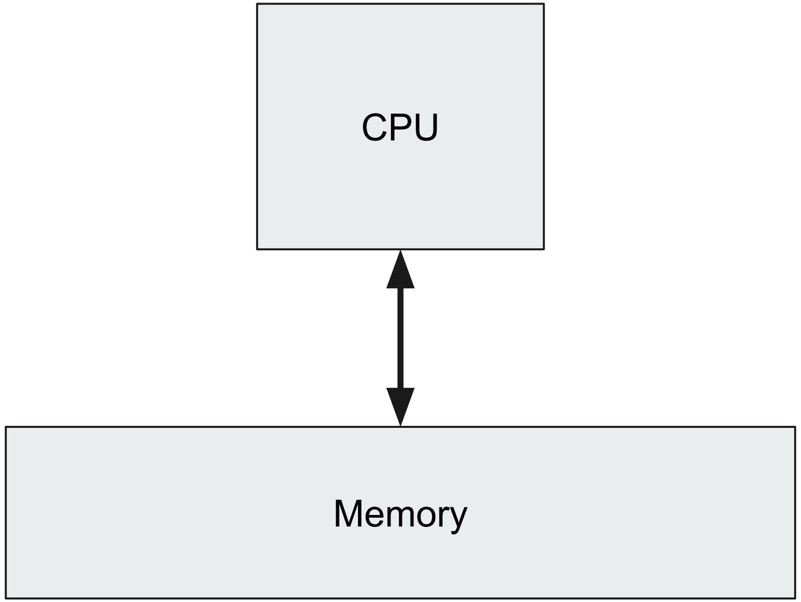


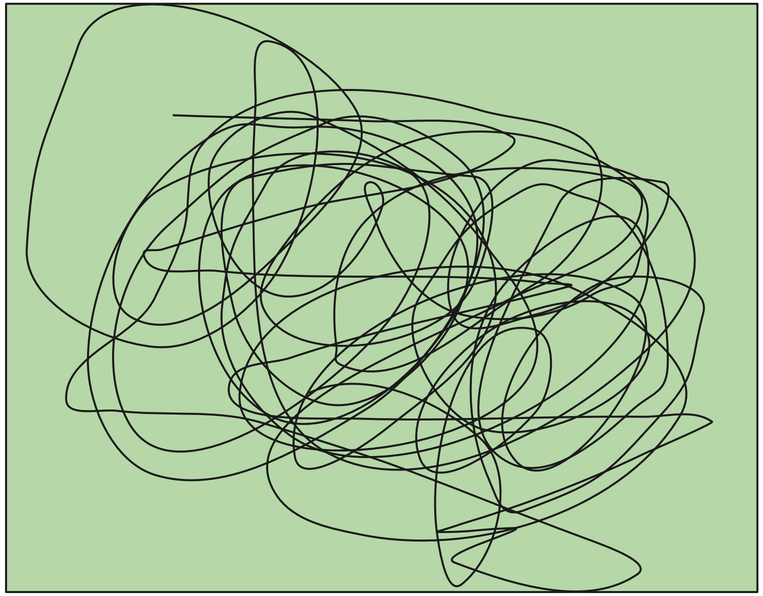
```
                ;-- func.100003dc0:
                ; UNKNOWN XREF from aav.0x100000020 @ +0xb0
┌ 92: sym.neon_l2_norm_short_const_ ();
│               0x100003dc0      00e4006f       movi v0.2d, 0000000000000000 ; [00] -r-x section size 428 na
med 0.__TEXT.__text
│               0x100003dc4      e8038092       movn x8, 0x1f
│               0x100003dc8      01e4006f       movi v1.2d, 0000000000000000
│          ┌──> 0x100003dcc      0224df4c       ld1 {v2.8h, v3.8h, v4.8h, v5.8h}, [x0], 0x40
│          ╎    0x100003dd0      4090620e       sqdmlal v0.4s, v2.4h, v2.4h
│          ╎    0x100003dd4      6090630e       sqdmlal v0.4s, v3.4h, v3.4h
│          ╎    0x100003dd8      8090640e       sqdmlal v0.4s, v4.4h, v4.4h
│          ╎    0x100003ddc      a090650e       sqdmlal v0.4s, v5.4h, v5.4h
│          ╎    0x100003de0      4190624e       sqdmlal2 v1.4s, v2.8h, v2.8h
│          ╎    0x100003de4      6190634e       sqdmlal2 v1.4s, v3.8h, v3.8h
│          ╎    0x100003de8      8190644e       sqdmlal2 v1.4s, v4.8h, v4.8h
│          ╎    0x100003dec      a190654e       sqdmlal2 v1.4s, v5.8h, v5.8h
│          ╎    0x100003df0      08810091       add x8, x8, 0x20
│          ╎    0x100003df4      1f8107f1       cmp x8, 0x1e0
│          └──< 0x100003df8      a3feff54       b.lo 0x100003dcc
│               0x100003dfc      0004a14e       shadd v0.4s, v0.4s, v1.4s
│               0x100003e00      0140006e       ext v1.16b, v0.16b, v0.16b, 8
│               0x100003e04      000ca10e       sqadd v0.2s, v0.2s, v1.2s
│               0x100003e08      083c0c0e       mov w8, v0.s[1]
│               0x100003e0c      0101271e       fmov s1, w8
│               0x100003e10      000ca15e       sqadd s0, s0, s1
│               0x100003e14      0000261e       fmov w0, s0
└               0x100003e18      c0035fd6       ret
[0x100003dc0]>
```

