Project 2 Kernel Design

Team: Dawei Feng (daweif), Jamie Wang (ziyuew2), Hamzah Ahmadi (hahmadi)

1. Kernel Description
    i. Apply Gaussian filter
        a. The first independent operation is applying the Gaussian Filter matrix on each of the pixels of the image. The second independent operation is when applying the Gaussian Filter matrix, the multiplication of each element in the Gaussian filter and the corresponding surrounding pixel values.
        b. When applying the Gaussian Filter matrix on each of the pixels of the image, we need to sum all the computed surrounding pixel values. The addition operations here are dependent.
        c. We will use functional units that can perform scaler additions and SIMD multiplications:
        For multiplication of Gaussian Filter matrix and pixel, we can use VMULPD because each multiplication is independent. When adding the values, we can use ADD.
    ii. Find the intensity gradients
        a. The first independent operation is applying the Sobel operator, which uses two convolution kernels that one is in X direction and another is in Y direction. These two kernels will be applied separately,allowing the use of SIMD instructions such as VMULPD for multiplying filter values with pixel values.
        b. After the convolution with the Sobel filter, we will have two matrices of values: one representing the gradients in the X direction G_x and one representing the gradients in the Y direction G_y. We then need to calculate: $G = sqrt(G_x^2 + G_y^2)$.
        c. The squaring of G_x and G_y is independent for each pixel and can be accelerated using SIMD instructions such as VMULPD to square the gradient values. Then we will use ADD for the addition of these two gradients. Finally, to compute the square root, vectorized square root instructions like VSQRTPD can be used, which are independent per pixel.

iii. Apply gradient magnitude thresholding
    a. The independent operations include initializing convolve, theta, and loop counter values, getting mmm value of input matrix, and comparing theta for assigning matrix value.
    b. The dependent instructions are mov for variables, cmp for loop conditions, addition for convolve vals, and VFMADDRND231PD or mmm is creating a matrix with the size of the independent operations. Then, loop through each row and column of the matrix, use compare for conditional statements and mv for their bodies. Also add for indexing. Finally, return the output image.
    c. Any address for mov, cmp and add, FMA3 for VFMADDRND231PD, FMA3 for VFMADDRND231PD, and SSE3 for divpd
iv. Apply double threshold
    a. In this step, Canny classifies each pixel's gradient as either a Strong edge, Weak edge, or Ignored edge based on the intensity gradient values and two threshold bounds provided by the user: an upper-bound and a lower-bound. If the value is between the lower-bound and the upper-bound, the pixel is classified as a Weak edge.If the pixel’s gradient value is below both the upper-bound and lower-bound thresholds, it is classified as an Ignored edge, and they are set to 0. If the pixel’s value exceeds the upper-bound threshold, it is classified as a Strong edge.
    b. For each comparison operation, it’s dependent instruction that uses cmp. And if the pixel is ignored, we can use VANDPD.
v. Track edge by hysteresis
    a. The independent operations include storing the low and high ratios in variables, calculating the difference between the image max and min, calculating ratio’d min and max, copying the image into a temp var for manipulation, and subtracting the image element by 1 for loop conditions
    b. The dependent operations include comparing max ratiod with image and copied image in loop iteration-dependent index and changing that index’s element, having a var for calculating number of weak pixels in copied image, and returning the output which is the copied image divided by its max
    c. Any address for mov, cmp and add,

2. Performance Peak
    i. Apply Gaussian filter kernel
        a. We are choosing the ece006 machine on the ECE Number Cluster. It has a x86 CPU and the CPU model is Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz
        b.

	Latency	Throughput
VMULPD	3	2
ADD	1	4

        c. Yes, the machine has those SIMD instructions
        d. Not using specialized units
        e. VMULPD is the bottleneck for the kernel
        f. The theoretical peak is 2 instructions per cycle or 8 flops per cycle
        g. The bottleneck is VMULPD and the throughput is 2 so the peak performance is 2 instructions per cycle. Since it is a SIMD instruction, there are 4 outputs for each instruction. Therefore, there are 2 * 4 = 8 flops per cycle.
    ii. Find the intensity gradients
        a. Same as i
        b.

