18-645 Final Report

Weixin Liu, Changhao Yang, Hong Chen

1. Introduction

This project is to implement and optimize a feature detection algorithm that is called Scale-invariant feature transform (SIFT), which came up in 2004 by D.Lowe [1]. The algorithm performs reliable recognition and provides a robust feature description. It is widely used in object recognition, robotic mapping, and navigation. As the implementation of SIFT algorithm is available in a popular open-source library OpenCV [2], OpenCV is taken as the baseline.

This algorithm involves multiple convolutions in different scales and linear algebra intensive computations, resulting in slow processing time. Optimizing this algorithm and reducing processing time is in great demand in industry and research.

2. Algorithm

SIFT is an involved algorithm with multiple steps, including computing Difference of Gaussian (DoG) of images in different scale-space, extrema identification, keypoint localization, orientation assignment, constructing keypoint descriptor and keypoint matching. Due to SIFT's complexity, this project only focuses on optimizing the first part: computing DoG of images in different scale-space.

The input data of this algorithm is a two-dimensional grayscale image. To construct the scale space, the input image is scaled into *m* various sizes and then be convoluted with *n* Gaussian filters with different variances. resulting in *n***m* Gaussian-blurred feature maps.

For this step, *n***m* convolutions are required in total, which is computationally expensive. As Fig.1 shows, the Gaussian filter is a separable filter and the 2D filter is decomposed into two one-dimensional filters applied consecutively. So the Gaussian filter, with a window size of *k***k* (so *k***k* multiplications per pixel) can be reduced to a single-column

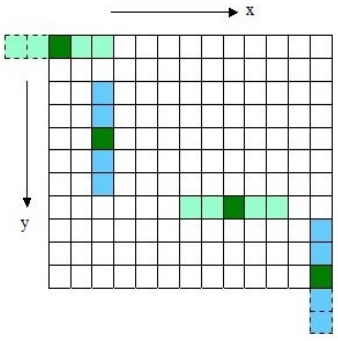

Figure 1 Separable Filter

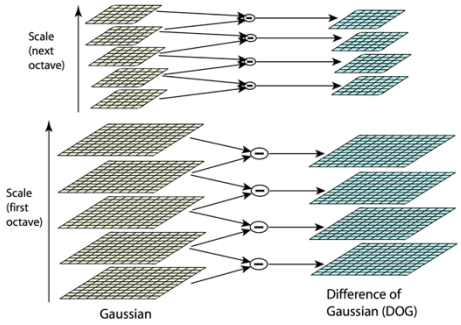

Figure 2 SIFT drawing

filter with a size of *k*1* and a single-row filter with a size of *1*k*. That means *2*k* multiplications and additions for each pixel.

Then, DoG maps are constructed by subtracting every two adjacent Gaussian-blurred feature maps. This process is done for every scale of the image in Gaussian Pyramid. It is represented Fig.2. For each octave of scale space, the image is repeatedly convolved with Gaussian filters with different variances to produce a Gaussian Pyramid shown on the left. Adjacent Gaussian feature maps in Gaussian Pyramid are subtracted to produce the difference-of-Gaussian images on the right. After each octave, the Gaussian image is down-sampled by a factor of 2, and the process repeated.

The Gaussian feature map of an image is defined as a function, L(x, y, kσ):

$$L\left(x, y, k\sigma\right) = G\left(x, y, k\sigma\right) * I\left(x, y\right)$$

which is the convolution of the input image I(x, y) with the Gaussian blur G(x, y, kσ) at scale kσ. Then, a DoG image D(x, y, σ) is given by

$$D\left(x, y, \sigma\right) = L\left(x, y, k_i\sigma\right) - L\left(x, y, k_j\sigma\right)$$

3. Analysis

3.1. Theoretical peak analysis

From the algorithm, it can be derived that there are two sequential steps in computing DoG of images: convolution with Gaussian filter and matrix-matrix subtraction.

