CSE 291: FPGA for Computer Vision

Janarbek Matai

Department of Computer Science and Engineering

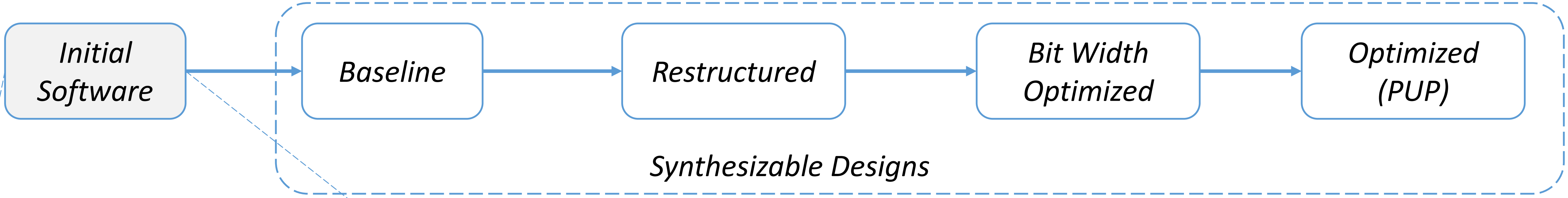
University of California, San Diego

04/20/2017

UCSDCSE
Computer Science and Engineering

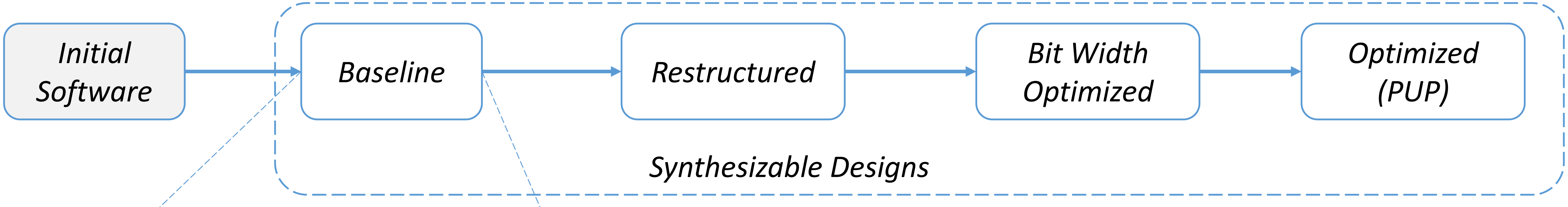
Recap

HLS Design Flow



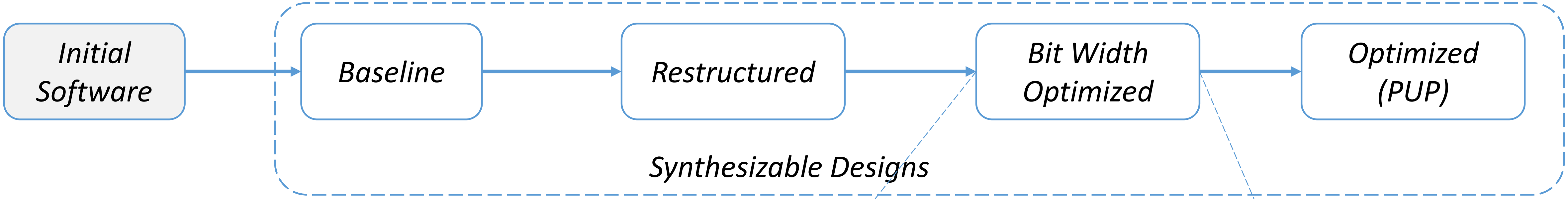

```c
int *array;
array = (int *)malloc(sizeof(int)*N);
for(int i=0;i<N;i++) {

array = … ;
}
```

HLS Design Flow



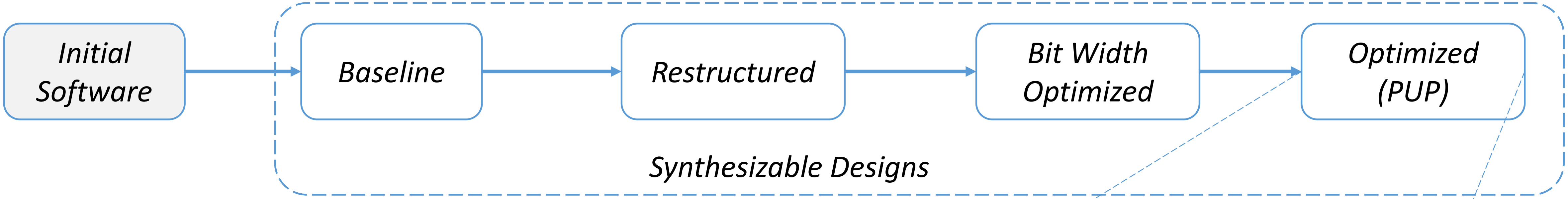

```
int *array;
array[N] ={0};
for(int i=0;i<N;i++) {

array = … ;
}
```

HLS Design Flow



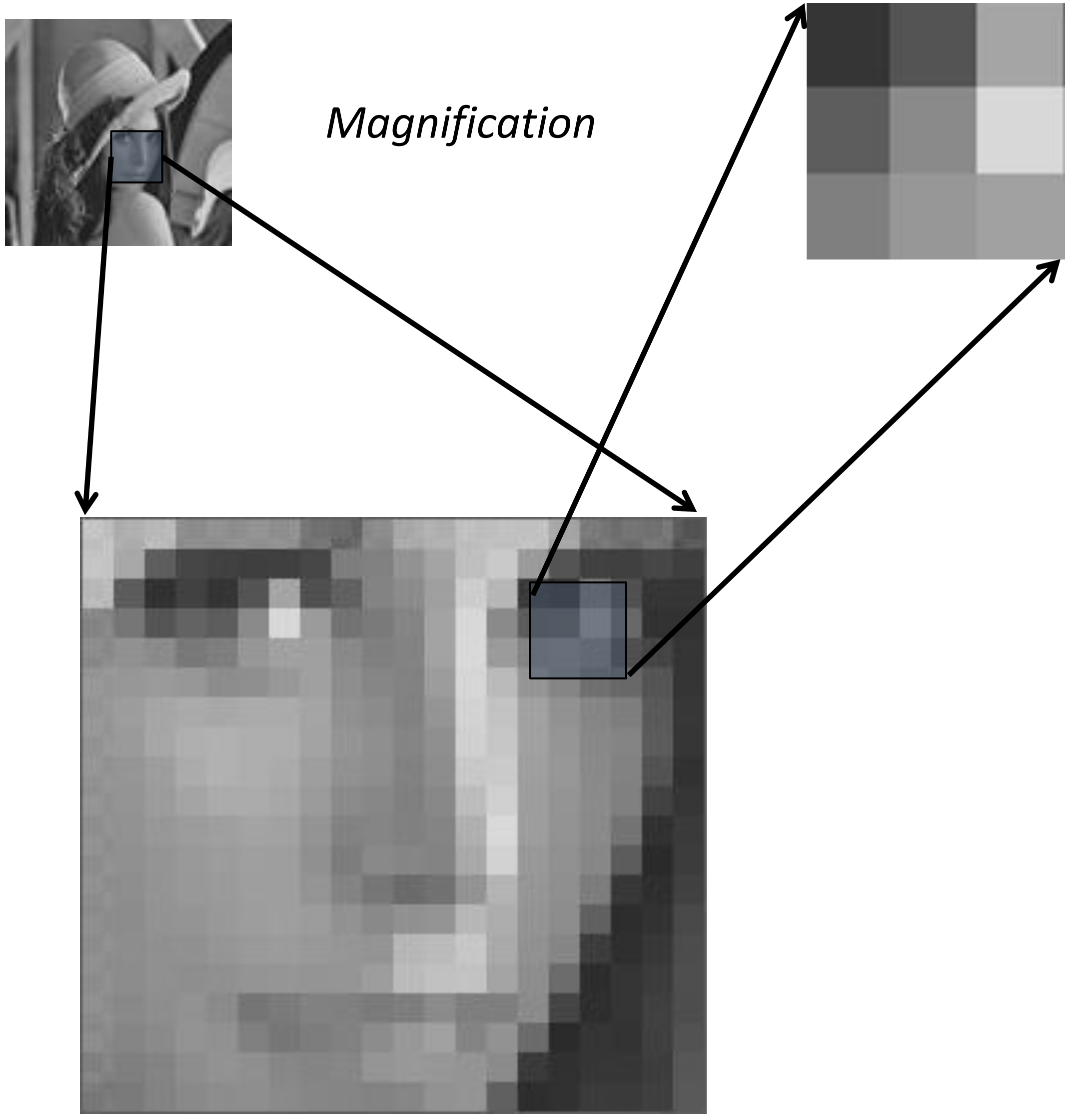

```
ap_int<10> array[N] ={0};
for(int i=0;i<N;i++) {

array = … ;
}
```

HLS Design Flow




```
ap_int<10> array[N] ={0};
for(int i=0;i<N;i++) {
#pragma HLS PIPELINE
array = … ;
}
```

Convolution: Software Sobel Code



-1	0	+1
-2	0	+2
-1	0	+1

+1	+2	+1
0	0	0
-1	-2	-1

G_X = 1(-1) + 2*0 + 3*1 +…+ = 8*

G_Y = 1(-1) + 2*2 +3*1 + … + = -32*

$G = sqrt(G\_X^2 + G\_Y^2)$

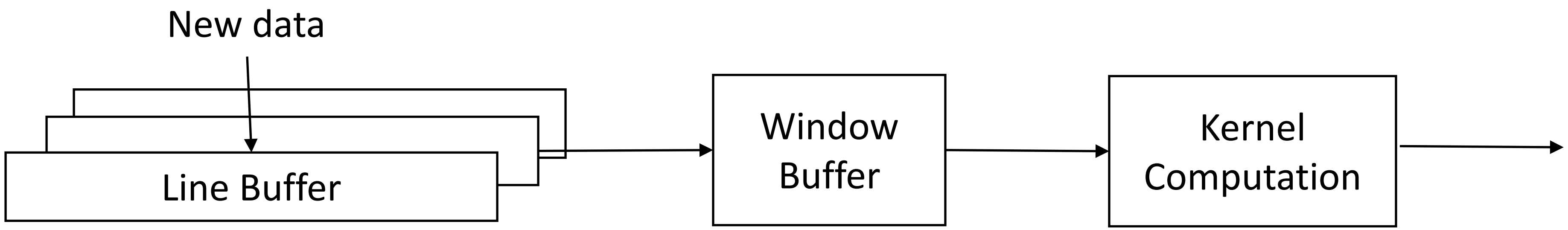
```
for(int i = 0; i < rows; i++){
   for(int j=0; j < cols; j++){
      Gx = 0;
      Gy = 0;
      for(int rowOffset = -1; rowOffset <= 1; rowOffset++){
         for(int colOffset = -1; colOffset <=1; colOffset++){
            Gx = Gx + …;
            Gy = Gy + …;
            G = …;
         }
      }
   }
}
```

Convolution: Software

```
for(int i = 0; i < rows; i++){
   for(int j=0; j < cols; j++){
      Gx = 0;
      Gy = 0;
      for(rowOffset = -1; rowOffset <= 1; rowOffset++){
         for(colOffset = -1; colOffset <=1; colOffset++){
            Gx = Gx + …;
            Gy = Gy + …;
            G = …;
         }
      }
   }
}
```

Based on Xilinx tutorial: "Zynq all programmable soc sobel filter implementation using the vivado hls tool."

Convolution: Hardware Architecture



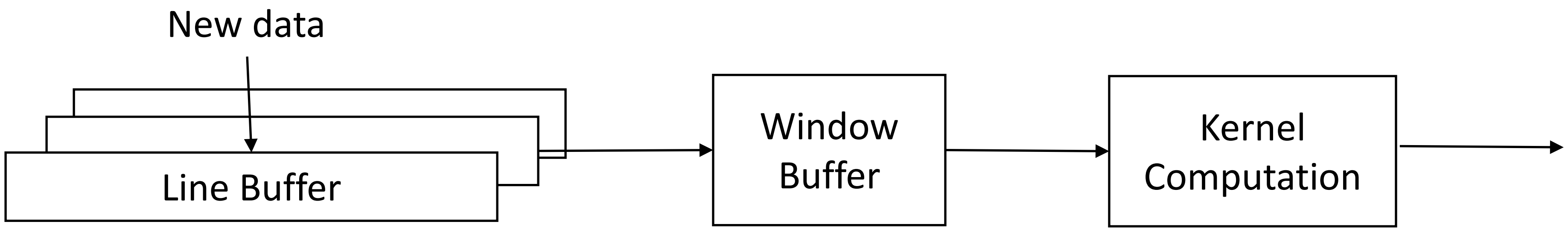
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Input Image