Load/Store Architecture vs Hardware Accelerator



Load/Store Architecture vs Hardware Accelerator



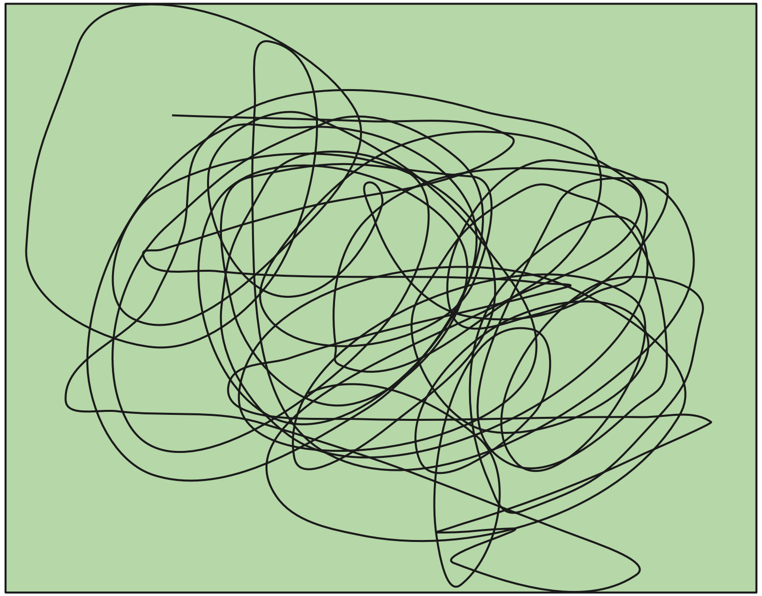


Load/Store Architecture vs Hardware Accelerator

- No pressure on CPU
- No Store and Load

Real world example:

- Video decoder: There are plenty optimized SW implementations. If you don't enable HW, it's give low framerate and high CPU usage.



Thank you

Question ???

Reference

Code used in this talk:

- https://github.com/cothan/Vitis_High_Level_Synthesis_Training
- https://github.com/cothan/Vitis_High_Level_Synthesis_Training_Part_2
-

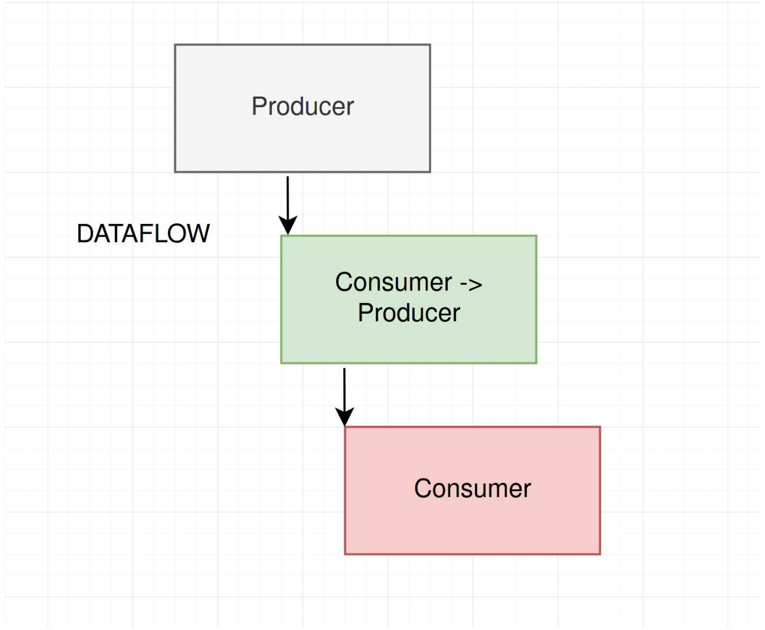
Helpful documents which are not from Xilinx:

- Transformations of High-Level Synthesis Codes for High-Performance Computing

  https://arxiv.org/abs/1805.08288

#pragma HLS DATAFLOW

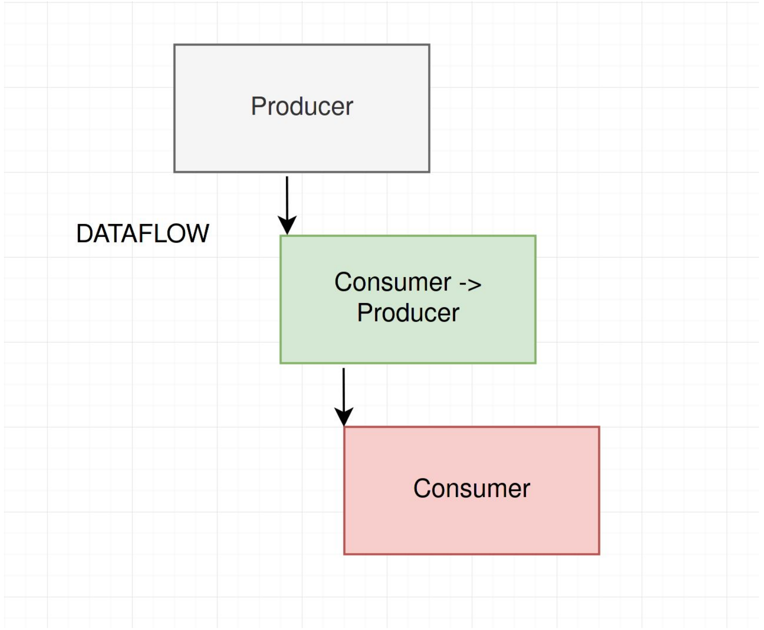
DATAFLOW need to be written in producer-consumer fashion



#pragma HLS DATAFLOW

DATAFLOW need to be written in producer-consumer fashion

May increase or decrease the resources utilization.


Producer
DATAFLOW
Consumer -> Producer
Consumer