Assume the size of the Gaussian filter is *k*k* and the size of one input image is *N*N*.
In one dimensional convolution, each pixel in the output feature map is the summation of k multiplications, which is one dependent chain of *k* FMA operations. Such summation-multiplication is called filter operation. And the FMA operations in one filter operation are independent of those in another filter operation. So there are *k* FMA operations in one chain and *N^2* independent chains in every convolution.

In matrix-matrix subtraction, the subtraction operations between the pixels in two feature maps are independent. So there is 1 subtraction operation in one chain and *N^2* independent chains in every image subtraction. The number of subtractions is defined by the size of scale space and the number of Gaussian variances.

The cycles required for sequential operations are the number of operations multiplied by their corresponding latencies. The theoretical peak is computed by dividing the independent operations by throughput and adding it up with cycles required for sequential operations. As the two convolutions are applied sequentially and the throughput of FMA operation and SUB operation are both 1, there are *2*k*N^2* cycles for

each image in the first step and *N^2* cycles for every two adjacent feature maps. (All operations stated above are single-precision float point operations.)

The overall theoretical peaks are obtained by summing up the time used in all steps. So, the theoretical peak of Gaussian Blur is *2*∑ki∑Nj^2*, where *ki* denotes the variance of i-th Gaussian filter in each scale and *Nj* denotes the size of the image in j-th scale. And the theoretical peak of matrix-matrix subtraction is *(p-1)*∑Nj^2*, where *Nj* denotes the size of the image in *j-th* scale and *P* denotes the number of the variance of Gaussian filter in each scale. It can be derived that the majority of the time consumed is in Gaussian Blur. The bottleneck of the algorithm on a regular desktop/laptop is float FMA.

3.2. OpenCV Analysis

One problem of the OpenCV implementation is that when making padding for matrices in convolution it allocates a new padded matrix. The memory allocation and access are very time-consuming, and the overhead can be potentially reduced. Also, the OpenCV implementation does not optimize for the specific hardware of users. Specifically, for convolution, it is better to fill up the pipeline for the fused multiply-add functional units. Based on the benchmark information, we are supposed to design kernels for the particular machine.

4. Kernel Design

Since this algorithm is divided into different parts, we design different kernels for each part. The data type used in this algorithm is float and our machine supports up to AVX2 whose SIMD registers have a size of 256 bits and can hold 8 floats.

4.1.1. Convolution kernel

One of the flaws in OpenCV implementation is that it requires to create a new matrix with padding. This creation leads to the overheads of allocation and accessing memory. Our strategy is to use index mapping when reading data to avoid these costly overheads.

As discussed in the previous sections, the convolution with a Gaussian filter can be decomposed into a convolution with a column filter followed by another convolution with a row filter. The first technique utilized is vectorizing the convolution with SIMD to speed up the computation. To obtain more independent operations, we should compute the outer product instead of the dot product. The broadcast-based method is used because it requires fewer cycles compared to the permute-shuffle-based method. The minimum number of independent chains in order to avoid any bubbles in FMA pipelines is 2*4 = 8 because the number of functional units of FMA is 2 and its latency is 4 in our machine. So, we define our kernel size as 4 by 16 floats. Since the filters in this algorithm have sizes larger than 4, the filters will need to be chunked. That is, for column filter, the horizontal SIMD registers load the data from the image, the vertical SIMD registers

broadcast the elements in the kernels. After computing the outer product, the kernel moves along the column and repeats this process until it completes the computation for one filter operation. The drawing of the kernel is shown in figure 3.

