Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 8, Number 1, pages 35–38, January 2019

Vectorization of Kernel and Image Subsampling in FIR Image Filtering

Teppei Tsubokawa, Yoshihiro Maeda, and Norishige Fukushima
Nagoya Institute of Technology, Nagoya, Japan

***Abstract*—Image subsampling is a traditional algorithm for accelerating image processing. The subsampling natively causes aliasing; thus, we use randomized sampling to moderate the issue. Also, SIMD vectorization speeds up processing by computing multiple data with a single instruction by hardware acceleration. However, randomized sampling is not suitable for SIMD vectorization. In this paper, we accelerate image filtering by subsampling filtering kernel and images by randomized algorithms. Also, the subsampling is vectorized by vector addressing. We describe how to implement subsampling of only images, the only kernel, and subsampling images and kernels. Also we compare where loop is unrolled is useful. Experimental results show that vectorization of kernel loop unrolling is faster than pixel loop unrolling. Also, kernel and image subsampling is effective for acceleration of bilateral filtering which a smoothing parameter of a range kernel is large.**

***Index Terms*—FIR image filter, SIMD, pixel subsampling, kernel subsampling**

I. Introduction

Image processing often requires real-time processing. Particularly, with the spread of high-resolution images such as 4K images and 8K images, the demand is increased. It is difficult, however, for usual algorithms to execute high-resolution images in real-time. Along with the development of computer architecture, we can accelerate processing by various arithmetic units.

One of them is vectorization of single instruction, multiple data (SIMD). SIMD speeds up by computing multiple data (vectors) with a single instruction. There are MMX, SSE, AVX-512, AltiVec, and NEON as the SIMD instruction set in the CPU. Also, the vector length, which is the number of data that can be handled simultaneously by SIMD, has been increased with the development of CPU architecture. As a precondition of this SIMD instruction, it is necessary that the alignment of the data arranged in the memory space match. Although it is possible to process even when the alignment is not met, it costs more than when matching. Also, if the loading of data is not sequential, it will cost more.

Especially in image processing, technology is necessary for application because it handles three-dimensional information of space and color. In SIMD, there is a method that called loop unrolling which simultaneously processes multiple elements by unrolling a loop.

For accelerating image processing by algorithm, downsampling images is a simple and general approach. The processed images are then upsampled. The approximation accuracy is low with the method. In for image convolution, it is effective for performance between accuracy and speed to subsample filtering kernels than image itself. However, the aliasing problem is caused by the downsampling.

Randomization of the sampling moderate the problem. Also, kernel subsampling with also image subsampling improve the performance. The randomization of subsampling with vectorization is, however, not a travail problem.

In this paper, we implement the image and kernel subsampling with SIMD vectorization. The contribution of this paper is as follows;

- we combine the kernel subsampling approach with image subsampling with randomization for accelerating image filtering.
- we effectively vectorize the random access processing in the proposed approach.

II. FIR Filtering and Its Acceleration

A. FIR Filtering

General filters for images are divided into FIR type filters and IIR type filters. In particular, the FIR type filter is a typical image processing. The definition formula of the FIR filter is shown in Equation 1

$$\bar{\boldsymbol{I}}(\boldsymbol{p}) = \frac{1}{\eta} \sum_{\boldsymbol{q} \in \boldsymbol{N}(\boldsymbol{p})} \sum_{c \in \boldsymbol{\lambda}} f(\boldsymbol{p}, \boldsymbol{q}) \boldsymbol{I}(\boldsymbol{q}, c), \tag{1}$$

where $\boldsymbol{I}$, $\bar{\boldsymbol{I}}$ is input image and output image, $\boldsymbol{p}$, $\boldsymbol{q}$ is the target pixel and the reference pixel, f is weight function, N is a function that returns a set of reference pixels in the kernel, λ is collection of color channels, and η is a term for normalization.

B. Acceleration for FIR Filtering

As a speeding up method in the FIR filter, there is a variable separating type filter [3] and subsampling pixels. In variable separating type filter, a function of the weight is separated in the vertical direction and the horizontal direction. In FIR filter, computation of $O(r^2)$ is required for each pixel. By separating this in the vertical direction and horizontal direction, it can be suppressed to the calculation of $O(r)$.

