Adaptive Thresholding using Integral Images

Aditya Saripalli (20173071)
Tanmay Khabia (2018102038)

Introduction

- *Scanned document images have noise add the edges / corners.*
- *A popular technique called "Image Thresholding" is used to clear the images*
- *Digital is segmented based on the spatial illumination of the pixels.*
- *Fixed thresholding fails – if illumination varies spatially.*
- *Solution – Adaptive Thresholding (AT).*
- *One of the simple but efficient technique in AT is by using "Integral Images".*

Integral Image

- *An Integral Image is basically a summed-area table of the input image.*
- *It can be used whenever we have a function from pixels to real numbers f(x,y) (e.g., pixel intensity).*
- *Is computed as follows:*

$$I(x,y) = f(x,y) + I(x-1,y) + I(x,y-1) - I(x-1,y-1)$$

- *So we have:*

$$f(x,y) = I(x,y) - I(x-1,y) - I(x,y-1) + I(x-1,y-1)$$

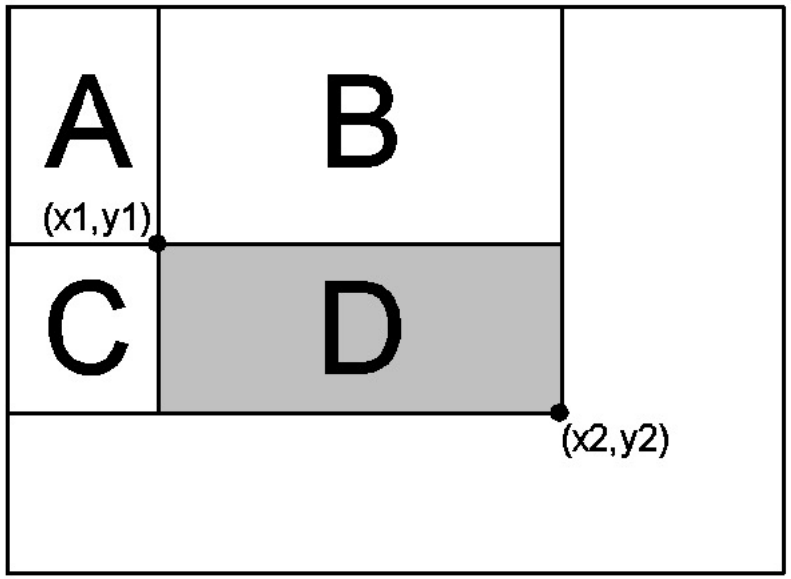
Integral Image (contd.)

- *Once we have the integral image, the sum of the function for any rectangle with upper left corner* (x_1, y_1) *and lower right corner* (x_2, y_2) *can be computed in constant time using the following equation:*

$$\sum_{x=x_1}^{x_2} \sum_{y=y_1}^{y_2} f(x,y) = I(x_2, y_2) - I(x_2, y_1 - 1) - I(x_1 - 1, y_2) + I(x_1 - 1, y_1 - 1)$$

Integral Image (contd.)

- *The figure below shows the computing sum of $f(x, y)$ over the area of the rectangle "D", which is equivalent to area computed as: (A+B+C+D) – (A+B) – (A+C) + (A).*



Integral Image (contd.)

- *A sample computation of the Integral Image for a 4x4 matrix is as shown below:*

4	1	2	2
0	4	1	3
3	1	0	4
2	1	3	2

Image

4	5	7	9
4	9	12	17
7	13	16	25
9	16	22	33

Integral Image

Adaptive Thresholding Technique

- *Adaptive thresholding using Integral Images is a two-pass technique.*
- *In the first pass we compute the integral image of the input image in linear time.*
- *In the second pass we compute the average of a S x S window of pixels centered around each pixel, in constant time and perform threshold comparison.*
- *If the value of the current pixel is T % less than this average, set the pixel as '0', otherwise set the pixel as '255'.*
- *For quality of thresholding choose T (as 15) and S (as $1/8^{th}$ of image size).*

Host Implementation

- *Used OpenCV library for all the image management activities like – reading, image to matrix conversions, serializing the images, writing the images, etc.*