Window Buffer 1

1	2	3
5	6	7
9	10	11

Convolution: Hardware Architecture



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Input Image

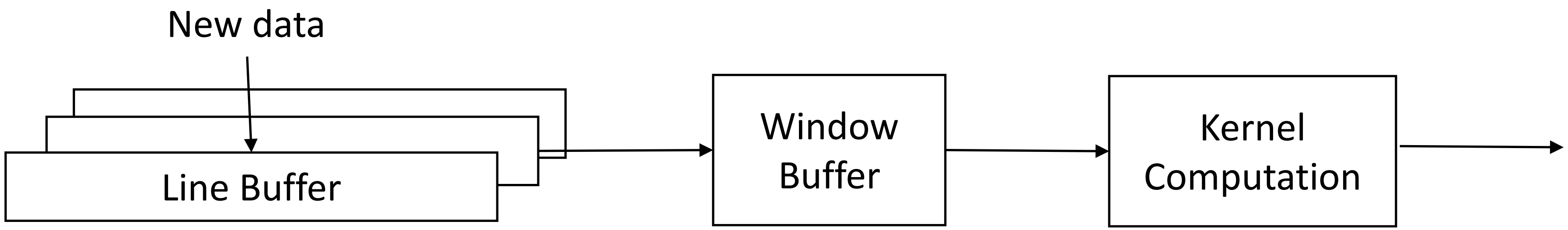
Window Buffer 1

1	2	3
5	6	7
9	10	11

Window Buffer 2

2	3	4
6	7	8
10	11	12

Convolution: Hardware Architecture



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Input Image

Line Buffer at time *t*

Line 1	1	2	3	4
Line 2	5	6	7	8
Line 3	9	10	11	12

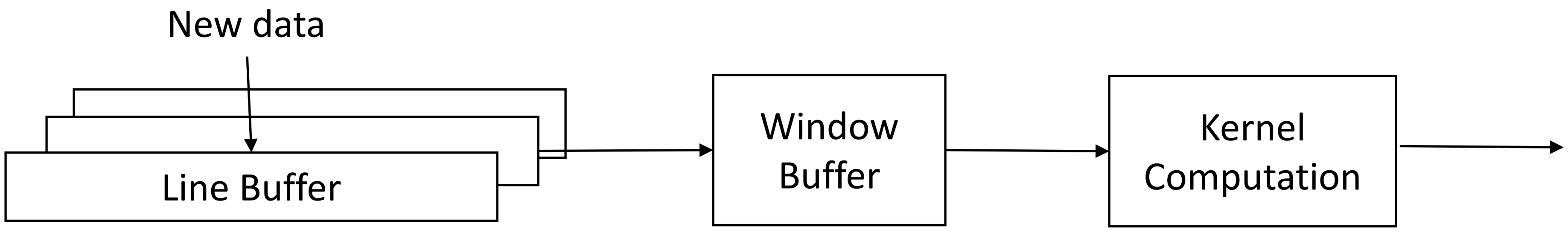
Window Buffer 1

1	2	3
5	6	7
9	10	11

Window Buffer 2

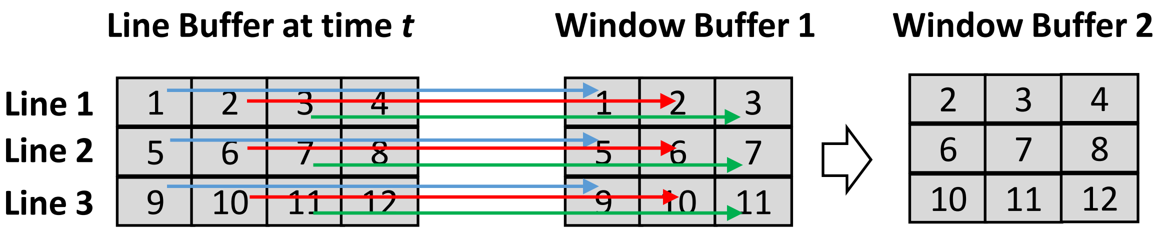
2	3	4
6	7	8
10	11	12

Convolution: Hardware Architecture

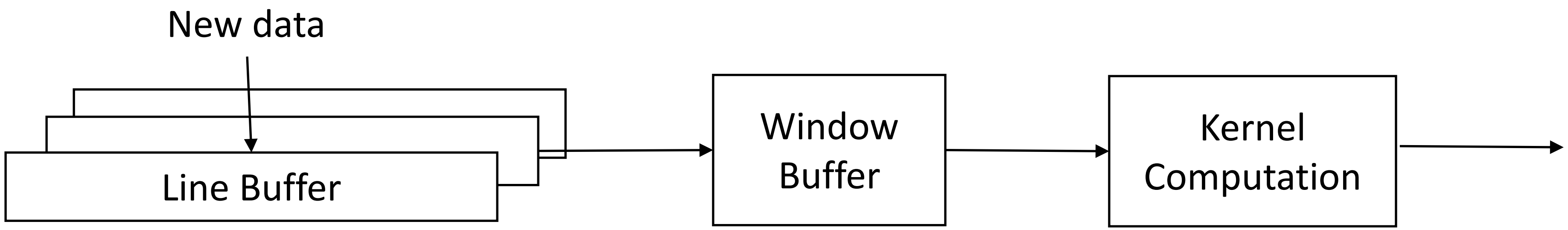


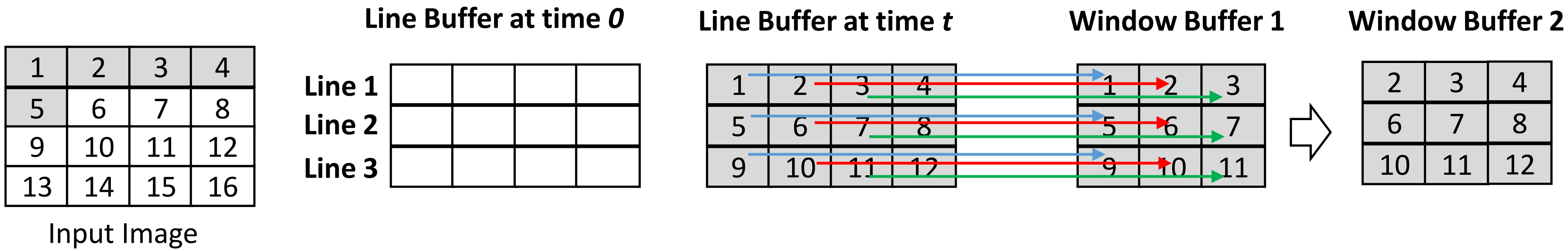
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Input Image

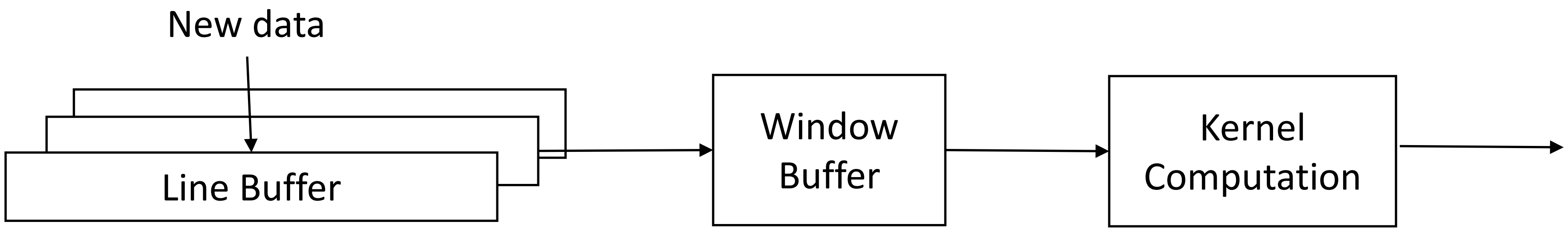


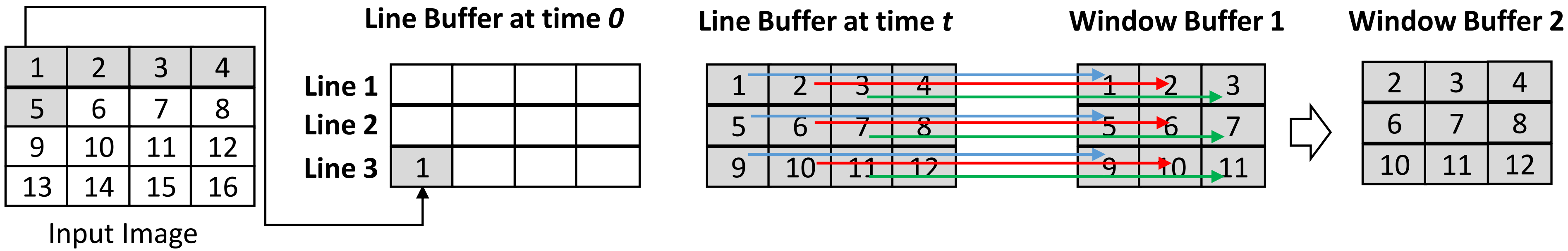
Convolution: Hardware Architecture



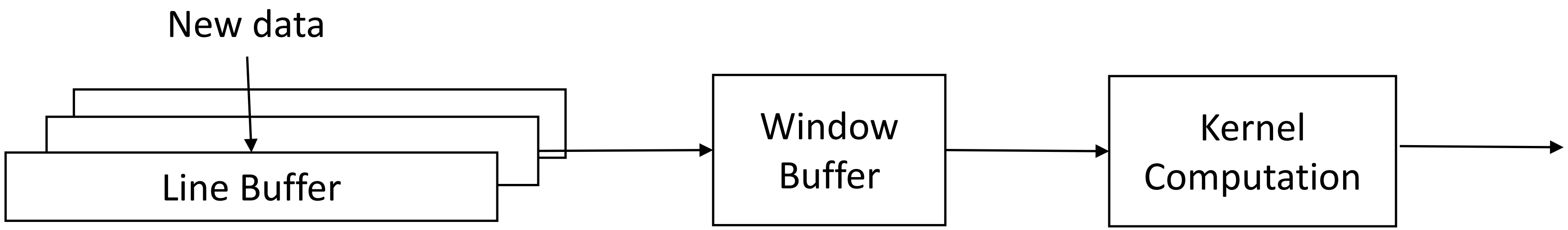


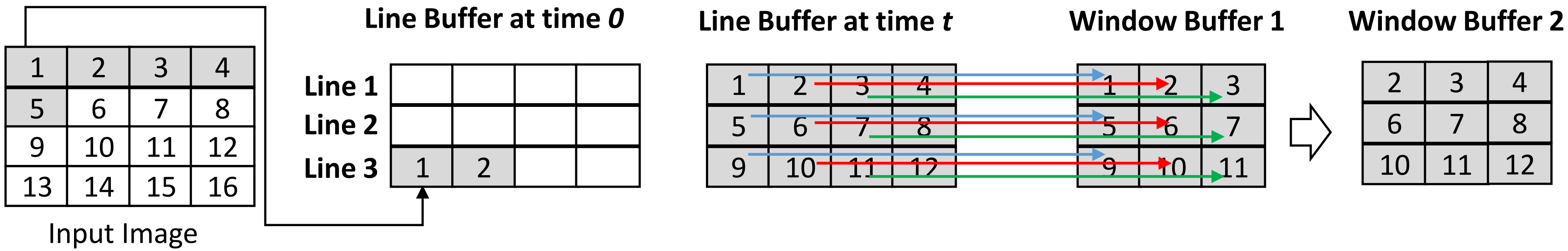
Convolution: Hardware Architecture



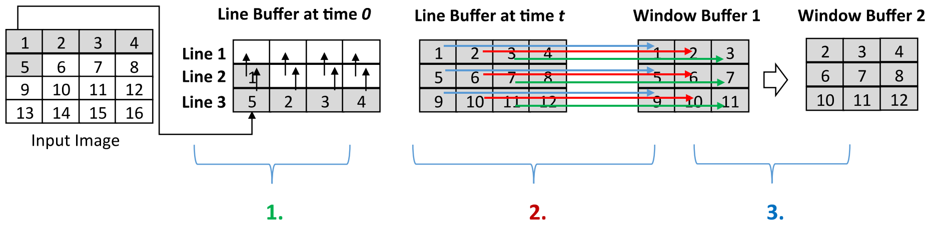


Convolution: Hardware Architecture





Convolution: Hardware




```
sobel_filter (WindowBuffer) {

for(int i=0; i<kernel_size; i++)
   for(int j=0; j<cols; j++)
      #pragma pipeline
     // Do  kernel computation using Window Buffer
      result = ,...
    }
}
return result;
```

```
int LineBuffer[3][IMG_W];
int WindowBuffer[3][3];

for(int i=0; i<rows; i++)
   for(int j=0; j<cols; j++)
      #pragma pipeline

      LineBuffer[0][j]=LineBuffer[1][j];
      LineBuffer[1][j]=LineBuffer[2][j];
      LineBuffer[2][j]=input_image[i][j];

      WindowBuffer[0][0] = LineBuffer[0][j];
      WindowBuffer[1][0] = LineBuffer[1][j];
      WindowBuffer[2][0] = LineBuffer[2][j];

      for(int k = 0; k < 3; k++) {
         WindowBuffer[k][2] = WindowBuffer[k][1];
         WindowBuffer[k][1] = WindowBuffer[k][0];
      }
      sobel_filter(WindowBuffer);
```

Convolution as a Matrix Multiplication

- Chellapilla, Kumar, Sidd Puri, and Patrice Simard. "High performance convolutional neural networks for document processing." *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.

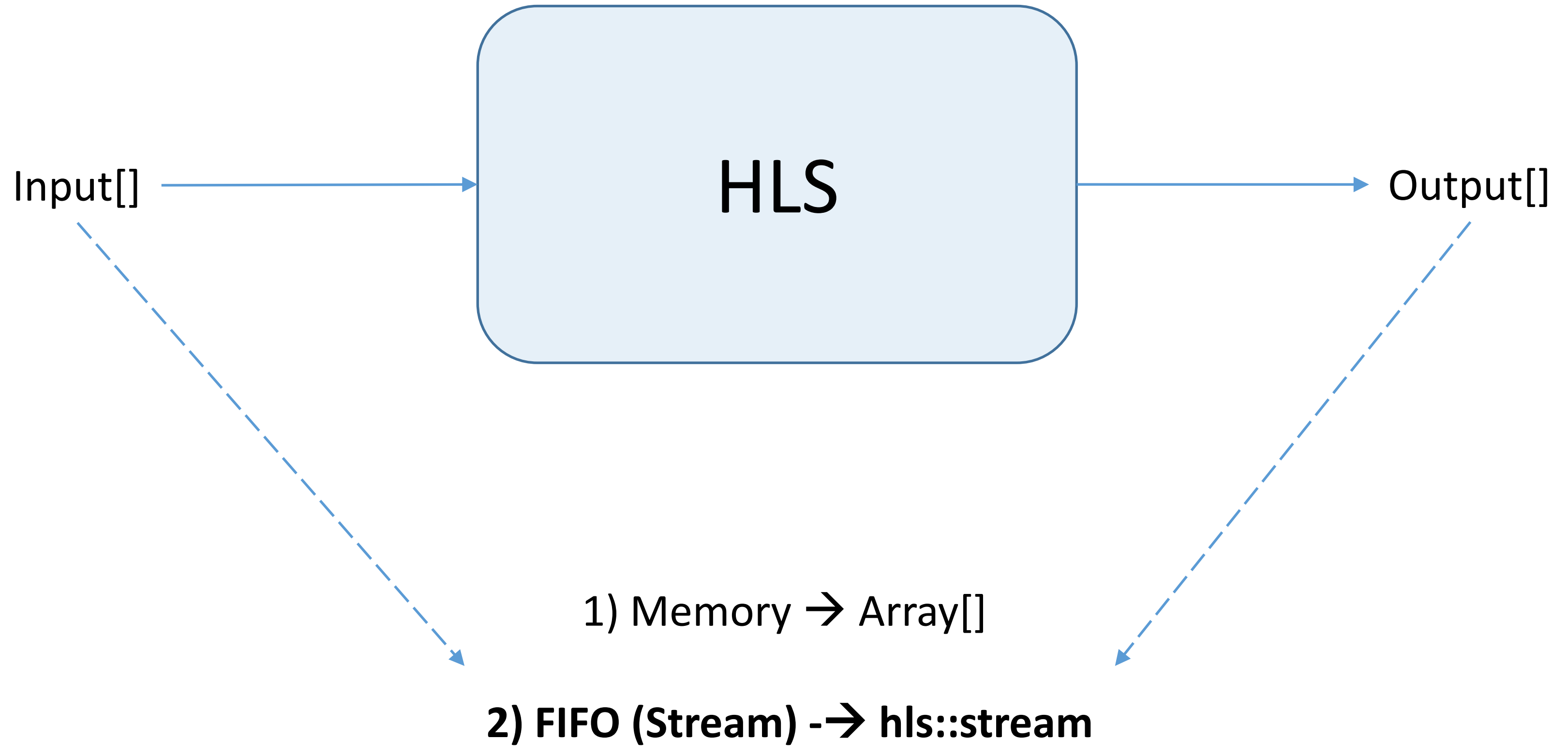
Advanced HLS Design Methods

1. Interface Synthesis
2. Dataflow Design
3. Streaming Design
4. Coding Techniques
    1. Histogram
    2. Streaming vector multiplication
    3. Convolution
    4. Prefix sum
    5. Streaming sorting HLS (example)
5. Implementing HLS IP on an FPGA ?