In the approximate speeding up method of thinning pixels, downsampling the input image, filtering it, upsampling the result. Since the input image to the filter is smaller than the original input image, it is possible to perform the processing at high speed as compared with the case of the original input image. The approximation accuracy at this time depends on the subsampling method and how much subsampling is done. There is also the approach to subsampling the kernel.

C. Importance Image Subsampling

In this section, we describe a method for speeding up the FIR filter by subsampling the pixel. We use the method of *Image Perforation* [5]. There are two kinds of methods for subsampling the pixels: thinning pixels regularly and thinning pixel irregularly. The method of thinning out pixels regularly is called grid sampling, which simply thin out pixels in a lattice form so that complex preprocessing and the like are unnecessary and simple linear interpolation is applicable when upsampling. Performing processing at high speed is possible. However, since the image is thinned out without considering the characteristics of the input image, accuracy may be remarkably low depending on the type of the filter. In this paper, we use importance sampling as a method to thin out pixels irregularly. Importance sampling relies on an importance map to determine where to place samples. In general, it is infeasible to compute an optimal importance map. So, we use the image textureness measure of Bae et al. [1]. For efficiency, we modify their method to use a simple Gaussian blur instead of a bilateral filter. Our importance sampling method works by simply remapping the intensities of the importance map followed by dithering. We remap the intensities of the importance map, so that is sums to the target number of samples, fn, after begin clamped to the range [0,1]. This remapping can be done using histogram operations. We then use dithering to determine the final sample locations, as shown in Figure 1. The definition formula of the FIR filter using importance sampling is shown in Equation 2.

$$\bar{\boldsymbol{I}}(\boldsymbol{p}) = M(\boldsymbol{p})\frac{1}{\eta}\sum_{\boldsymbol{q}\in\boldsymbol{N}(\boldsymbol{p})}\sum_{c\in\boldsymbol{\lambda}} f(\boldsymbol{p},\boldsymbol{q})\boldsymbol{I}(\boldsymbol{q},c) \qquad (2)$$
$$(M(\boldsymbol{p}) = \{0,1\}),$$

where $M(\boldsymbol{p})$ is the value of importance map at the target pixel. When we use subsample the pixel using importance sampling, there are black pixels in the output image like a Figure 2. So, we interpolate using IIR type Laplacian smoothing filter as a method of upsampling when pixels are subsampled using importance sampling. The formula of upsampling is shown in Equation 3.

$$\bar{\boldsymbol{I}} = \frac{\boldsymbol{L} * \boldsymbol{I} \cdot \boldsymbol{M}}{\boldsymbol{L} * \boldsymbol{M}}, \qquad (3)$$

where $\boldsymbol{L}$ is Laplacian smoothing filter, and $\boldsymbol{M}$ is importance map.

D. Randomized Kernel Subsampling

As an approach similar to the method of subsampling pixels, there is a method of increasing the speed by subsampling reference pixels in the kernel. Even when subsampling the kernel as well as pixel subsampling, there is also method subsampling regularly or irregularly. When the kernel is thinned regularly in the form of a grid, although it is easy to implement, there is a possibility that aliasing occurs. Therefore, as shown in the Figure 3, it is possible to improve the accuracy by randomly selecting reference pixels in the kernel and creating a plurality of patterns. Equation 4 shows an expression for randomly selecting reference pixels in the kernel in the FIR filter.

(a) Input (b) Importance map

Fig. 1: Importance map for image subsampling.

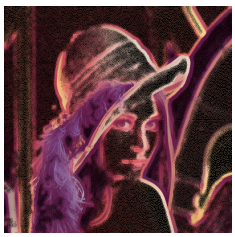


Fig. 2: Resulting image of bilateral filtering with importance image subsampling.

$$\bar{\boldsymbol{I}}(\boldsymbol{p}) = \frac{1}{\eta}\sum_{\boldsymbol{n}}\sum_{c\in\boldsymbol{\lambda}} f(\boldsymbol{p}, R(\boldsymbol{p}))\boldsymbol{I}(R(\boldsymbol{p}), c), \qquad (4)$$

where R is a function to randomize reference pixels from $N(\boldsymbol{p})$, and n is the number of pixels to be referred to.