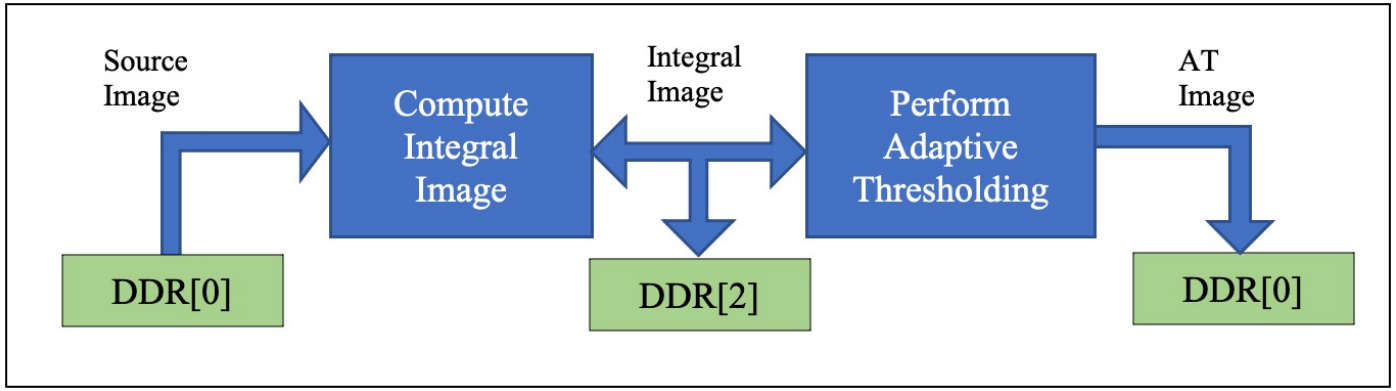
- *Computation of integral image is done using the OpenCV library function* `cv::integral()`.

- *The adaptive thresholding is performed on the integral image computed in previous step.*

NOTE: OpenCV library will pad the image with an additional row and column while computing the Integral Image. Hence the adaptive thresholding algorithm should consider this additional row and column in its computations.

FPGA Basic Version

- *Initially we tried to temporarily store the integral image inside the FPGA block memory.*
- *HW creation failed - saying that we have over utilized the BRAM blocks in FPGA.*
- *Then we slightly modified this approach, by storing the integral image in the DDR ram, after the first pass and then reading it from there in the second pass to compute the adaptive thresholding image.*



FPGA Basic Version

- *HW creation succeeded, however the run-times of the computations were not optimal.*

- *With a sample image of size 640x480 pixels, we were able to achieve the following run times with the HOST and the FPGA basic version.*

```
[ec2-user@ip-172-31-88-240 Hardware]$
[ec2-user@ip-172-31-88-240 Hardware]$ ./adaptive_thresholding afi/at_bin.awsxclbin image.bmp
--------------- Key execution times ---------------
[ET] HOST: Perform Adaptive Thresholding :    4.005 ms
[ET] Memory object migration enqueue     :    0.371 ms
[ET] OCL Enqueue task                    :    0.070 ms
[ET] Wait for kernel to complete         :   62.343 ms
[ec2-user@ip-172-31-88-240 Hardware]$
```

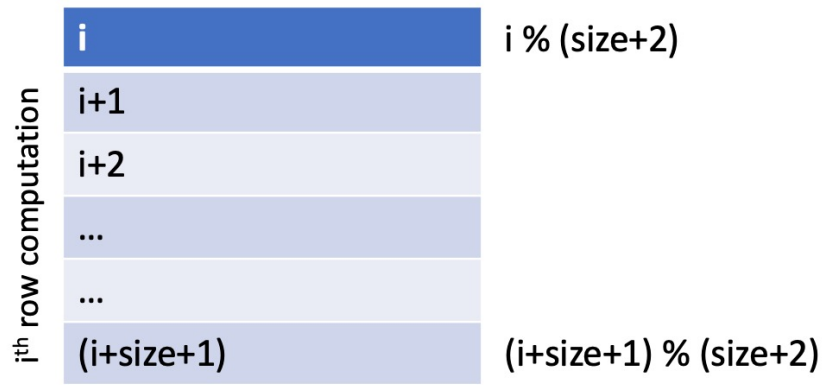
- *As we can see that the FPGA run time is almost 16 times slower than the CPU, which is clearly not optimal.*