Interface Synthesis

Streaming Design

Discrete Fourier Transform (DFT)



Matrix Operations

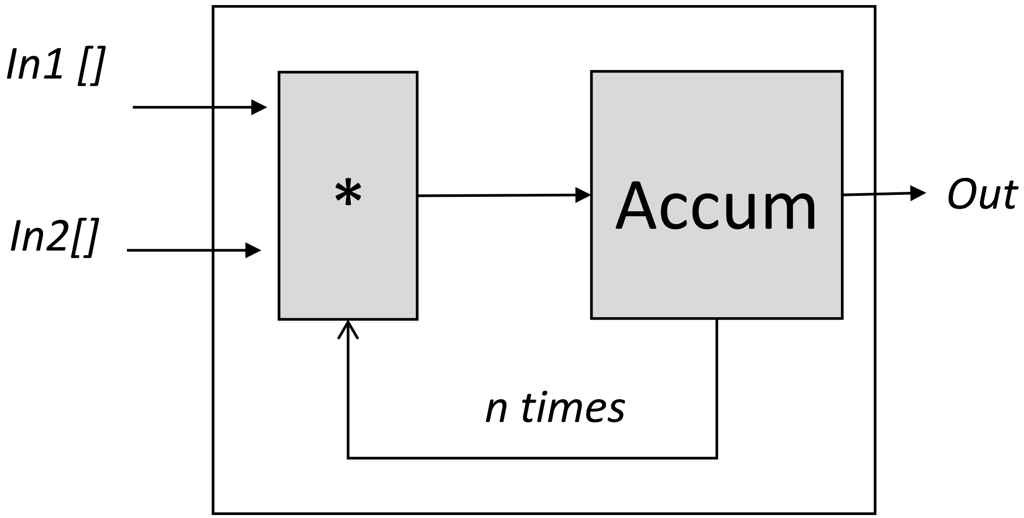
- Level 1: Vector *Vector
- Level 2: Matrix*Vector
- Level 3: Matrix*Matrix

Level 1: Vector*Vector operations

```
void vecvecmul(DTYPE in1[SIZE],
               DTYPE in2[SIZE],
               DTYPE out[1]){


for(int i=0;i<SIZE;i++){
   #pragma HLS PIPELINE II=1
   sum = sum+in1[i]*in2[i]
}
out[0] = sum;
}
```

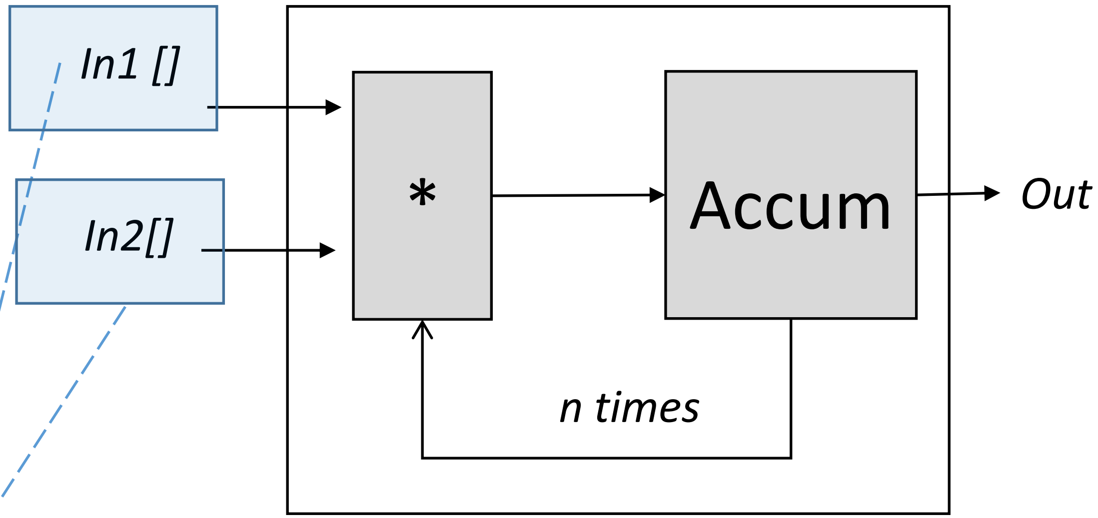
Level 1: Vector*Vector operations

```
void vecvecmul(DTYPE in1[SIZE],
               DTYPE in2[SIZE],
               DTYPE out[1]){


for(int i=0;i<SIZE;i++){
   #pragma HLS PIPELINE II=1
   sum = sum+in1[i]*in2[i]
}
out[0] = sum;
}
```



Level 1: Vector*Vector operations

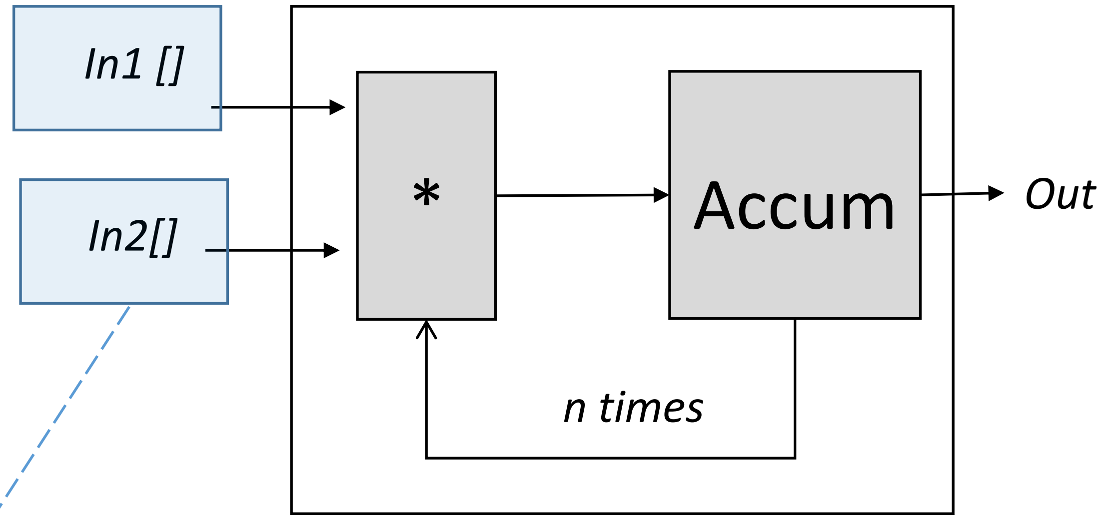
```
void vecvecmul(DTYPE in1[SIZE],
               DTYPE in2[SIZE],
               DTYPE out[1]){


for(int i=0;i<SIZE;i++){
   #pragma HLS PIPELINE II=1
   sum = sum+in1[i]*in2[i]
}
out[0] = sum;
}
```



Array is stored in a memory (BRAM)

Level 1: Vector*Vector operations

```
void vecvecmul(DTYPE in1[SIZE],
               DTYPE in2[SIZE],
               DTYPE out[1]){


for(int i=0;i<SIZE;i++){
   #pragma HLS PIPELINE II=1
   sum = sum+in1[i]*in2[i]
}
out[0] = sum;
}
```



Array is stored in a memory (BRAM)

What if I do not want to use BRAM (s) ?

Level 1: Vector*Vector operations

Hls::stream

-read(0

-write()

Level 1: Vector*Vector operations

```
void vecvecmul(hls::stream<DTYPE> &in1,
                hls::stream<DTYPE> &in2,
                hls::stream<DTYPE> &out){









}
```

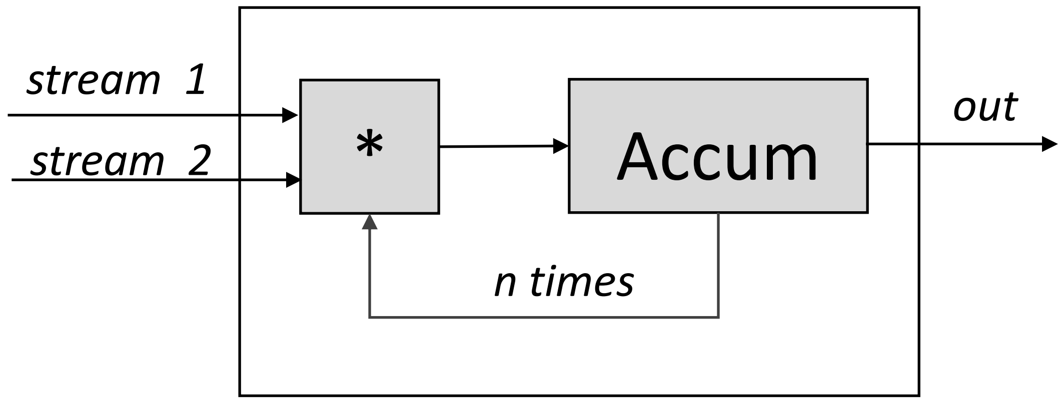
Level 1: Vector*Vector operations

```
void vecvecmul(hls::stream<DTYPE> &in1,
                hls::stream<DTYPE> &in2,
                hls::stream<DTYPE> &out){


for(int i=0;i<SIZE;i++){




}


}
```

Level 1: Vector*Vector operations

```
void vecvecmul(hls::stream<DTYPE> &in1,
               hls::stream<DTYPE> &in2,
               hls::stream<DTYPE> &out){
DTYPE b=0,a=0;
DTYPE sum=0;
for(int i=0;i<SIZE;i++){
   #pragma HLS PIPELINE II=1
   a = in1.read();
   b = in2.read();
   sum=sum+a*b;
}
out.write(sum);
}
```

Level 1: Vector*Vector operations

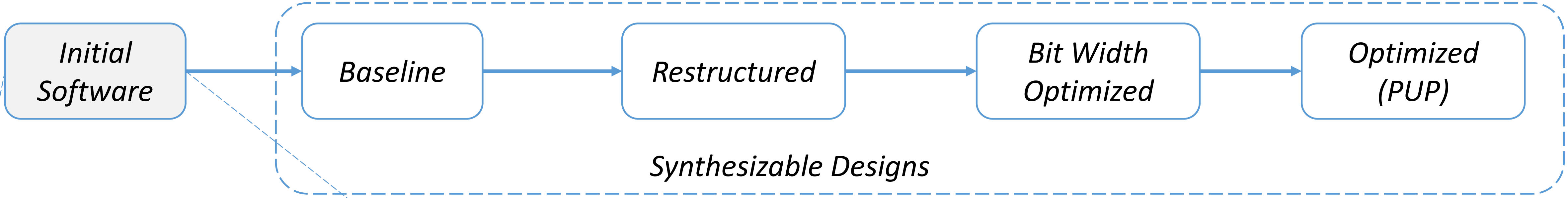



```
void vecvecmul(hls::stream<DTYPE> &in1,
                hls::stream<DTYPE> &in2,
                hls::stream<DTYPE> &out){
DTYPE b=0,a=0;
DTYPE sum=0;
for(int i=0;i<SIZE;i++){
   #pragma HLS PIPELINE II=1
   a = in1.read();
   b = in2.read();
   sum=sum+a*b;
}
out.write(sum);
}
```

Coding Techniques

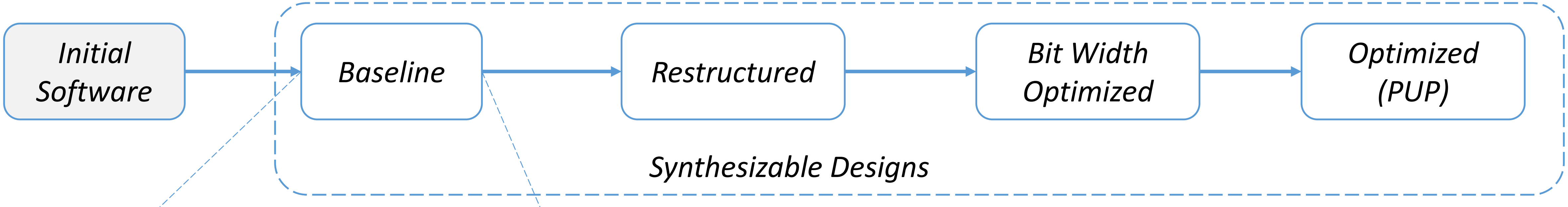
- Prefix sum
- Insertion sort
- Sobel filter
- Histogram

HLS Design Flow



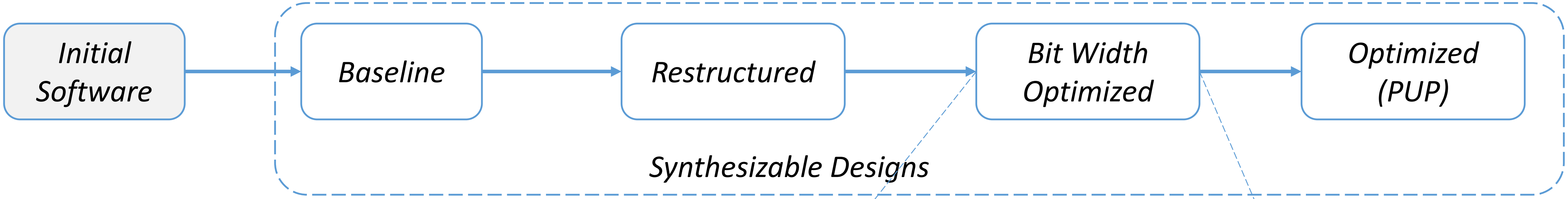

```c
int *array;
array = (int *)malloc(sizeof(int)*N);
for(int i=0;i<N;i++) {

array = … ;
}
```

HLS Design Flow



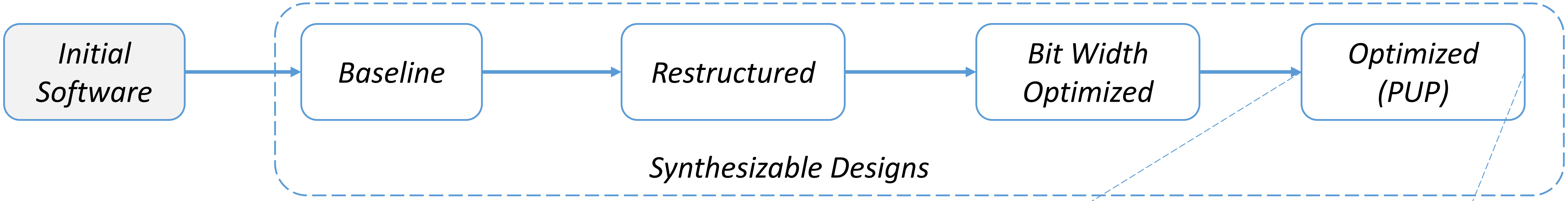

```
int *array;
array[N] ={0};
for(int i=0;i<N;i++) {

array = … ;
}
```

HLS Design Flow




```
ap_int<10> array[N] ={0};
for(int i=0;i<N;i++) {

array = … ;
}
```

HLS Design Flow




```
ap_int<10> array[N] ={0};
for(int i=0;i<N;i++) {
#pragma HLS PIPELINE
array = … ;
}
```

❖*Restructured HLS Code = Hardware design patterns + C code + directives*

Prefix sum

Input:	0	2	1	6	3	1	2	5
Output:	0	2	3	9	12	13	15	20

```
void prefixsum(I_TYPE in[SIZE], I_TYPE out[SIZE]){
    for(int i = 1; i < SIZE; ++i){
        out[i] = out[i-1] + in[i];
    }
}
```

Applications: Radix-Sort, Quick Sort, Line-Of-Sight, Recurrence Equations, Compaction Problem, String comparison, Polynomial evaluation, Histogram,..