III. Vectorized Implementation

A. Implementation to Subsample Pixel and Kernel

In the FIR filter, there are five loops. There is a loop (horizontal and vertical double loop) for scanning the target pixel, and a loop (horizontal and vertical double loop) for scanning the reference pixel in the kernel, and a loop for scanning the color channel of the reference pixel. In this paper, we call this pixel loop, kernel loop, color loop in order. In this paper, we don't implement using the color loop but implement using pixel loop and kernel loop.

B. Implementation to Subsample Pixel

In this section, we describe how to load the target pixel and the reference pixel when subsampling the pixel, then perform the processing and store the result. It is assumed that the vector length of SIMD is 8.

First, we describe an implementation in the case of subsampling the pixel using pixel loop unrolling. In the original pixel loop unrolling, sequential pixels can be handled as vectors, and reference pixels corresponding to these target pixels can be loaded as vectors for calculation. However, when subsampling pixels, the pixel to be sampled is not sequential data, so in this paper, we create a LUT for the offset of the pixel to be sampled, and when we load the target pixel, we first load 8 LUT data. Then, using this as an index, we load 8 pixels

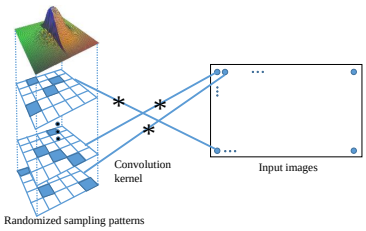


Fig. 3: Overview of randomized kernel subsampling.

using the set instruction or the gather instruction. This method is called non-vectorized load pixel loop unrolling. Then, load reference pixels corresponding to these target pixels. When storing, we store the elements of the vector one element at a time or store using the scatter instruction. When the number of pixels to be sampled is not a multiple of 8, the remaining pixels need to be calculated by the non-vector operation.

Next, we describe an implementation in the case of subsampling the pixel using kernel loop unrolling. In the kernel loop unrolling, load one target pixel using the LUT of offset to the pixel to be sampled, and store the result of the processing. In original kernel loop unrolling, reference pixels in the kernel are loaded as vectors and calculation is performed. Therefore, the width of the kernel needs to be a multiple of the vector length. In this paper, we use a method called non-vectorized load kernel loop unrolling. In this method, since a plurality of reference pixels are collected and made into a vector, it is possible to adapt to the case where the shape of the kernel is different for each target pixel or kernel thinning. We use non-vectorized load kernel loop unrolling when use kernel loop unrolling.

C. Implementation to Subsample Kernel

When subsampling the kernel, as mentioned in II-D, to suppress aliasing, the number of reference pixels is thinned out by using a random algorithm. There are two kinds of implementation methods when we select reference pixels randomly. One is a method of dynamically selecting reference pixels randomly, and the other is a method of using LUT. The former method is simple and intuitive. However, if we use a complicated random sampling pattern, the calculation cost will be high. On the other hand, in the case of the method using the LUT, since the random pattern is calculated in advance and stored in the LUT, the calculation cost is can be reduced. In a paper [2], reduce the size of a binary mask representing the reference pixel to be sampled and store in the LUT using Carley-Patterson method [4], [6]. However, the amount of memory used is still not sufficiently reduced, and as the filter size increases, the amount of memory used increases. In that case, since the number of reference pixels to be sampled doesn't change greatly, the LUT is redundant. Therefore, in this paper, we stored only the coordinates which were randomly selected in the kernel. We introduce LUT whose memory usage depends only on the number of reference pixels. In this LUT, coordinates are stored in raster order to improve cache efficiency.