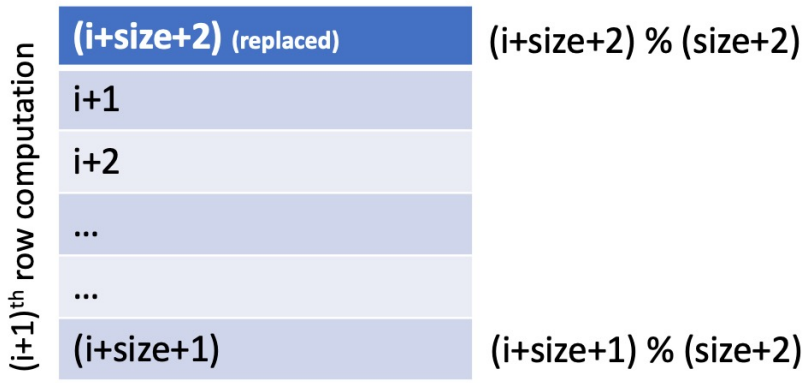
FPGA Improved Version

- *Improvised the integral image computations by considering only a strip of the input image and computing the integral image for that strip.*
- *Then by using a sliding window approach we computed the integral image of the complete image.*
- *PIPELINE the loading of source image into the BRAM and creating the integral image.*
- *PIPELINE the integral image computed at each step to compute the adaptive threshold for that portion of the input image.*
- *Implementing multiple LOOP-UNROLL sections for integral image and threshold computations.*

FPGA Improved Version (contd.)

- *For saving the integral image we only used/re-used (image_width) * (filter_size+2) size memory.*
- *Using modular computations, we were able to manage the new incoming rows by replacing them in place of the irrelevant rows at the top of the integral image.*
- *Thus, re-using the same memory and reducing the need for additional DDR memory transactions.*

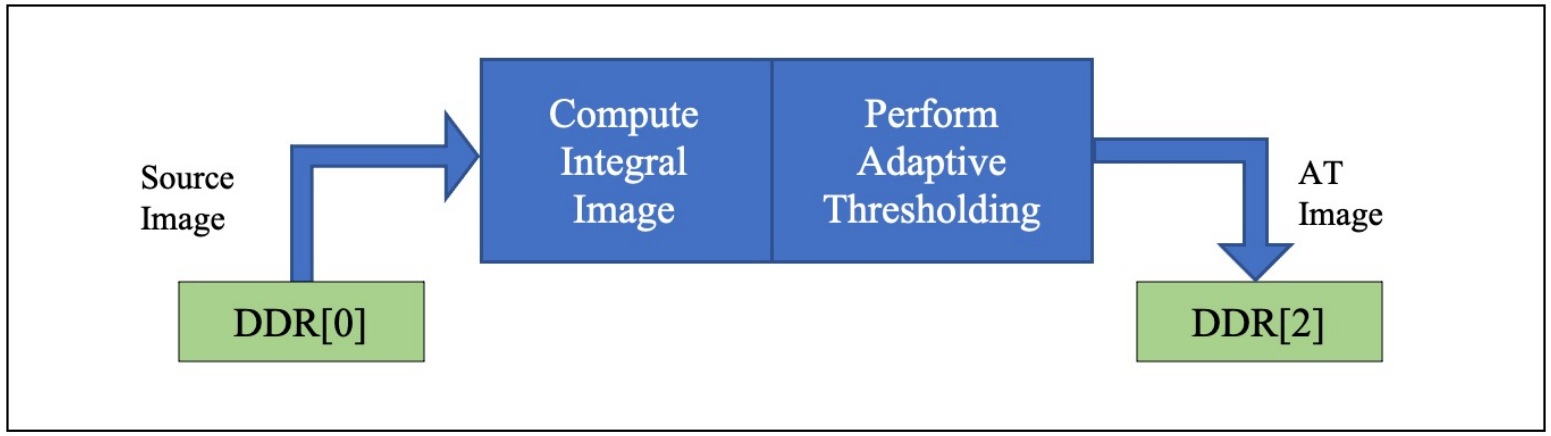




FPGA Improved Version (contd.)

- *By above approach we were able to substantially reduce the integral image memory footprint.*

- *As a result, we were able to remove the interim storage of the integral image on the DDR memory and avoid the additional data transfer latencies involved with it.*



FPGA Improved Version (contd.)