Prefix sum: Baseline Optimization (BO)

- Optimize with **#pragmas** without changing code

```
void prefixsum(I_TYPE in[SIZE], I_TYPE out[SIZE]){


out[0] = in[0];
for(int i = 1; i < SIZE; ++i)
{


out[i] = out[i-1] + in[i];
}
}
```

Prefix sum: Baseline Optimization (BO)

- Optimize with **#pragmas** without changing code

```
void prefixsum(I_TYPE in[SIZE], I_TYPE out[SIZE]){


out[0] = in[0];
for(int i = 1; i < SIZE; ++i)
{
#pragma HLS UNROLL factor=4
#pragma HLS PIPELINE
out[i] = out[i-1] + in[i];
}
}
```

Prefix sum: Baseline Optimization (BO)

- Optimize with **#pragmas** without changing code

```
void prefixsum(I_TYPE in[SIZE], I_TYPE out[SIZE]){
#pragma HLS ARRAY_PARTITION variable=out cyclic factor=4 dim=1
#pragma HLS ARRAY_PARTITION variable=in cyclic factor=4 dim=1
out[0] = in[0];
for(int i = 1; i < SIZE; ++i)
{
#pragma HLS UNROLL factor=4
#pragma HLS PIPELINE
out[i] = out[i-1] + in[i];
}
}
```

Prefix sum: Baseline Optimization (BO)

- Optimize with **#pragmas** without changing code

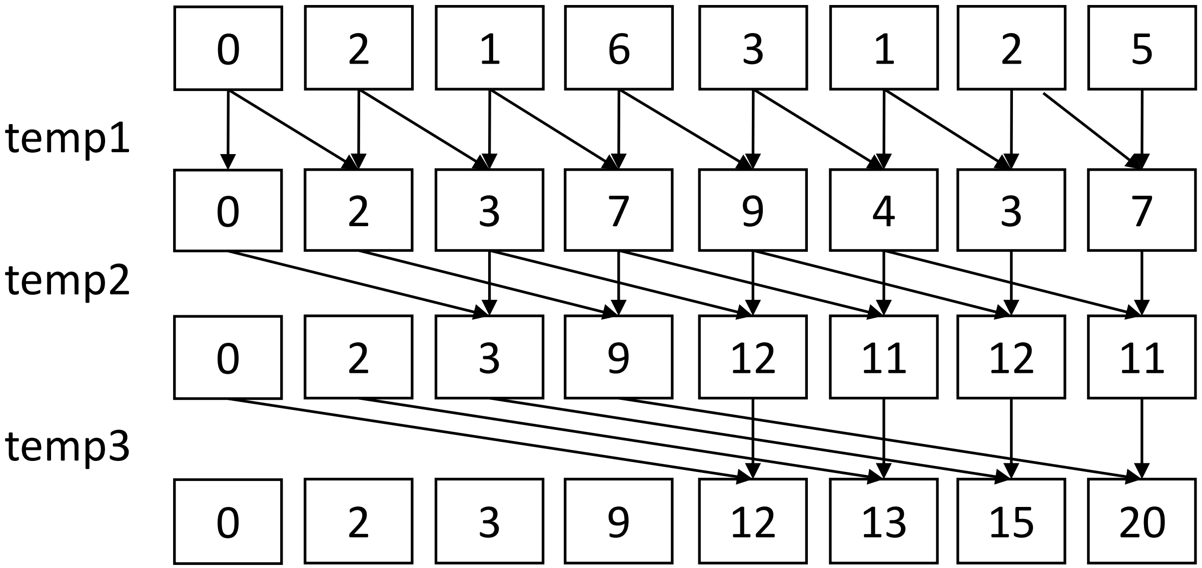
```
void prefixsum(I_TYPE in[SIZE], I_TYPE out[SIZE]){
#pragma HLS ARRAY_PARTITION variable=out cyclic factor=4 dim=1
#pragma HLS ARRAY_PARTITION variable=in cyclic factor=4 dim=1
out[0] = in[0];
for(int i = 1; i < SIZE; ++i)
{
#pragma HLS UNROLL factor=4
#pragma HLS PIPELINE
out[i] = out[i-1] + in[i];
}
}
```

✓Expect to speed up by 4X, but it will give around 2X!

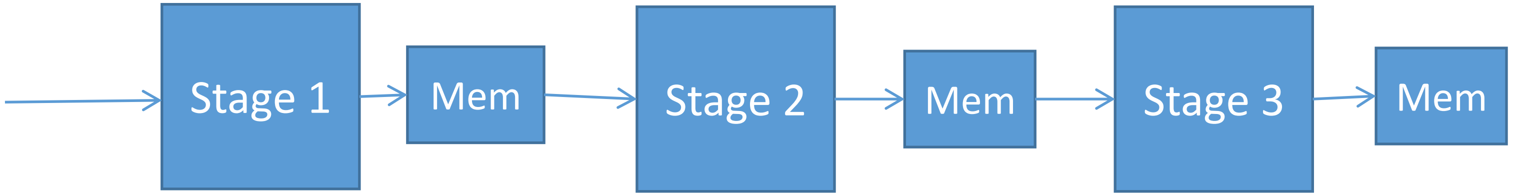
✓Factor = 4, 8, 16,.. gives the same result

  ✓→ Must change code to get 4X speed up

Prefix sum: Restructured and Optimized 1 (RO1)



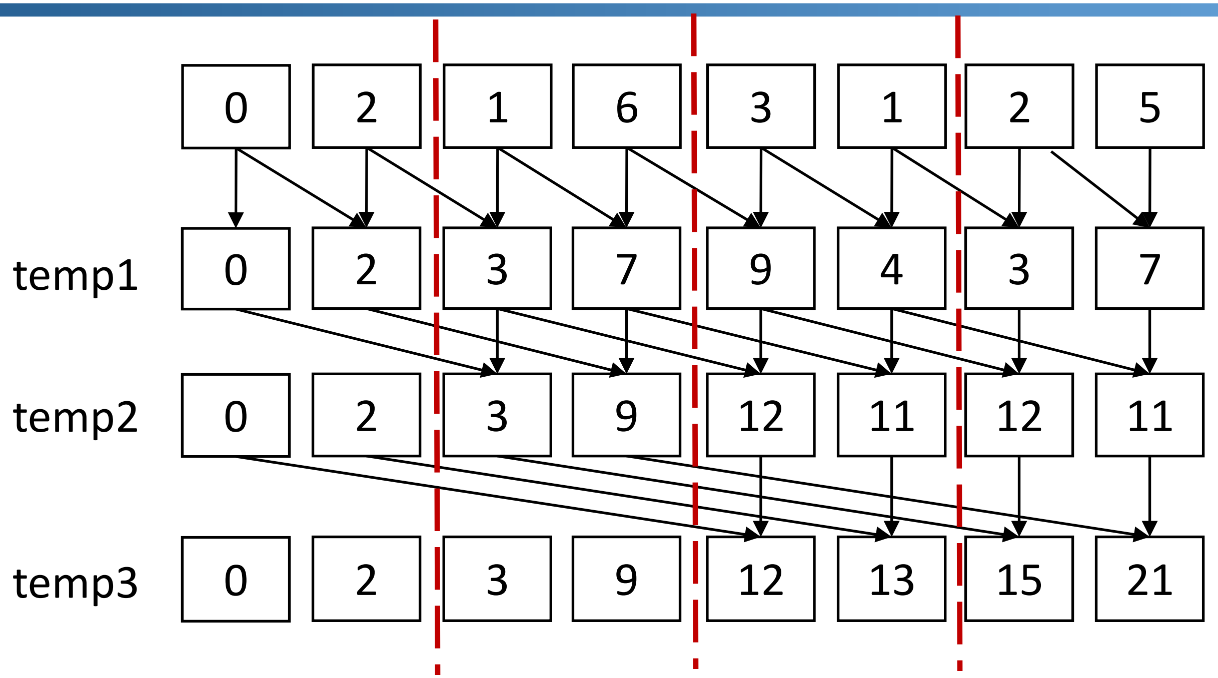

```
void prefixsum(I_TYPE in[SIZE], I_TYPE
out[SIZE]){
#pragma dataflow

for(int i = 1; i < SIZE;){
temp1[i] = in[i-1]+in[i];}

for(int i = 2; i < SIZE;){
temp2[i] = temp1[i-2]+temp1[i];}

for(int i = 4; i < SIZE;){
temp3[i] = temp2[i-4]+temp2[i];}
}
```



✓Throughput increases

✓How about large prefix sum like 1024 ?

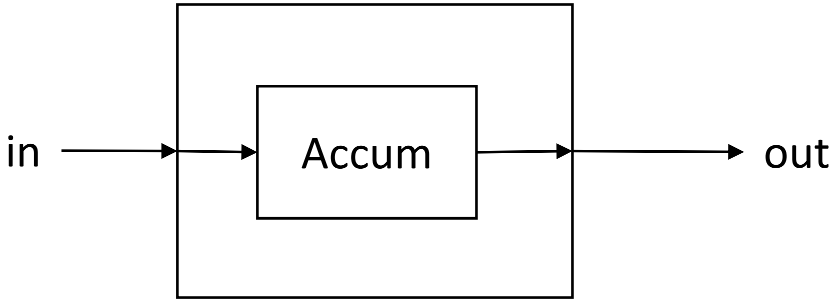
Prefix sum: Restructured and Optimized 1 (RO1)




```
Int temp1[SIZE];
#pragma HLS ARRAY_PARTITION variable=temp1 cyclic factor=4 dim=1
Int temp2[SIZE];
#pragma HLS ARRAY_PARTITION variable=temp2 cyclic factor=4 dim=1
Int temp3[SIZE];
#pragma HLS ARRAY_PARTITION variable=temp3 cyclic factor=4 dim=1

for(int i = 1; i < SIZE;){
#pragma HLS UNROLL factor=4
#pragma HLS PIPELINE
temp1[i] = in[i-1]+in[i];}

for(int i = 2; i < SIZE;){
#pragma HLS UNROLL factor=4
#pragma HLS PIPELINE
temp2[i] = temp1[i-2]+temp1[i];}

for(int i = 4; i < SIZE;){
#pragma HLS UNROLL factor=4
#pragma HLS PIPELINE
temp3[i] = temp2[i-4]+temp2[i];}
```

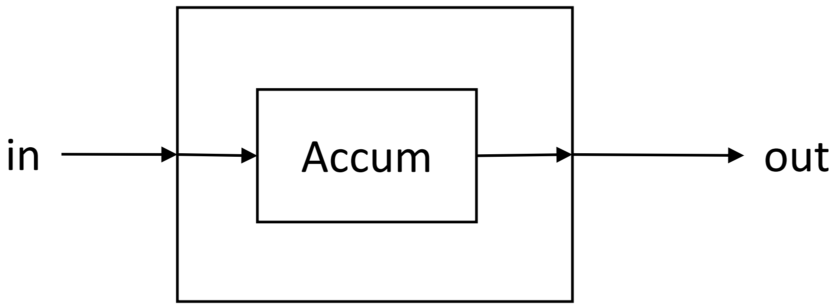
- Factor = 4, 8 , 16
- Size =1024, 2048, 4096,..

Prefix sum: Restructured and Optimized 1 (RO2)




```
void prefixsum(I_TYPE in[SIZE], I_TYPE out[SIZE]){



I_TYPE A=in[0];
for(int i = 1; i < SIZE; ++i){


          A = A+in[i];
          out[i] =  A;
          }
}
```

Prefix sum: Restructured and Optimized 1 (RO2)

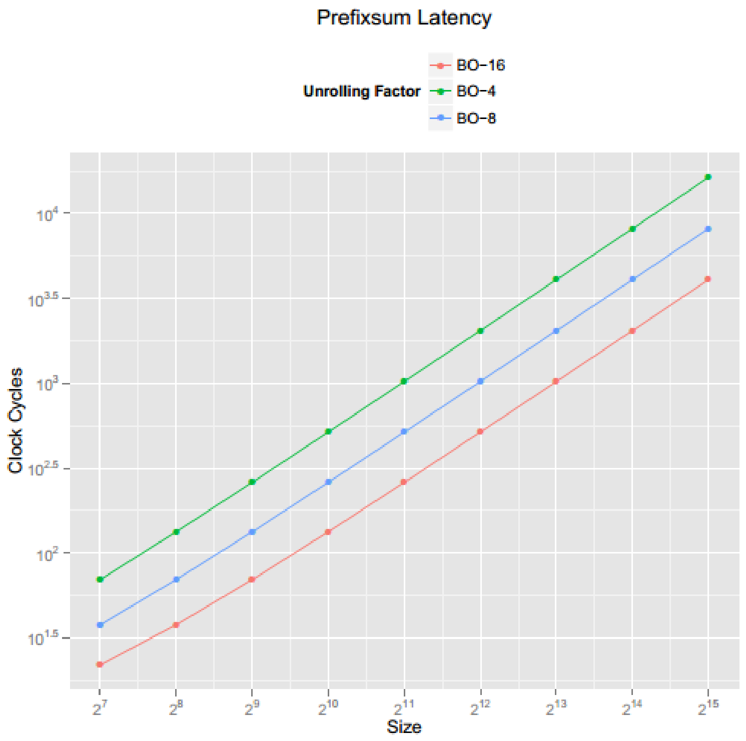


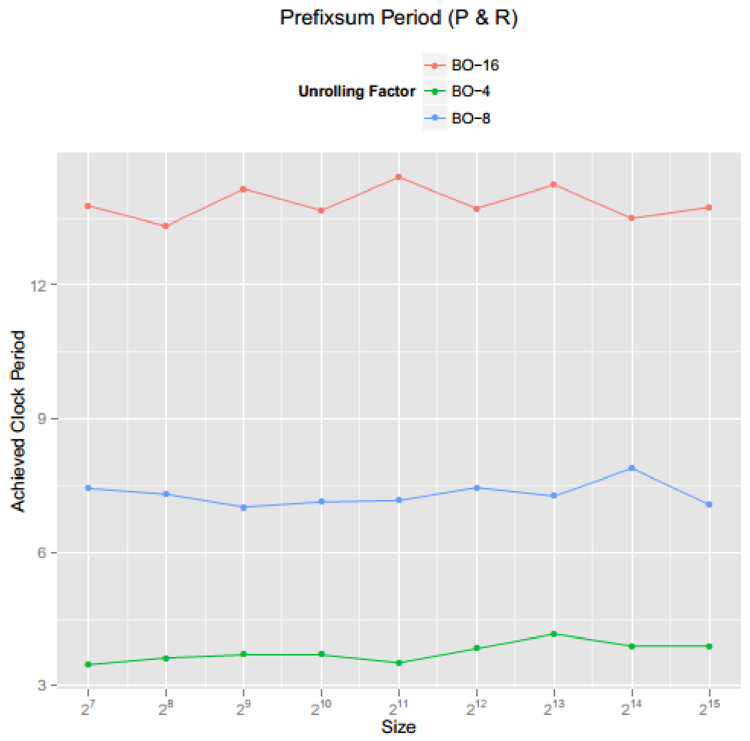

```
void prefixsum(I_TYPE in[SIZE], I_TYPE out[SIZE]){
#pragma HLS ARRAY_PARTITION variable=out cyclic factor=4 dim=1
#pragma HLS ARRAY_PARTITION variable=in cyclic factor=4 dim=1

I_TYPE A=in[0];
for(int i = 1; i < SIZE; ++i){
         #pragma HLS UNROLL factor=4
         #pragma HLS PIPELINE
         A = A+in[i];
         out[i] =  A;
         }
}
```