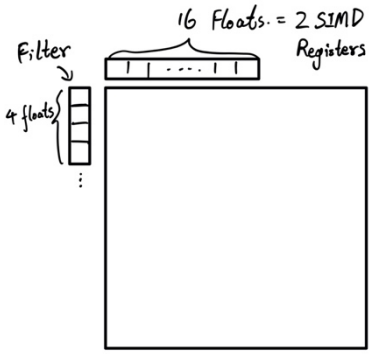

Figure 3 Convolution kernel

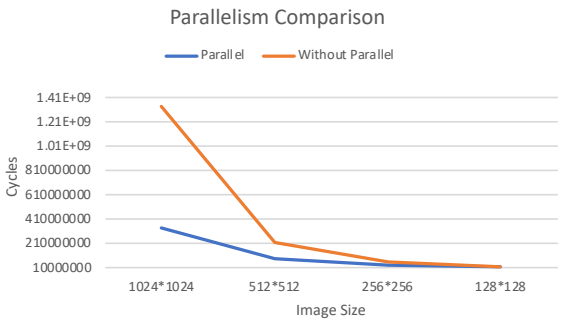

Figure 4 Parallelism Comparison

This kernel discussed above can easily adapt the idea of index mapping because, for column filters, the padding is added along the column. When we need to load data from the image, we can simply compute the mapped index and load the data from the mapped index. However, for row filters, we need to load data into SIMD registers one-by-one instead of using intel intrinsic because the data is stored in row-major order. Loading SIMD registers one-by-one can be time-consuming. Therefore, we propose a transpose-based algorithm: first, convolve the image with column filters and store the transpose of the result; second, repeat the last step. This algorithm avoids loading the data one-by-one into the SIMD.

4.1.2. Techniques Used

As mentioned above, the first technique used is SIMD. The second technique used in our algorithm is parallelism. Since our machine has 4 cores and each core has two hardware threads. We create 8 threads for parallelism. Figure 4 shows that parallelization is more effective in this problem as the input sizes increase. As for small input sizes, the overhead of creating threads takes a great portion of the time and we do not observe any significant performance improvement. When input size is large, parallelism leads to much better performance because the benefits due to parallelism start to surpass the overheads. After parallelization, the code runs 4 times faster than the unparalleled version at the input size of 1024*1024.

The next technique used is loop interchange. The original iteration order of the filter is looping along the column in the outer loop and looping along the row in the inner loop. This normally leads to fewer cache misses because the matrix is stored in row-major order. However, we are storing the transpose of the convolution result, looping along the row will require storing the result along the column which causes cache misses. In addition, our kernel has a width of 16 floats which equals the cache-line length of our

machine. Interchanging the loop will lead to little performance drop compared to the performance improvement caused by more cache hits when storing the result. Figure 5 shows the performance of the loop-interchanged version and the one without loop interchange in different image sizes. Loop interchange results in about 40% improvement in all image sizes.

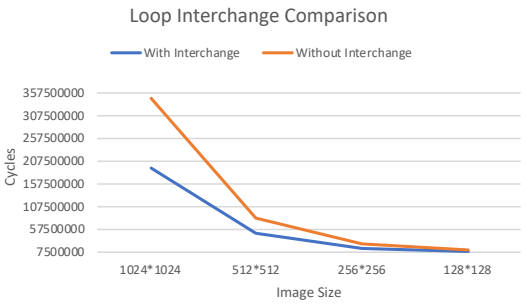

Figure 5 Loop Interchange Comparison

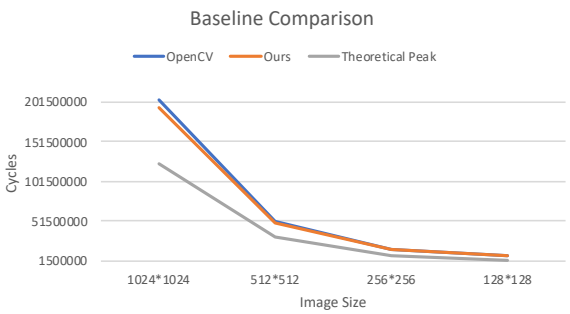

Figure 6 Baseline Comparison

4.1.3. Comparison with Baseline and Theoretical Peak

The performance of our kernel, baseline, and theoretical peaks are shown in figure 6. Our kernel is about 10% faster than the baseline performance in all image sizes. Besides, we achieve about 50% of the theoretical peak. The gap between the theoretical peak and our kernel performance may be due to the additional time required to access memory including loading data and storing the data.