	Latency	Throughput
VMULPD	3	2
ADD	1	4
VSQRTPD	18	8

        c. Yes, the machine has those SIMD instructions
        d. Not using specialized units.
        e. VSQRTPD
        f. The theoretical peak is 8*4 = 32 FLOPS/cycle or 8 instructions per cycle.
        g. The bottleneck is VSQRTPD and the throughput is 8 so the peak performance is 8 instructions per cycle. Since it is a SIMD instruction, there are 4 outputs for each instruction. Therefore, there are 8 * 4 = 32 flops per cycle.

iii. Apply gradient magnitude thresholding

a. same as i

b. mv, len (for getting number of rows and columns in initial image, cmp (for conditional statements), and add

	Latency	Throughput
cmp	1	4
add	1	4
VFMADDRND231PD	4	1
mov	1	4

c. same as i

d. same as i

e. VFMADDRND231PD

f. The theoretical peak is 1 instruction per cycle or 4 flops per cycle

g. The bottleneck is VFMADDRND231PD and the throughput is 1 so the peak performance is 2 instructions per cycle. Since it is a SIMD instruction, there are 4 outputs for each instruction. Therefore, there are 1 * 4 = 4 flops per cycle.

iv. Apply double threshold

a. Same as i

b. VANDPD

	Latency	Throughput
cmp	1	4
VANDPD	1	1

c. same as i

d. same as i

e. VANDPD

f. The theoretical peak is 1 instruction per cycle or 4 flops per cycle

g. The bottleneck is VANDPD and the throughput is 1 so the peak performance is 2 instructions per cycle. Since it is a SIMD instruction, there are 4 outputs for each instruction. Therefore, there are 1 * 4 = 4 flops per cycle.

v. Track edge by hysteresis

a. Same as i

b. sub for getting difference of max and min and subtracting copied image array's elements by 1 for loop range, fma for ratio'd max and min, cmp for comparing image elements with ratiod max/min or static val, and div for final output (divided copied image by its max)

	Latency	Throughput
cmp	1	4
sub	1	4
VFMADDRND231PD	4	1
divpd	20	20
mov	1	4

c. same as i

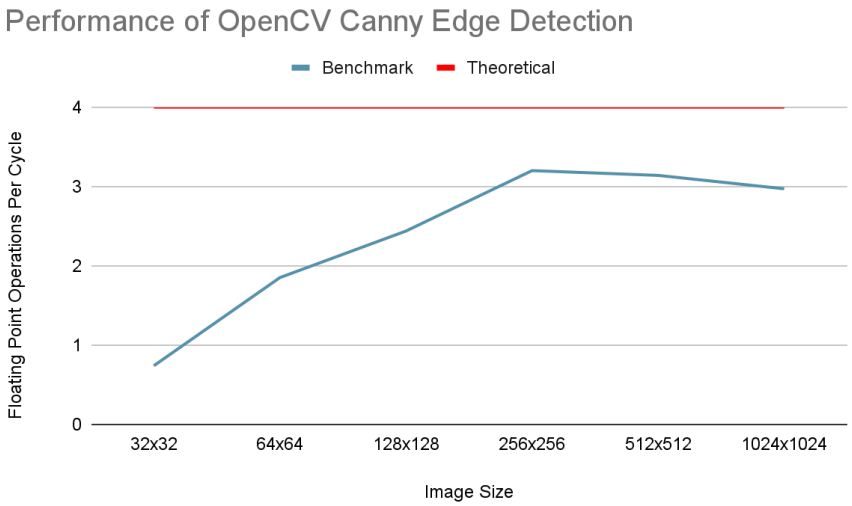
d. same as i

e. VFMADDRND231PD

f. The theoretical peak is 1 instruction per cycle or 4 flops per cycle

g. The bottleneck is VFMADDRND231PD and the throughput is 1 so the peak performance is 2 instructions per cycle. Since it is a SIMD instruction, there are 4 outputs for each instruction. Therefore, there are 1 * 4 = 4 flops per cycle.