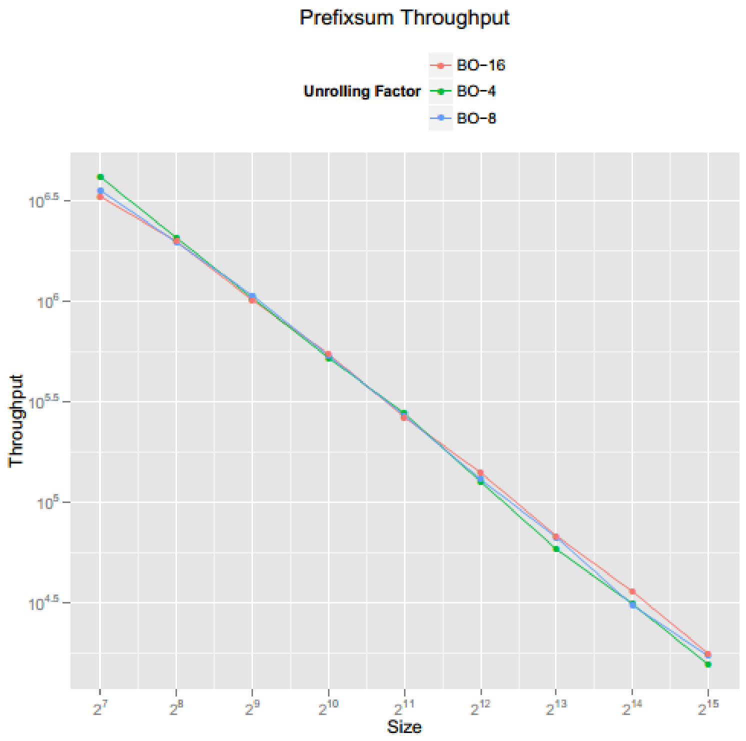
Results

- **How clock cycles, frequency, area impacted when:**
  - **Baseline Optimized (BO):** C code + best set of #pragmas
  - **Restructured and Optimized 1 (RO1):** Task level parallelism
  - **Restructured and Optimized 1 (RO2) :** Memory

Baseline Optimized (BO)





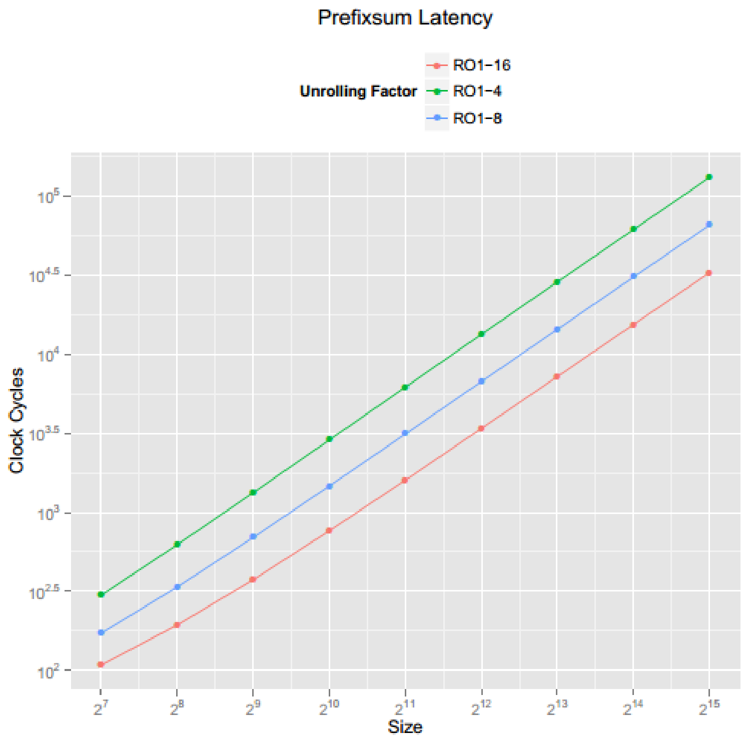


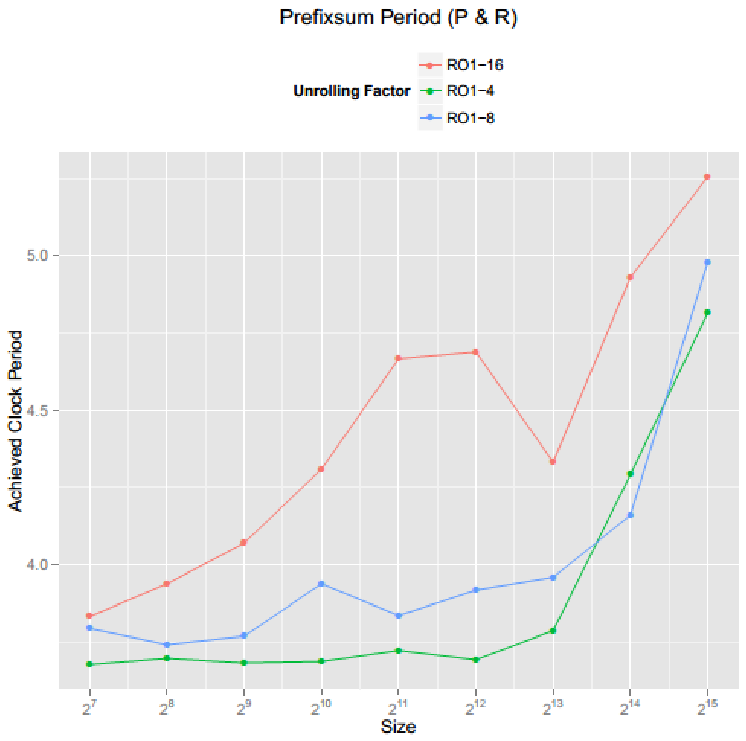
Clock cycles improve by a factor (4)

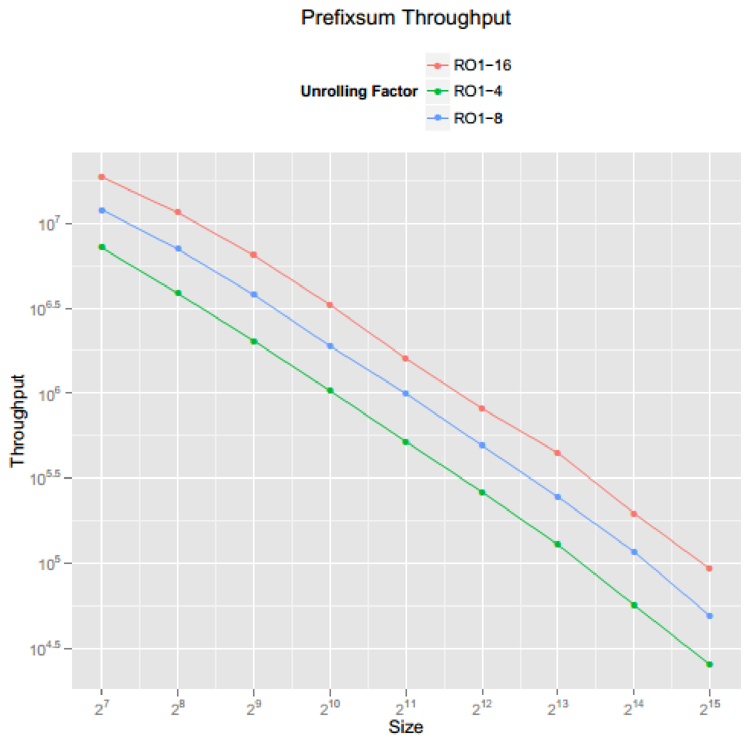
Clock Period (Frequency) decrease by a factor of 4

Throughput does not imrpove!

Restructured and Optimized (RO1)





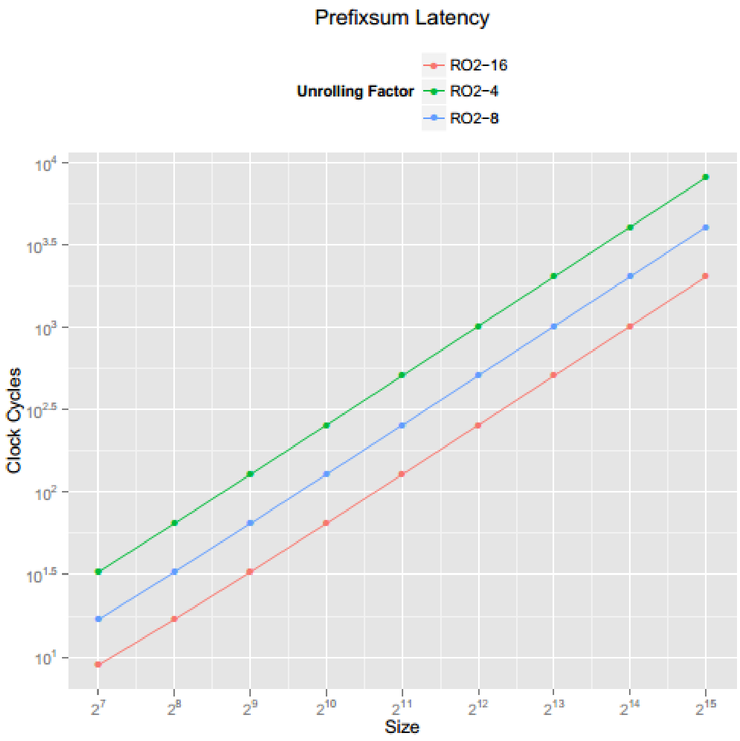


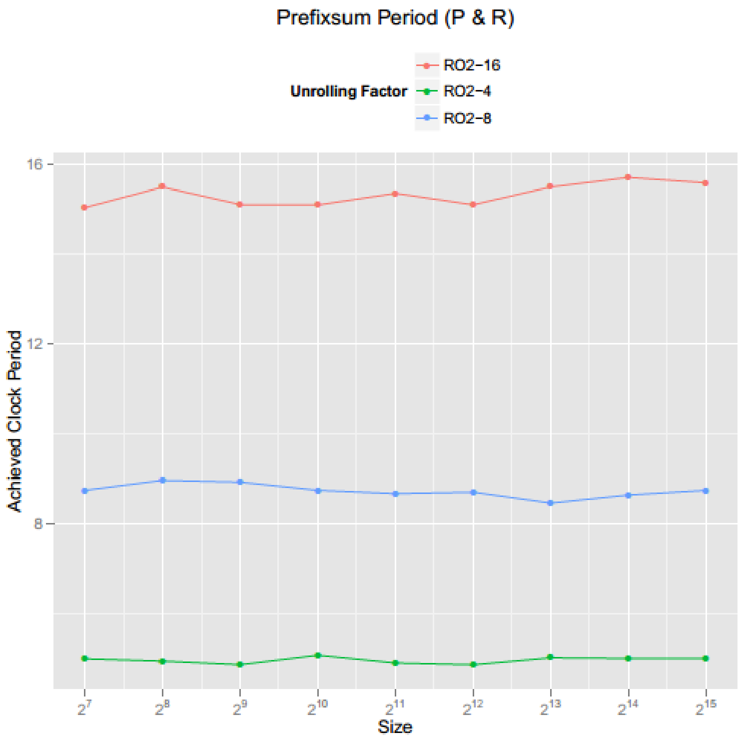
Clock cycles improve by a factor (4)

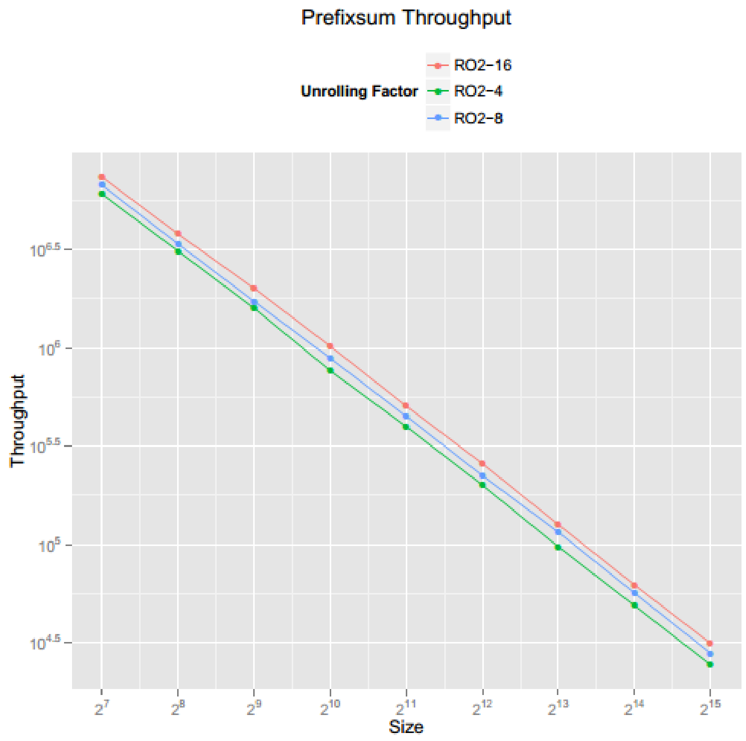
Clock Period (Frequency) remain the same!

Throughput does improve as we want!

Restructured and Optimized (RO2)







Clock cycles improve by a factor (4)

Clock Period (Frequency) gets worse!

Throughput does imrpove by litle!