First, we describe the implementation using pixel loop unrolling. When the target pixel is loaded with 8, and the reference pixel is randomly selected using the same random pattern for each pixel, the relative position from the target pixel to the reference pixel is fixed every 8 pixels. In such a case, since pixel values are likely to be similar for every 8 pixels, streaking noise may occur. Therefore, when loading reference pixels, we load offsets to reference pixels which are different random patterns for each target pixel from the LUT using the gather instruction. Furthermore, we use this as an index and load reference pixels using the gather instruction and perform processing. We store it sequentially using the store instruction.

Next, in the case of implementation using kernel loop unrolling, we load one target pixel, and load 8 reference pixels using the gather instruction and perform processing.

D. Implementation to Subsample Image and Kernel

In this section, we describe implementation to subsample the pixel and kernel. At this time as well as II-D we use a random algorithm for subsampling the kernel. In the case of pixel loop unrolling, we load 8 target pixels using the gather instruction. When loading reference pixels, we load reference pixels using the gather instruction twice as well as II-D of pixel loop unrolling. When storing, we store one element at a time or use the scatter instruction.

Next, in the case of kernel loop unrolling, we load one target pixel and load 8 reference pixels for it using the gather instruction.

IV. Experimental Results

We use a bilateral filter in the experiment. Bilateral filter is one of the edge-preserving smoothing filters. The following equation calculates the weight of a bilateral filter.

$$f(\boldsymbol{p},\boldsymbol{q}) = \exp\left(\frac{\|\boldsymbol{p}-\boldsymbol{q}\|_2^2}{-2\sigma_s^2}\right)\exp\left(\frac{\|\boldsymbol{I}(\boldsymbol{p})-\boldsymbol{I}(\boldsymbol{q})\|_2^2}{-2\sigma_c^2}\right) \quad (5)$$

where $\|\cdot\|_2$ is L2 norm, σ_s is space standard deviation, and σ_c is range standard deviation. Because the bilateral filter is

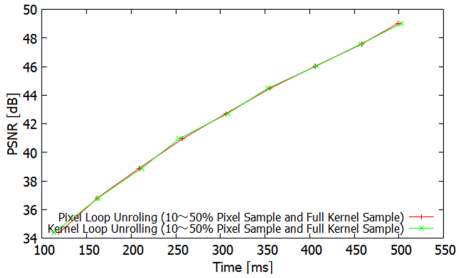


Fig. 4: Comparison between pixel loop unrolling and kernel loop unrolling when subsampling only pixels.

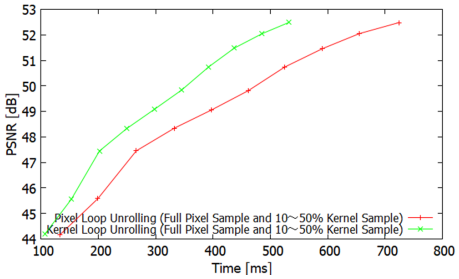


Fig. 5: Comparison between pixel loop unrolling and kernel loop unrolling when subsampling only the kernel.

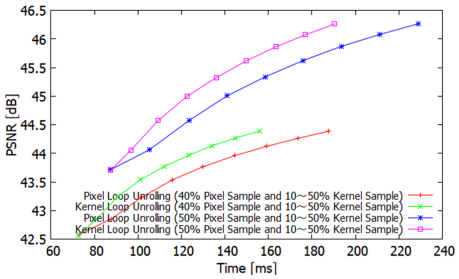
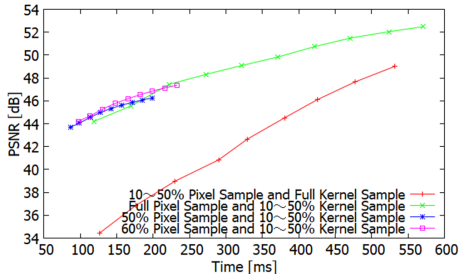
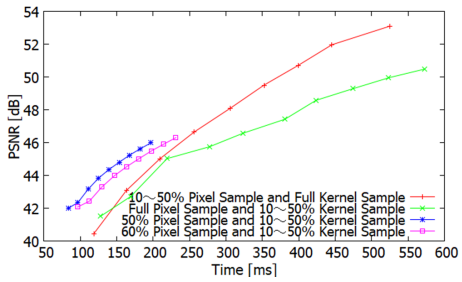


Fig. 6: Comparison between pixel loop unrolling and kernel loop unrolling when subsampling pixel and kernel.