4.2.1. Subtraction kernel

Constructing DoG feature maps is implemented by matrix-matrix subtraction. For our machine, the latency of SIMD subtraction for floating-point is 4 cycles, throughput is 1 and the number of functional units is 1. A two-dimensional matrix could be viewed as a one-dimensional array. We can perform subtraction on contiguous memory. Since the matrix is stored in row-major order, it is already optimized to avoid cache misses. To fill up the pipeline, the kernel is designed to hold at least 32 floating points. Therefore, the kernel size of subtraction is 1 by 32 floats.

4.2.2. Techniques Used

The first technique used in subtraction is SIMD. Instead of using simple for-loop to subtract each element one by one, SIMD subtraction performs 8 float subtraction simultaneously, which makes computation more effective. Besides, since the latency of SIMD subtraction is 4 and the functional unit is 1, four SIMD instructions are needed to fill up the pipeline. After SIMD is used, figure 7 shows the result of the improvement. When image size is 1024*1024, it is about 2 times faster than before. The second technique used in this part is parallelism. Because all subtraction is independent, every thread can be in charge of one matrix-matrix subtraction.

4.2.3. Comparison with Baseline and Theoretical Peaks

The OpenCV implementation already includes SIMD instructions and parallelism, so our implementation does not have much improvement compared to the baseline performance. As figure 8 shows, our implementation is about 10% faster than the baseline code. This is because we use enough operations to fill up the pipeline which makes the functional units busy all the time.

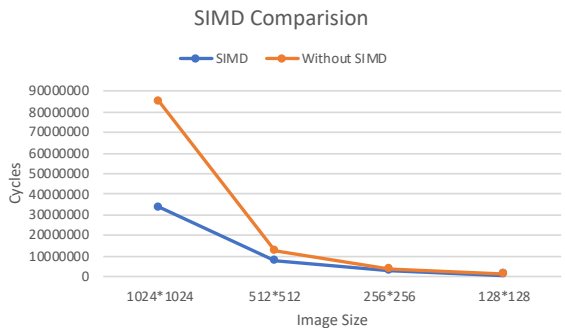

Figure 7 SIMD Comparison

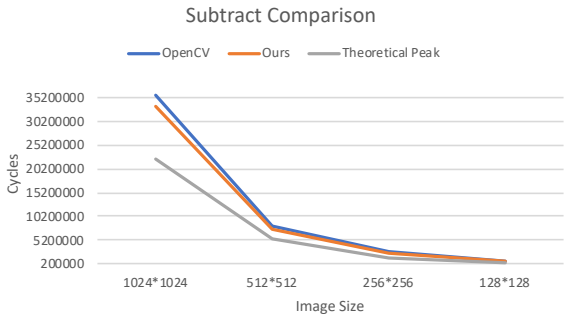

Figure 8 Subtract Comparison

5. Future Improvement

One future improvement is to apply strip mining to avoid cache misses. Since the filters are moving with a strip of 1, the data brought into the cache in the previous filter operation are still needed for the next filter operation. However, the current implementation might kick out the cache before executing the next filter operation. Another future improvement is instruction interleaving. Since the kernel, first, loads all the required data and, then, executes the FMA instructions, interleaving them can potentially avoid idling of FMA when loading the data.

6. Conclusion

This project focuses on optimizing the Gaussian blur and matrix-matrix subtraction in SIFT transform. In particular, the designed kernel for Gaussian blur is a convolution with column filters with a kernel size of 4 by 16 floats. SIMD, parallelism and loop interchange are implemented to optimize the kernel, resulting in a performance that is 10% faster than baseline and achieves 50% of the theoretical peak. The designed kernel for the matrix-matrix subtraction is 1 by 32 floats SIMD subtraction. It applies SIMD and parallelism to optimize as well. The results show that it is about 10% faster than the baseline and achieves 67% of the theoretical peak.

7. Github Link

https://github.com/stevenliu000/18645_Project/tree/master

Source code is also included along this report.

Reference

[1] D. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints", International Journal of Computer Vision, vol. 60, no. 2, pp. 91-110, 2004. Available: 10.1023/b:visi.0000029664.99615.94.

[2] O. Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge", International Journal of Computer Vision, vol. 115, no. 3, pp. 211-252, 2015. Available: 10.1007/s11263-015-0816-y.