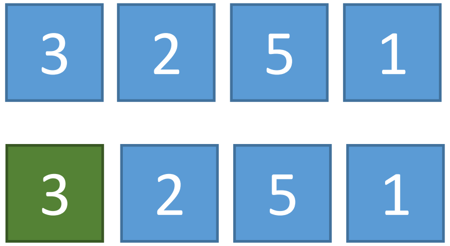
Insertion Sort

3	2	5	1

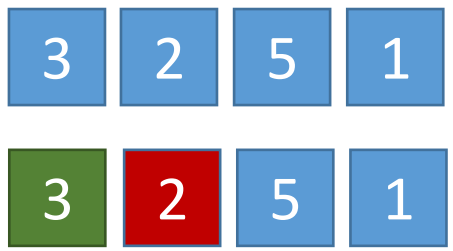
Insertion Sort

3	2	5	1

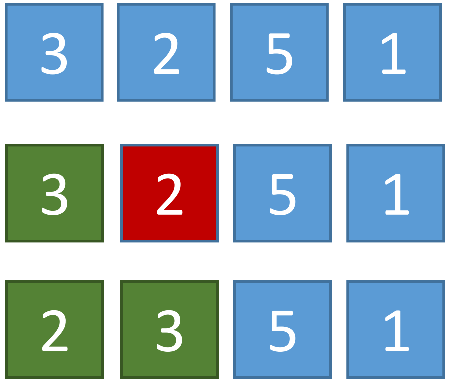
Insertion Sort



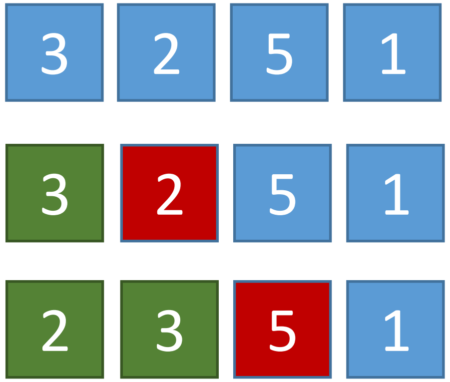
Insertion Sort



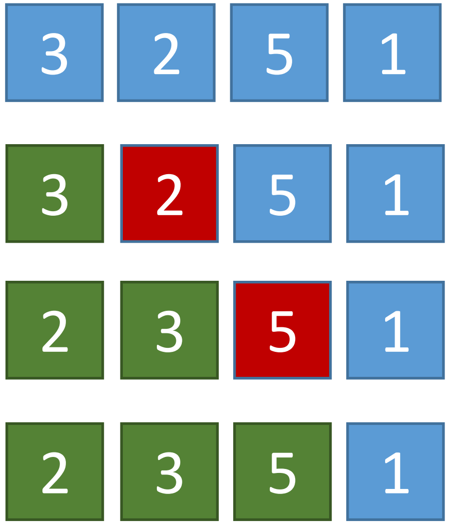
Insertion Sort



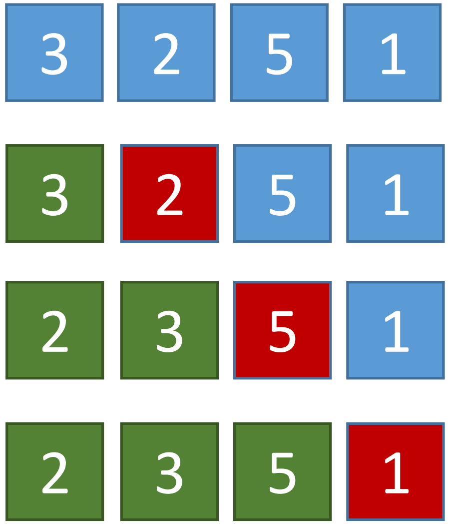
Insertion Sort



Insertion Sort

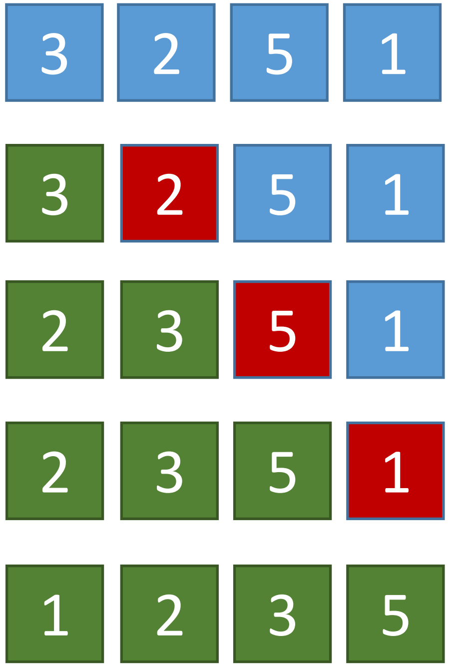


Insertion Sort



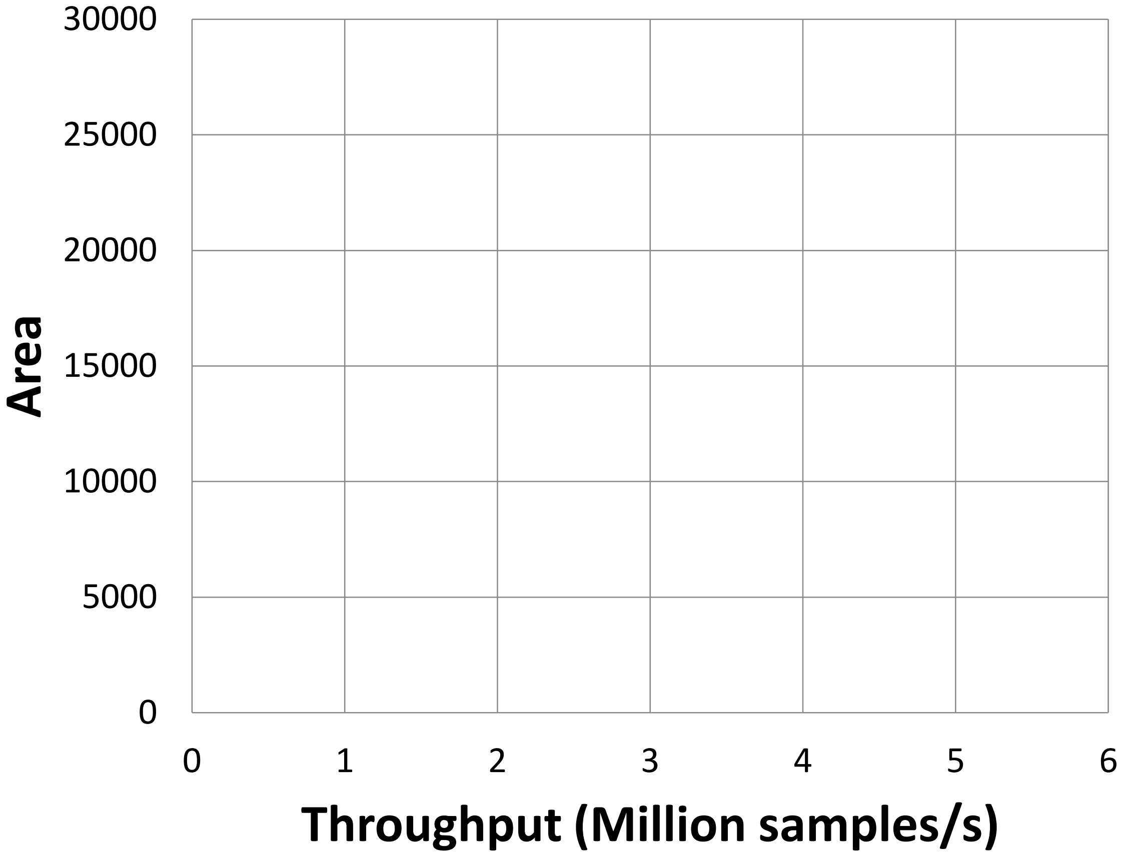
Insertion Sort

```
void InsertionSort(DTYPE numbers[n])
{

  int i, j, index;
  for (i=1; i < n; i++)
  {
    index = numbers[i];
    j = i;
    while ((j > 0)&& (numbers[j-1] > index) )
    {


         numbers[j] = numbers[j-1];
         j = j - 1;
    }
    numbers[j] = index;
  }
}
```



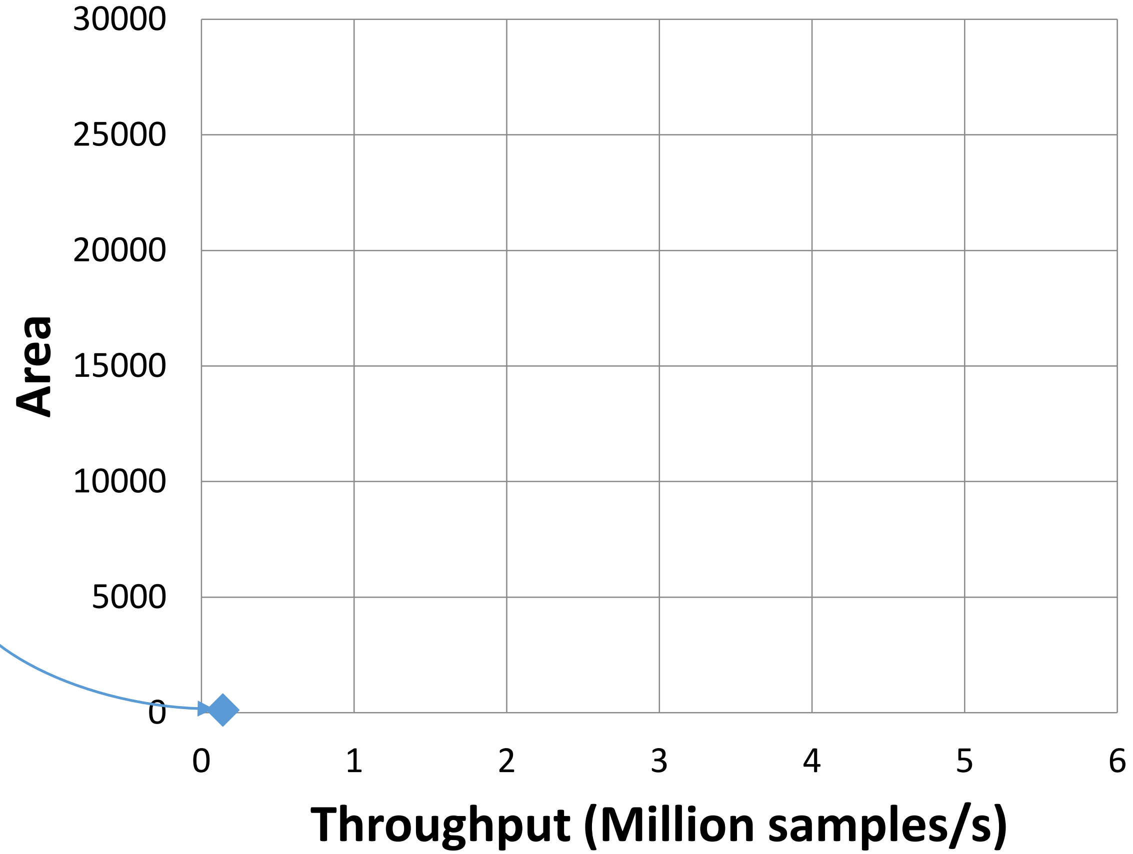
Reusable and Optimized Template

```
void InsertionSort(DTYPE numbers[n])
{

  int i, j, index;
  for (i=1; i < n; i++)
  {
    index = numbers[i];
    j = i;
    while ((j > 0)&& (numbers[j-1] > index) )
    {


         numbers[j] = numbers[j-1];
         j = j - 1;
    }
    numbers[j] = index;
  }
}
```



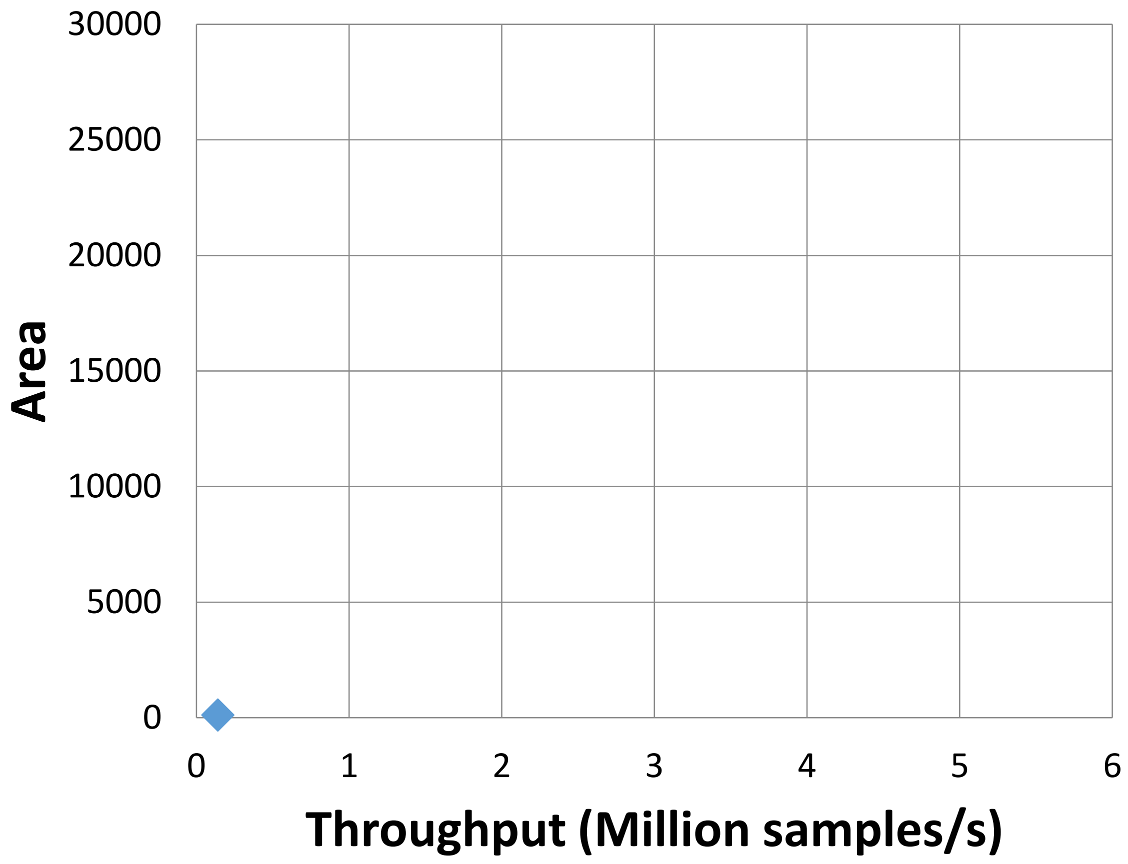
Reusable and Optimized Template

```
void InsertionSort(DTYPE numbers[n])
{

  int i, j, index;
  for (i=1; i < n; i++)
  {
    index = numbers[i];
    j = i;
    while ((j > 0)&& (numbers[j-1] > index) )
    {


         numbers[j] = numbers[j-1];
         j = j - 1;
    }
    numbers[j] = index;
  }
}
```



