# ESE5320 Homework 2

Rico Zhuang

September 12, 2025

## Answers

1. **Identify**

   1) **Scale**: this function downscales an image(stored in a 1d array "input") by only outputing every two pixel.

      **Filter**: applies a convolution filter to the image; first horizontally, then vertically, using hard-coded coefficients. It takes the input image to produce a smoothed output image.

      **Differentiate**: for each pixel, use the value of its left pixel and its up pixel to calculate an average, and output will be the difference between that pixel's value and the average.

      **Compress**: This compresses the input array using Huffman coding, packing bits into the output array one byte at a time. It returns the total number of bytes written to the output.

2. **Measure**

| Functions | $T_{\text{measured\_avg}}$(ns) | % of Total Latency | $T_{\text{measured\_avg}}$(cycles) |
|---|---|---|---|
| Scale | 2.09032e+07 | 3.64 | 7.10708+e07 |
| Filter horizontal | 1.03589e+08 | 14.57 | 3.52202e+08 |
| Filter vertical | 1.04882e+08 | 16.39 | 3.56599e+08 |
| Differentiate | 2.59741e+07 | 3.64 | 8.83120e+07 |
| Compress | 3.22591e+08 | 61.94 | 1.09681e+09 |

   1) Done

2) Done

3) Done

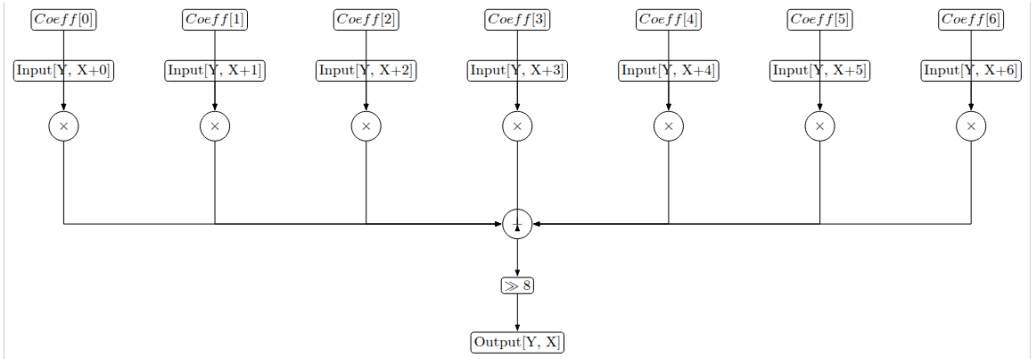
3. **Analyze**

1)

| Functions | $T_{\text{measured\_avg}}$ (ns) | $T_{\text{measured\_avg}}$ (cycles @ 1.2 GHz) |
|---|---|---|
| Scale | 8.426840e+08 | 1.011221e+09 |
| Filter_horizontal | 5.241310e+09 | 6.289572e+09 |
| Filter_vertical | 5.299610e+09 | 6.359532e+09 |
| Differentiate | 1.632450e+09 | 1.958940e+09 |
| Compress | 5.363130e+09 | 6.435756e+09 |

2) The Scale funtion has the highest latency.

3)




4) total operation count is **3.35664e+08**, and average latency is **3.35664e+08** cycles

5) I would choose **Compress** because it has the highest latency and therefore will benifit the most from speedups.

6) function Compress takes about 29% of total latency, so the maximum possible speedup is 1/(1-0.29) = **1.4**

7) Done

8) critical path length is **5 operations**

9) need two cycles for mult, 3 for add and 1 for shift, therefore lower bound is **3 cycles**

4. **Refine**

1) 6.289572e+09 cycles

| | Assembly | Annotation | Number of Executions $N$ |
|---|---|---|---|
| | add x6, x7, x12 | LOOP1 index++ | 4000 |
| | mov x4, x9 | store input size | 4000 |
| | mov x0, 0 | LOOP2 index++ | 23976000 |
| | mov w1, 0 | sum = 0 | 23976000 |
| | ldrb w3, [x4, x0] | load Input byte | 167832000 |
| | ldr w2, [x5, x0, lsl 2] | load Coefficients[i] | 167832000 |
| | add x0, x0, 1 | i++ | 167832000 |
| | madd w1, w3, w2, w1 | fused mult and add | 167832000 |
| 2) | bne .L4 | branch back LOOP3 | 167832000 |
| | lsr w1, w1, 8 | shift by 8 | 23976000 |
| | strb w1, [x6], 1 | store to output, then ++ output pointer for next one | 23976000 |
| | add x4, x4, 1 | X++ | 23976000 |
| | cmp x7, x6 | LOOP2 condition check | 23976000 |
| | bne .L7 | branch back LOOP2 | 23976000 |
| | add x7, x7, x10 | ++ to output pointer | 4000 |
| | add x9, x9, x11 | ++ to input pointer | 4000 |
| | cmp x7, x8 | LOOP1 condition check | 4000 |
| | bne .L3 | branch back LOOP1 | 4000 |

3) madd w1, w3, w2, w1 is the computation command
4) Done
5) $N_{\text{instr}} = 1174848000$
6) $N_{\text{mem}} = 3.59640 \times 10^8$
   $N_{\text{non\_mem}} = 8.15208 \times 10^8$
   $T_{\text{filter\_h\_measured}} = 6.289572 \times 10^9$ cycles
   $T_{\text{cycle\_mem}} \approx 1.63552 \times 10^1$ cycles
7) $N_{\text{slow\_mem}} = 47976007$
   $N_{\text{fast\_mem}} = 311663993$
8) $T_{\text{filter\_h\_measured}} = \frac{N_{\text{non\_mem}}}{2} + N_{\text{fast\_mem}} + N_{\text{slow\_mem}}\, T_{\text{cycle\_slow\_mem}}$
   $T_{\text{cycle\_slow\_mem}} \approx 1.16106 \times 10^2$ cycles

5. **Coding**

1) Done

2) Done

3) cdc naive: 1711894ns, cdc rolling: 39424ns. The new function takes 97% less time