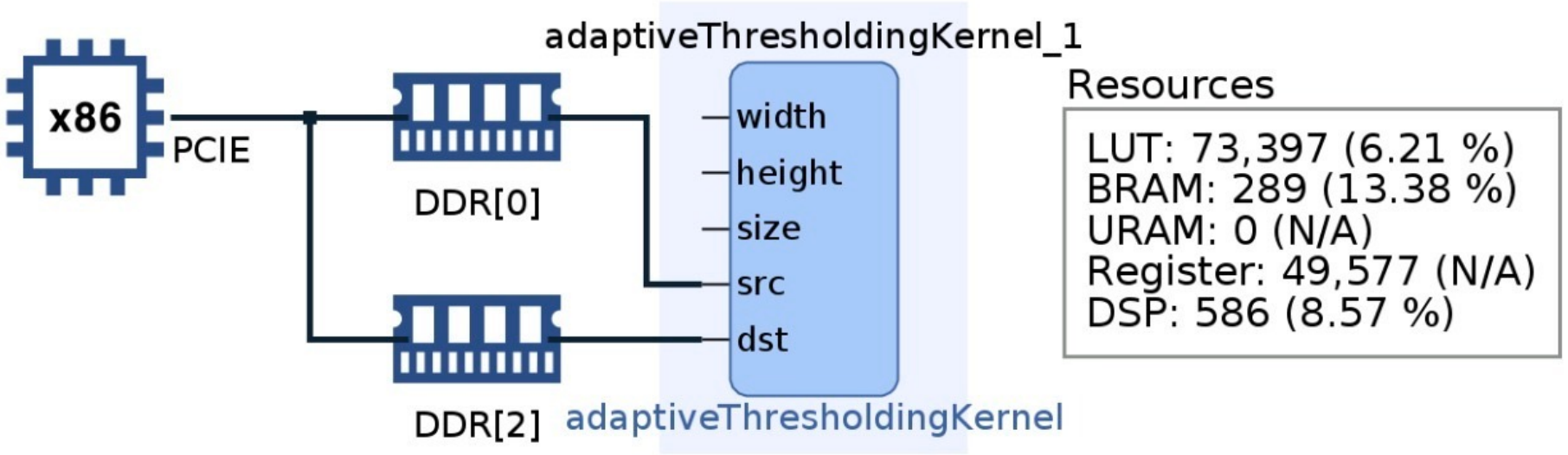
- *With the above optimizations in the FPGA kernel, we were able to get 95% improvement over the basic version of the kernel.*

- *The hardware run times (for the same image) are as shown below:*

```
[ec2-user@ip-172-31-37-15 Hardware]$
[ec2-user@ip-172-31-37-15 Hardware]$ ./adaptive_thresholding ada.awsxclbin image.bmp
--------------- Key execution times ---------------
[ET] HOST: Perform Adaptive Thresholding :    4.619 ms
[ET] Memory object migration enqueue     :    0.262 ms
[ET] OCL Enqueue task                    :    0.110 ms
[ET] Wait for kernel to complete         :    3.281 ms
[ec2-user@ip-172-31-37-15 Hardware]$
```

- *As we can see with the improved version of the kernel, we have improved the performance of the FPGA by 20 times (approx.).*