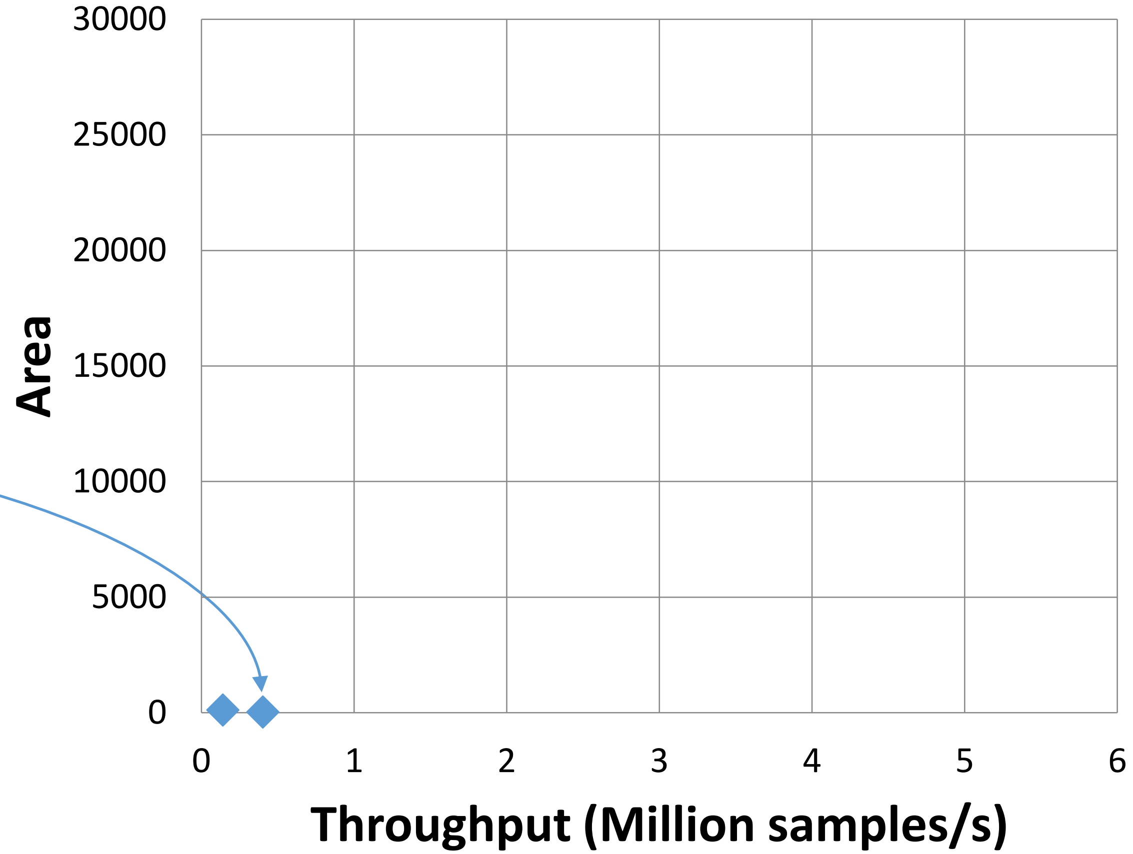
Reusable and Optimized Template

```
void InsertionSort(DTYPE numbers[n])
{

  int i, j, index;
  for (i=1; i < n; i++)
  {
    index = numbers[i];
    j = i;
    while ((j > 0)&& (numbers[j-1] > index) )
    {
          #pragma HLS PIPELINE II=1

          numbers[j] = numbers[j-1];
          j = j - 1;
    }
    numbers[j] = index;
  }
}
```



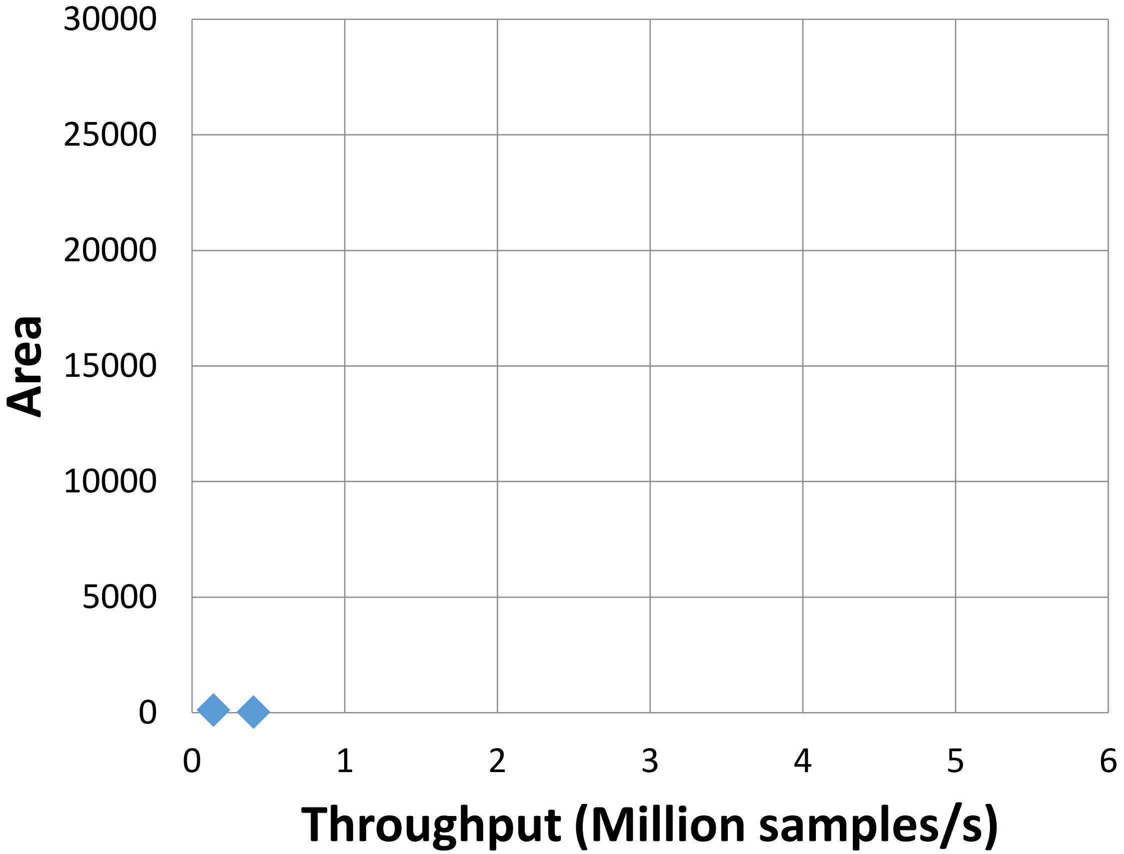
Reusable and Optimized Template

```
void InsertionSort(DTYPE numbers[n])
{

  int i, j, index;
  for (i=1; i < n; i++)
  {
    index = numbers[i];
    j = i;
    while ((j > 0)&& (numbers[j-1] > index) )
    {
         #pragma HLS PIPELINE II=1

         numbers[j] = numbers[j-1];
         j = j - 1;
    }
    numbers[j] = index;
  }
}
```



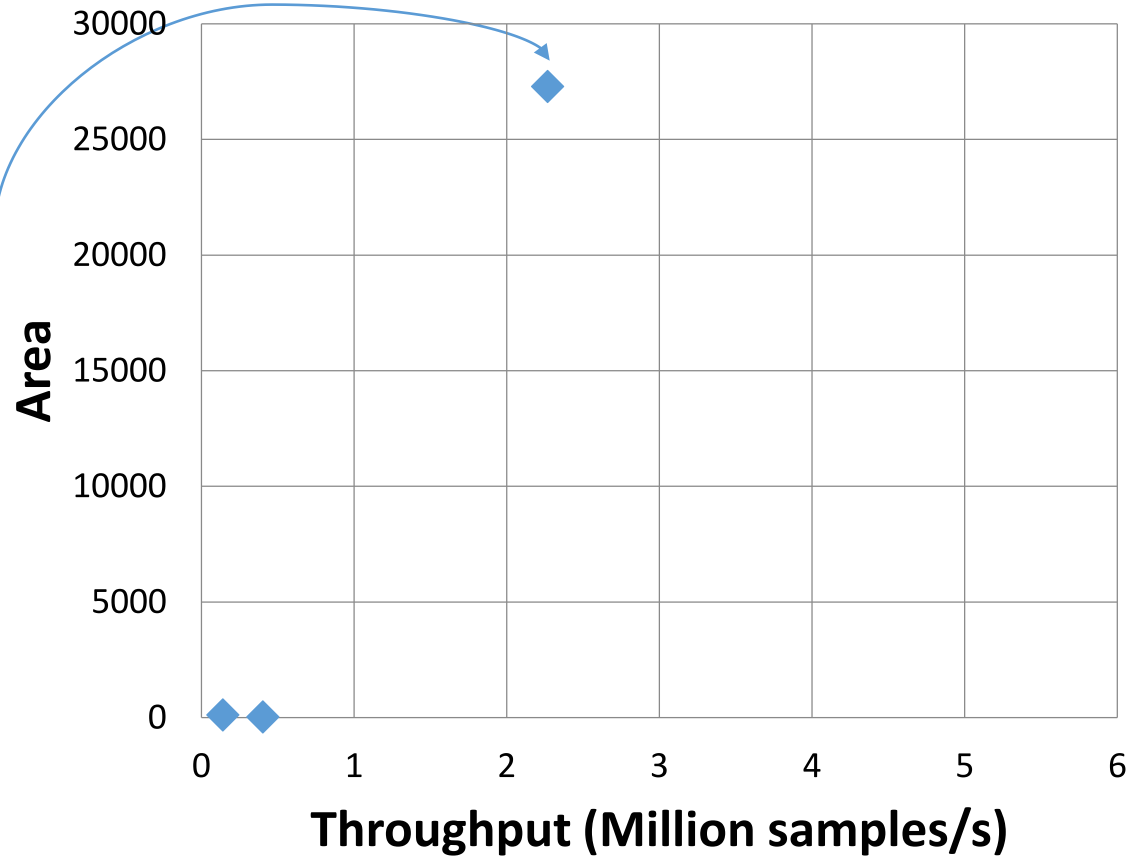
Reusable and Optimized Template

```
void InsertionSort(DTYPE numbers[n])
{
  #pragma HLS PIPELINE II=1
  int i, j, index;
  for (i=1; i < n; i++)
  {
    index = numbers[i];
    j = i;
    while ((j > 0)&& (numbers[j-1] > index) )
    {


         numbers[j] = numbers[j-1];
         j = j - 1;
    }
    numbers[j] = index;
  }
}
```



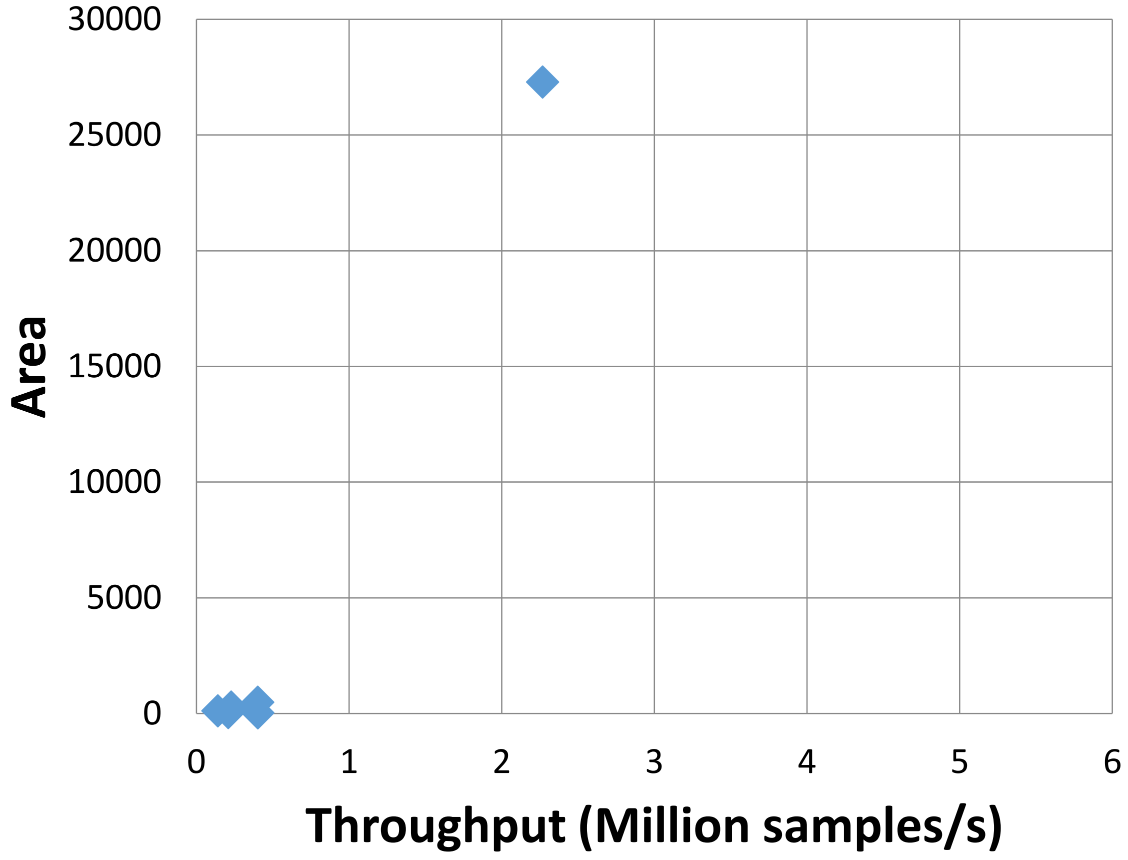
Reusable and Optimized Template

```
void InsertionSort(DTYPE numbers[n])
{
  #pragma HLS PIPELINE II=1
  int i, j, index;
  for (i=1; i < n; i++)
  {
    index = numbers[i];
    j = i;
    while ((j > 0)&& (numbers[j-1] > index) )
    {


        numbers[j] = numbers[j-1];
        j = j - 1;
    }
    numbers[j] = index;
  }
}
```



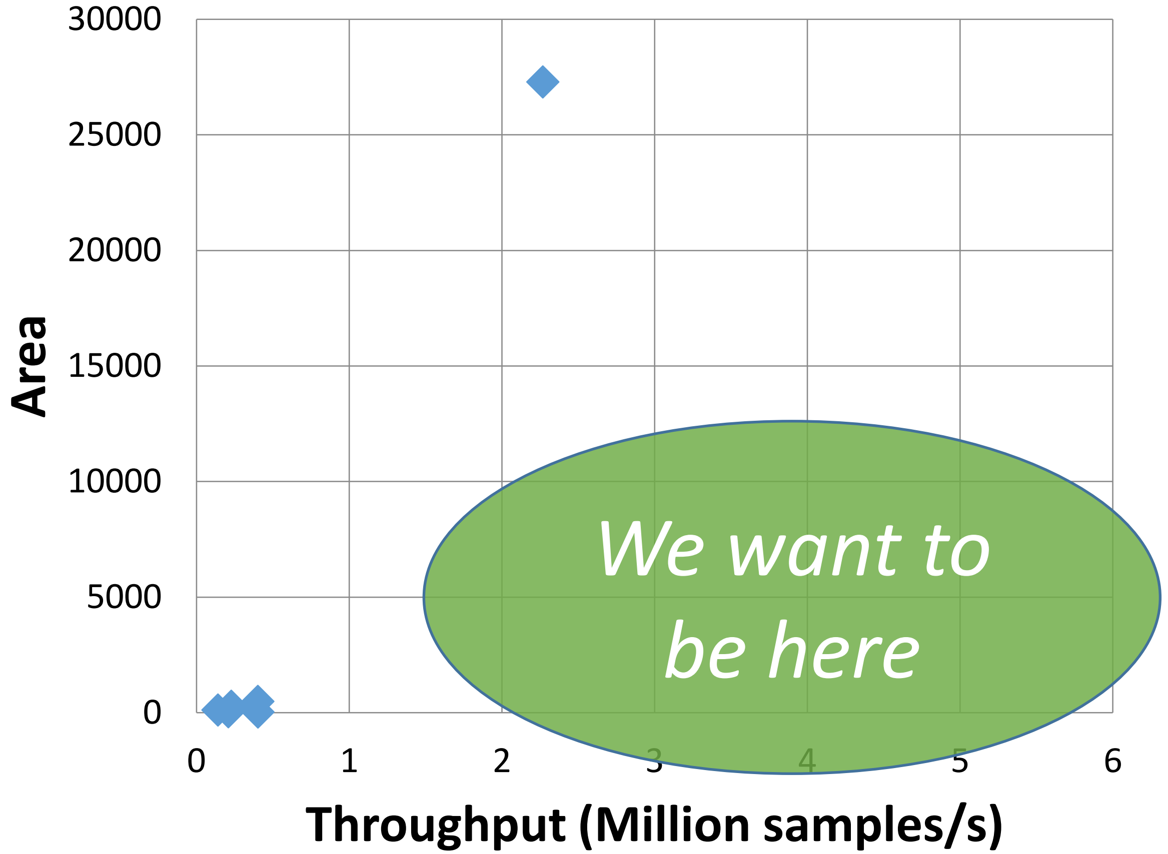
Reusable and Optimized Template

```
void InsertionSort(DTYPE numbers[n])
{

  int i, j, index;
  for (i=1; i < n; i++)
  {
    index = numbers[i];
    j = i;
    while ((j > 0)&& (numbers[j-1] > index) )
    {

         #pragma UNROLL FACTOR=4
         numbers[j] = numbers[j-1];
         j = j - 1;
    }
    numbers[j] = index;
  }
}
```