(a) $r = 24,\ \sigma_c = 48$



(b) $r = 24,\ \sigma_c = 120$

Fig. 7: Performance comparison of bilateral filtering with subsampling.

an edge-preserving smoothing filter, we use importance sampling that densely samples near the edge. We experiment on subsampling only the pixel, subsampling only the kernel, and subsampling the pixel and the kernel changing the parameters. We implemented this in C++. The compiler we use is Visual Studio 2015. We use AVX and FMA in SIMD and use Open MP as parallelization. Our computing environment, the CPU is Intel Core i7-6700 3.40GHz, the memory is 32G, and the OS is Windows 10 64 bit. First, we compare the performance of pixel loop unrolling and kernel loop unrolling for subsampling only the pixel, subsampling only the kernel, and subsampling the pixel and kernel. We set the parameters as the kernel radius $r = 24$, $\sigma_s = 8$, and $\sigma_c = 48$. As shown in Figure 4, the processing time for kernel loop unrolling is slightly short. Also, in Figure 5, 6, in pixel loop unrolling, we load the reference pixels by the gather instruction twice. Therefore, the processing time of kernel loop unrolling is considered to be short as a whole. Next, we show the result of verifying which subsampling method should be selected for what kind of parameters. The result of the verification is shown in Figure 7. In the case of $r = 24$, $\sigma_c = 48$ of Figure 7a, accuracy is low when only pixels are subsampled, and accuracy is high when only the kernel is subsampled. When pixels and kernel are subsampled, we set the pixel usage rate to 50% or 60%, and change the kernel usage rate from 10% to 50%. In that case, processing time is short, but when the usage rate of the kernel is the same, accuracy is lower than when only the kernel is subsampled. Therefore, when σ_c is small, it is considered that we should choose to subsample only the kernel.

In the case of $r = 24$, $\sigma_c = 120$ of Figure 7b, accuracy is high when only pixels are subsampled, and accuracy is low when only the kernel is subsampled. When subsampling the pixel and the kernel, when the pixel usage rate is 60%, it has the same accuracy when only the kernel is subsampled, and it is located on the upper left of another method in a graph. Therefore, when $\sigma_c = 120$, it is better to subsample the kernel using about 60% of the pixels.

V. Conclusion

In this paper, we accelerate image filtering by subsampling filtering kernel and images by randomized algorithms. Also, the subsampling is vectorized by vector addressing. Experimental results show that vectorization of kernel loop unrolling is faster than pixel loop unrolling. Also, kernel and image subsampling is effective for acceleration of bilateral filtering which a smoothing parameter of a range kernel is large.

Acknowledgment

This work was supported by KAKENHI JP17H01764, JP18K19813.

References

[1] S. Bae, S. Paris, and F. Durand, *Two-scale tone management for photographic look*, ACM Transactions on Graphics **25** (2006), 637–645.

[2] F. Banterle, M. Corsini, P. Cignoni, and R. Scopigno, *A low-memory, straightforward and fast bilateral filter through subsampling in spatial domain*, Computer Graphics Forum **31** (2012), no. 1, 19–32.

[3] R. C. Gonzalez and R. E. Woods, *Digital image processing*, Prentice Hall, 2008.

[4] T. Kollig and A. Keller, *Efficient multidimensional sampling*, Computer Graphics Forum **21** (2008), no. 3.

[5] L. Lou, P. Nguyen, J. Lawrence, and C. Barnes, *Image perforation:automatically accelerating image pipelines by intelligently skipping samples*, ACM Transactions on Graphics **35** (2016), no. 153.

[6] T. Schlomer, D. Heck, and O. Deussen, *Farthest-point optimized point sets with maximized minimum distance*, Proceedings of ACM SIGGRAPH Symposium on High Performance Graphics, 2011, pp. 135–142.