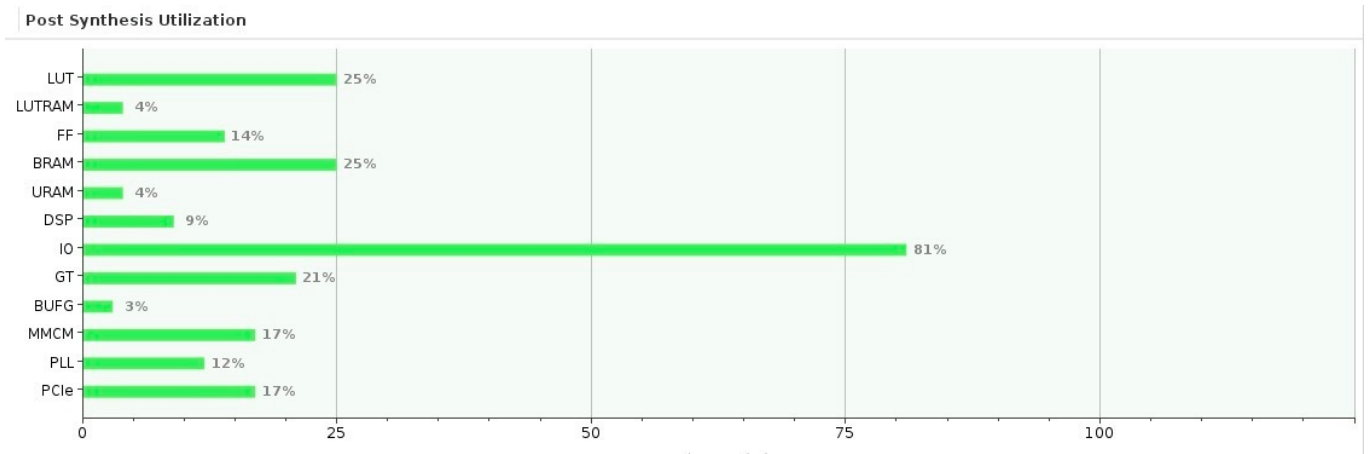
Block Design & HW Utilization



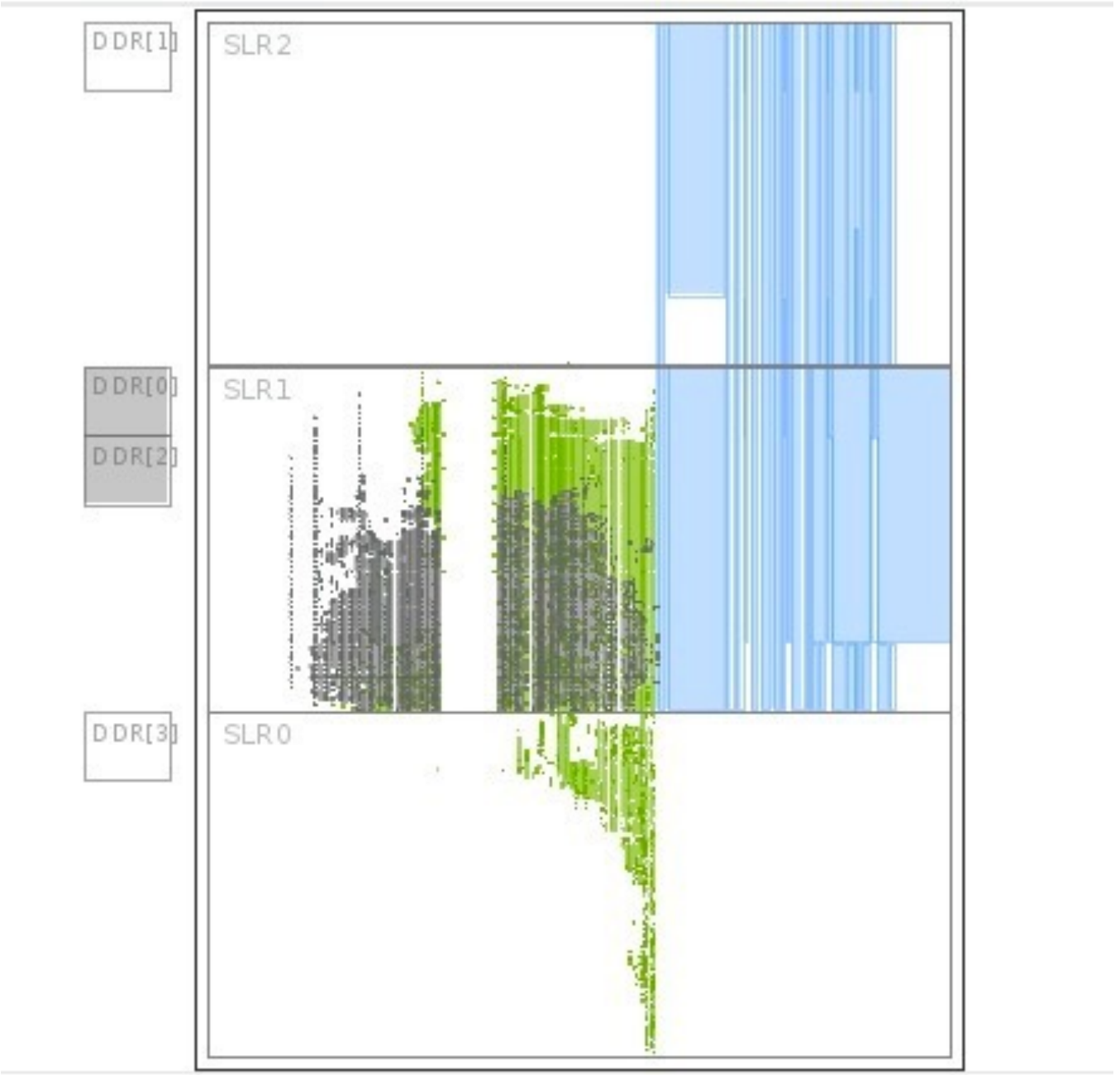
Kernel & Post Synthesis Utilization

Kernel Synthesis Utilization

Name	LUT	LUTAsMem	REG	BRAM	URAM	DSP
Platform	216499	22912	289415	258	43	6
User Budget	965269	568928	2075065	1902	917	6834
Used Resources	73397	975	49577	289	0	586
Unused Resources	891872	567953	2025488	1613	917	6248
adaptiveThresholdingKernel (1)	73397	975	49577	289	0	586
adaptiveThresholdingKernel_1	73397	975	49577	289	0	586