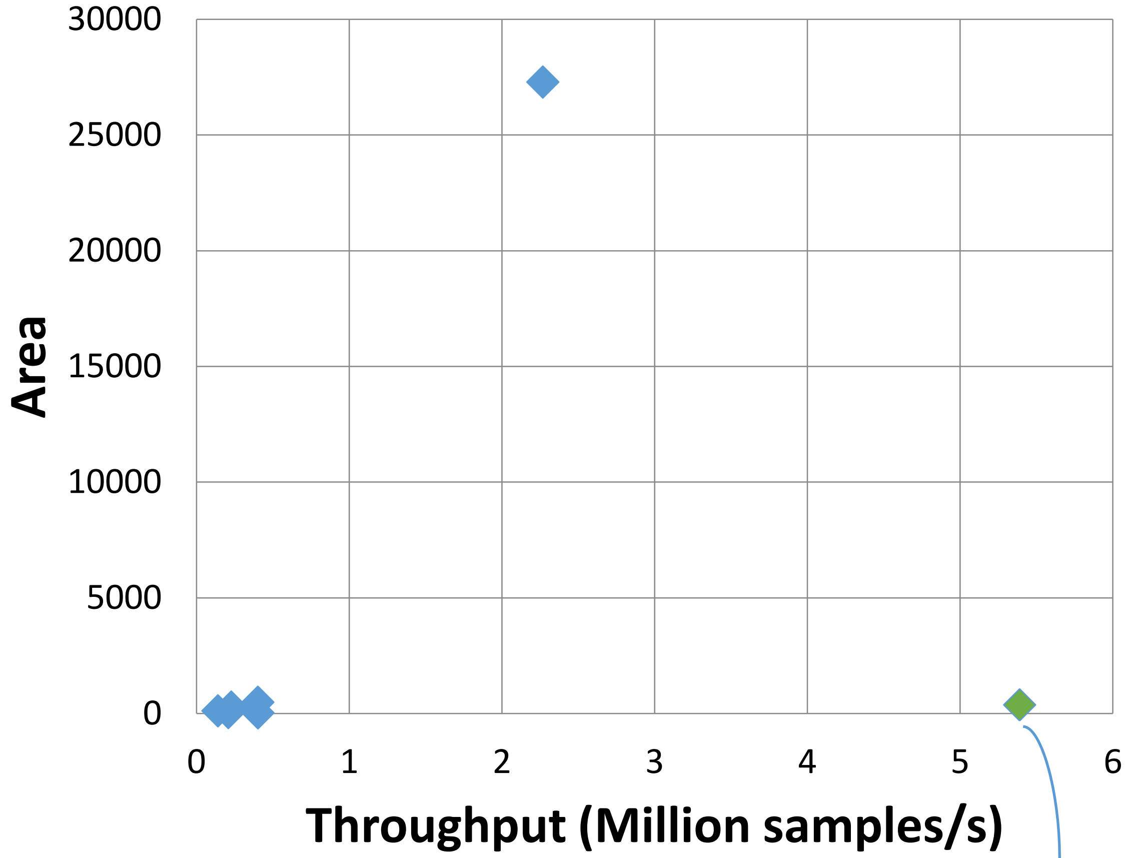
Reusable and Optimized Template

```
void InsertionSort(DTYPE numbers[n])
{

  int i, j, index;
  for (i=1; i < n; i++)
  {
    index = numbers[i];
    j = i;
    while ((j > 0)&& (numbers[j-1] > index) )
    {

         #pragma UNROLL FACTOR=4
         numbers[j] = numbers[j-1];
         j = j - 1;
    }
    numbers[j] = index;
  }
}
```

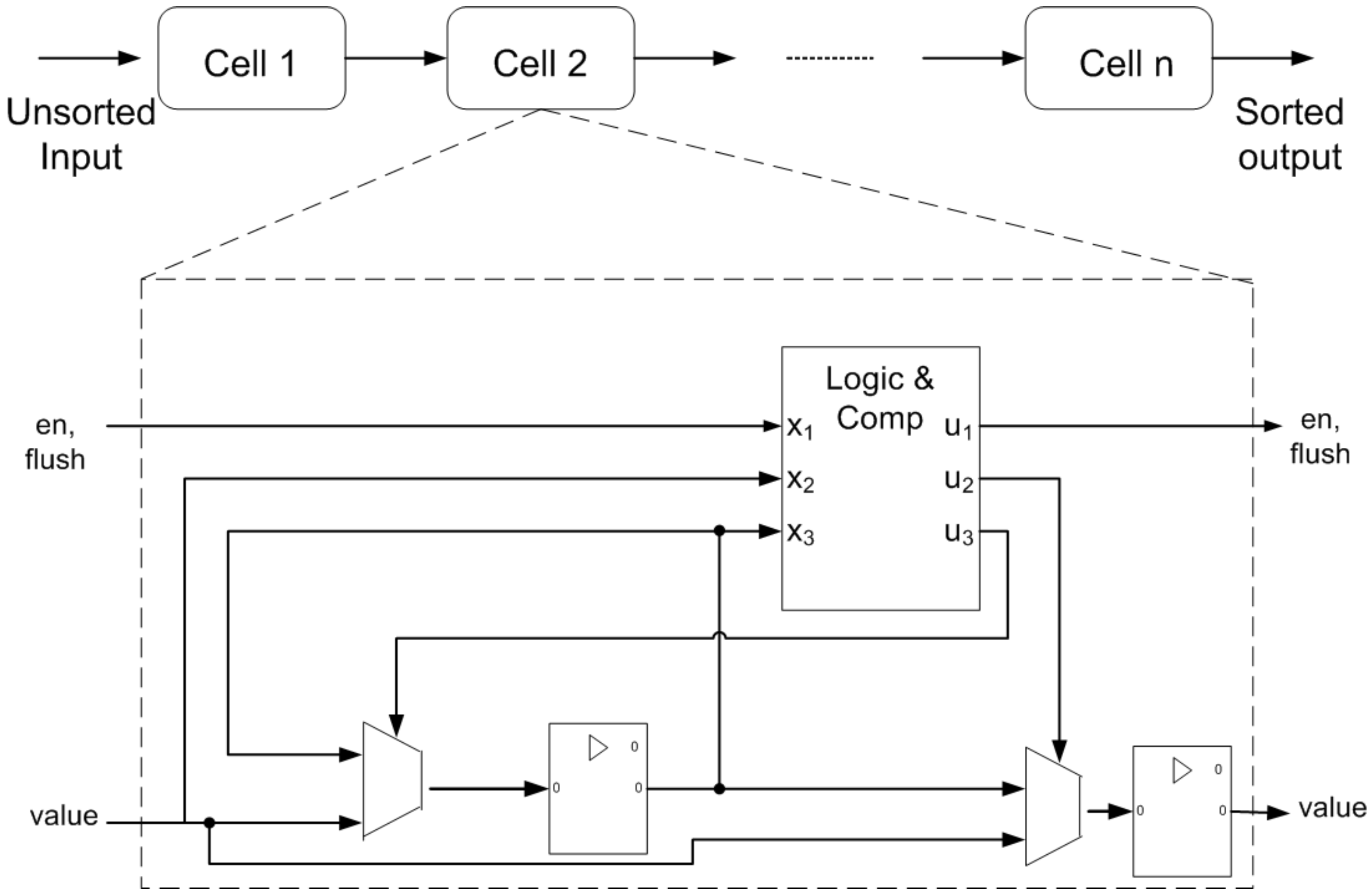


Reusable and Optimized Template

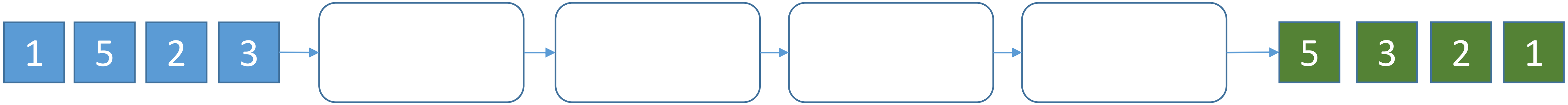
```
void InsertionSort(DTYPE numbers[n])
{

  int i, j, index;
  for (i=1; i < n; i++)
  {
    index = numbers[i];
    j = i;
    while ((j > 0)&& (numbers[j-1] > index) )
    {
          #pragma HLS PIPELINE II=1

          numbers[j] = numbers[j-1];
          j = j - 1;
    }
    numbers[j] = index;
  }
}
```



Insertion Sort: Hardware

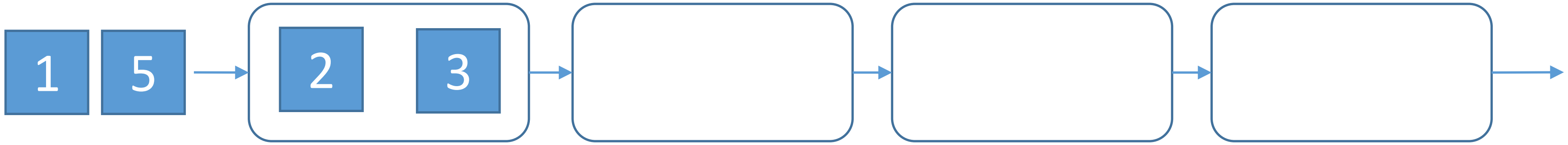


Insertion Sort: Hardware

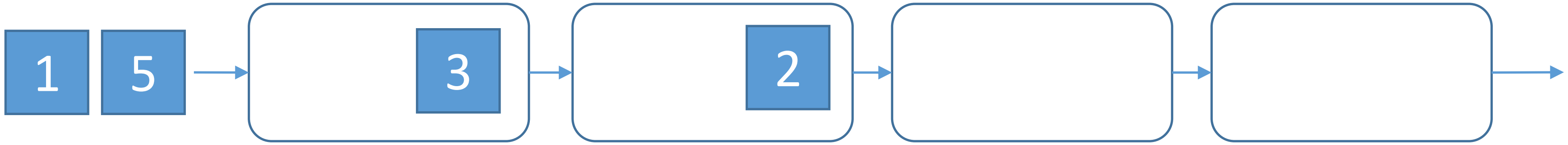


Insertion Sort: Hardware

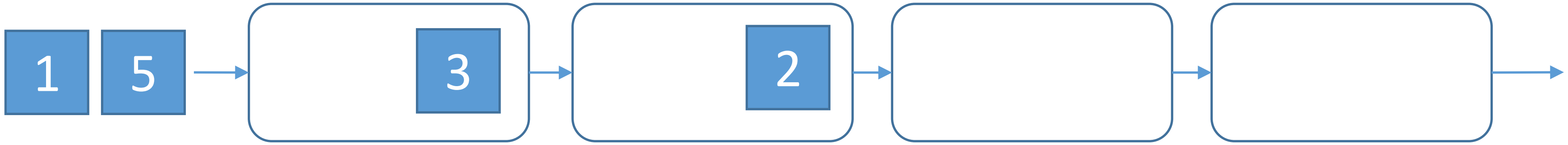
Insertion Sort: Hardware



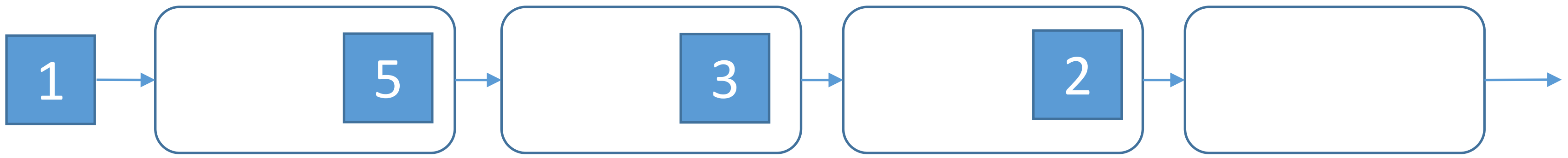
Insertion Sort: Hardware



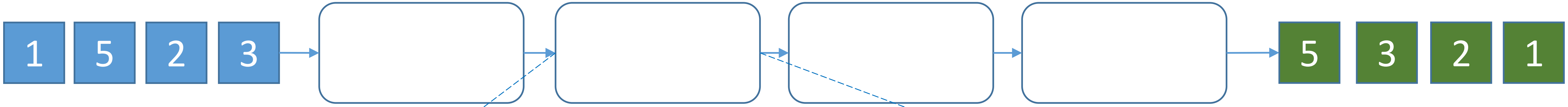
Insertion Sort: Hardware



Insertion Sort: Hardware

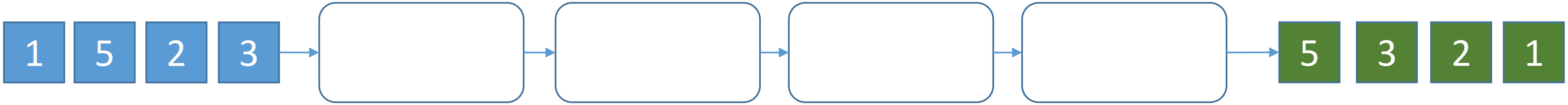


Insertion Sort: Hardware




```
void cell(hls::stream<DTYPE> &IN, hls::stream<DTYPE> &OUT){
static DTYPE CURR_REG=0;
DTYPE IN_A=IN.read();
        if(IN_A>CURR_REG) {
                OUT.write(CURR_REG);
                CURR_REG = IN_A;
        }
        else {
                OUT.write(IN_A);
        }
}
```

Insertion Sort: Hardware




```
void cell(hls::stream<DTYPE> &IN,
hls::stream<DTYPE> &OUT){
static DTYPE CURR_REG=0;
DTYPE IN_A=IN.read();
        if(IN_A>CURR_REG) {
                OUT.write(CURR_REG);
                CURR_REG = IN_A;
        }
        else {
                OUT.write(IN_A);

        }
}
```

```
void InsertionSort(hls::stream<DTYPE>
&INPUT, hls::stream<DTYPE> &OUT){

#pragma HLS DATAFLOW
hls::stream<DTYPE> out0("out0_stream");
hls::stream<DTYPE> out1("out1_stream");

// Function calls;
cell0(INPUT, out1);
cell1(out1, out2);

cell4(out103, OUT);
};
```