3. Performance Baseline



i.

The data points above are collected from the ECE006 machine. The images are downloaded from the Coco dataset(https://cocodataset.org/#home) and then they are scaled into 6 different sizes. For each size, there are 10 different images and we repeat the benchmark on those 10 images 100 times. Then, we can divide the total number of floating point operations by the total number of CPU cycles to get the final values.

ii. The theoretical peak should be 4 FLOPS per cycle. When the image size is less than or equal to 1024x1024. The performance is lower than the theoretical peak meaning that we can still improve the performance.

4. Design of implementation

i. The gaussian filter kernel has one output, an array that will make up the resulting image matrix. The kernel for finding the intensity gradient has one output, a matrix array calculated using sobel kernel. The third kernel, apply gradient magnitude thresholding, also has one output, which is a matrix array using the previous output to create a duplicate matrix array, but changing the values that have an insignificant difference from its neighboring values to 0. The fourth kernel, double threshold, simply outputs the resulting image matrix array after changing its values depending on its corresponding index in that previous kernel's output compares to the strong and weak thresholds. The final kernel has one output obviously, the resulting image matrix

ii. Our machine has 16 registers since it is 64-bit. Intel processors have eight general purpose registers in 32-bit mode, and sixteen general purpose registers in 64-bit mode, however, from the internal hardware perspective, Intel processors have many more registers.

One int is 4 bytes, one register is 8 bytes, we can fit two ints in one register. We can have 32 ints. We will be running one kernel at a time, our largest kernel will require 8 registers, since it has 15 variables and it can be placed by groups of two in 7 registers and one more register having the one remaining var and arrays in C++ are stored in memory.

Iii. Since all outputs are arrays, they are stored in memory. Our initial inputs are a mix of arrays and integers for thresholds and kernel size (width, height). Inputs that are used in multiple of our kernels will hold a place in a register for multiple kernels. The algorithm uses some variables in multiple kernels (like convolve in

both Gaussian and gradient finding steps) so some of our registers will not be changed for multiple kernels. The majority of the registers will be continuously replaced since temporary variables are only used for calculation in that kernel and some registers will not even be used since we do not require the 32 registers for any kernel

iv. Many details were provided in the answer to question 1 but our project is on edge Detection, which is a process separated into five steps that are all dependent on the previous steps output. These five steps are applying gaussian filter, which starts with inputted source matrix (memory), its width and height (register 1), upper and lower thresholds (register 2), and possibly an empty answer array (memory). Its algorithm loops through the width and height (register 3: counters for loops), and a few variables for calculation (registers 4/5). The next step, gradient find, keeps registers 1 and 2 and replaces registers 3 with the convolve values and, register 4 can be replaced with new counters for looping through the width and height again. Register 5 can be used for nested loop counters for getting convolve for offset matrix of element. Register 6 can be used for holding theta and segment value. The values of two matrix arrays that will be used for later calculations are dependent on the calculations using theta and segment; these arrays are stored in mem. Step three, non-maxima suppression, uses these arrays and the initial width and height to loop through them (loop uses two ints as a counter which can be stored in register 3 since we no longer need the convolve values) and stores values for threshold calculation in a new array (stored in mem). Step four, double threshold. can replace register 4 again since we loop through the matrix width and height again and don't need the previous counters that are currently stored in it. In this loop, we use the matrix multiplication value from the source matrix (stored register 5 now either low or high doesn't matter; we don't need the loop counters that were previously in there) to store a value in the output matrix. For the final stage, hysteresis, We are looping through the output matrix so we need counters for the nested loop, which can be stored in register 4 removing the previous counters and replacing register 5 using the same logic and reason as the previous stage. FInally, we output the matrix.