FPGA Dye Utilization



Roofline Analysis (Basic vs Improved)

- *Total no of computations = (12 * N * M)*
- *Total memory transfers = (27 * N * M) bytes*
- *Operational Intensity (O.I) = 12 / 27*
  = 0.44 Gops/bytes
- *Peak DDR B/W = 16 Gbps*
- *Attainable Gops = (O.I) * (Peak DDR B/W)*
  *= 0.44 * 16*
  = 7.04 Gops/sec

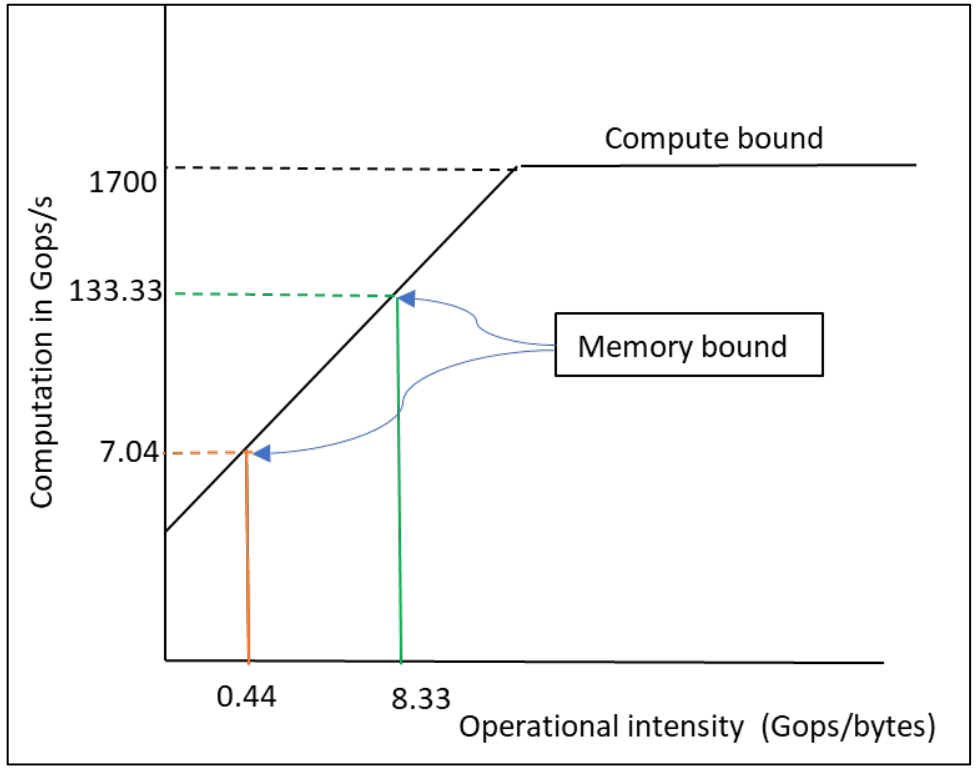
- *Total no of computations = (25 * N * M)*
- *Total memory transfers = (3 * N * M) bytes*
- *Operational Intensity (O.I) = 25 / 3*
  = 8.33 Gops/bytes
- *Peak DDR B/W = 16 Gbps*
- *Attainable Gops = (O.I) * (Peak DDR B/W)*
  *= 8.33 * 16*
  = 133.33 Gops/sec

Roofline Analysis (contd.)

- *The peak compute performance of the AWS F1 instance is = 6800 * 250Mhz = 1700 Gops.*

- *Based on the theoretical roofline analysis, the attainable performance is improved by 133.33/7.04 = **18.93 times***

- *As we can see from the experiment results, we are getting an improvement of 62.243/3.281 = **18.97 times** ☺*



Further Optimization Ideas

- *We can improve the performance & the run-times further by splitting the kernel into two different kernels.*

- *1st kernel to compute the integral image and pipeline it to 2nd kernel to compute the adaptive threshold.*

- *This approach would be very helpful in practical implementations of live streaming of videos where we need to process the frames continuously.*

Report Walk Through !!!

Repo/Code Walk Through !!!

Demo !!!

Questions ?

Thank You