Halide and OpenMP for generating High-performance recursive filters

Yuta Tsuji and Norishige Fukushima
Nagoya Institute of Technology, Nagoya, Japan

ABSTRACT

Halide is a domain-specific language for image processing. Halide can separate a programming code into an algorithm part and a scheduling part. We can write how to work image processing in the algorithm part, and how to compute it in the scheduling part. The scheduling part has a restriction, which does not change the calculation result. The restriction prevents efficient code generation for some kinds of image processing. In this paper, we propose high-performance recursive filters with Halide and OpenMP. The recursive filter is one of the difficult algorithms for optimizing code with Halide. In our implementation, we divided an input image into multiple tiles, and then each tile is processed with Halide code. Also, each processing for a tile in Halide is parallelized by OpenMP. The processing has an approximated computation in boundary conditions for forcefully cutting the effect from the next tiles; thus, the resulting image has slightly degraded. The closed tile, however, improves the cache efficiency in computation. In the experiment, the processing time of the box image filtering with and without tiling was compared. The box image filtering is the most simple recursive filter. The box image filtering uses an integral image technique to process in constant time, and its calculation includes recursive filters. Code with tiling, which was the proposed method, the tile size is 128×128. The experimental results showed that the proposed method had a better computational time performance than the code without tiling.

Keywords: Halide, OpenMP, recursive filter, approximated computing

1. INTRODUCTION

Halide[1-3] is a domain-specific language for image processing. In Halide, we can separate code into an algorithm part and a scheduling part, i.e., we can write how to work image processing in the algorithm part, and how to compute it in the scheduling part. The modulation makes the optimizer in the Halide compiler easier than the usual programming language. The programmer can provide optimized codes for various devices by changing only scheduling parts.

The scheduling part in Halide has a restriction. The scheduling part does not change the calculation result by changing scheduling for optimizing code. The restriction is usual but strict for some image processing codes. For example, in the recursive filtering case, the restriction prevents efficient code generation with tiling scheduling. Recursive filters include various types of filters, such as box filtering[4,5], Gaussian filtering[6-13], constant-time bilateral filtering[14-18], guided image filtering[19-22], domain transform filtering[23], and recursive bilateral filtering[24].

The work[20] presents an efficient code for guided image filtering for CPU, and also extend Halide to support FPGA. The code is optimized in the restriction of the Halide compiler. The extension for specific recursive filtering in Halide is also proposed[25]. This extension automatically generates an extremely complex algorithm part for efficient recursive filtering. However, we cannot write the scheduling of approximated computing for more efficient code generation in both cases.

In this paper, we propose a writing rule of Halide for high-performance recursive filtering with approximated computing. For the approximated computing in Halide, we additionally use OpenMP to break the rule of Halide. OpenMP is a compiler directive for parallel computing. With our approach, the image is forcefully tiled for efficient cache-hit without scheduling restrictions by approximating computation in boundary conditions. The proposed method can perform tiling scheduling that is not possible with a general description in Halide.

2. RECURSIVE FILTER

In this paper, we treat box filtering as a typical recursive filter. Algorithm 1 shows box image filtering written in Halide. Let input(x, y, c) and output(x, y, c) are input and output images, where x, y, and c are variables representing image rows, columns, and color channels, respectively. r is the kernel radius of filtering, R is RDom class with values of -r to width+r in the x-direction and -r to height+r in the y-direction, i.e., RDom class represents loop variables. The algorithm uses integral images[4,5] to compute box filtering in constant time. The naïve filtering of box filtering is just finite impulse response (FIR) filtering, but the integral image based filtering becomes recursive filtering. The relevant parts are pipeline stage 3 and 4 in Algorithm 1. The filter requires the computed result of previous pixels; thus, this filter is recursive filtering.

In this algorithm, we cannot use image tiling scheduling efficient. In tiling scheduling, we bound image as blocks and then perform processing. However, in the case of recursive filtering, we need to compute the entire image for the tiling part in the worst case. Algorithm 2 shows the case. The scheduling requires results from other tiling parts for one-tile computation; thus, the computation requires massive redundant computation.

Algorithm 1 BoxFilter

```
1: $s = (2*r+1)*(2*r+1)$
2: $\mathrm{inte}(x, y, c) = \mathrm{input}(x, y, c)$
3: $\mathrm{inte}(R.x, R.y, c) = \mathrm{inte}(R.x, R.y, c) + \mathrm{inte}(R.x-1, R.y, c)$
4: $\mathrm{inte}(R.x, R.y, c) = \mathrm{inte}(R.x, R.y, c) + \mathrm{inte}(R.x, R.y-1, c)$
5: $\mathrm{output}(x, y, c) = (\mathrm{inte}(x+r, y+r, c) - \mathrm{inte}(x-r-1, y+r, c)$
           $-\mathrm{inte}(x+r, y-r-1, c) + \mathrm{inte}(x-r-1, y-r-1, c) * s)$
```

Algorithm 2 Pseudo code of computation in the case of apply tiling scheduling to Algorithm1

```
$\mathrm{output.tile}(x, y, xo, yo, xi, yi, tile\_x, tile\_y)$;
$\mathrm{inte.compute\_at}(\mathrm{output}, xo)$;
$s = (2*r+1)*(2*r+1)$
for $c = 0$ to $chanels$ do
  for $yo = 0$ to $height/tile\_y$ do
    for $xo = 0$ to $width/tile\_x$ do
      for $y = 0$ to $height$ do
        for $x = 0$ to $width$ do
          $\mathrm{inte}(x, y, c) = \mathrm{input}(x, y, c)$
        end for
      end for
      for $R.y = -r$ to $height + r$ do
        for $R.x = -r$ to $width + r$ do
          $\mathrm{inte}(R.x, R.y, c) = \mathrm{inte}(R.x, R.y, c) + \mathrm{inte}(R.x-1, R.y, c)$
        end for
      end for
      for $R.y = -r$ to $height + r$ do
        for $R.x = -r$ to $width + r$ do
          $\mathrm{inte}(R.x, R.y, c) = \mathrm{inte}(R.x, R.y, c) + \mathrm{inte}(R.x, R.y-1, c)$
        end for
      end for
      for $yi = 0$ to $tile\_y$ do
        for $xi = 0$ to $tile\_x$ do
          $x = xo * tile\_x + xi$;
          $y = yo * tile\_y + yi$;
          $\mathrm{output}(x, y, c) = (\mathrm{inte}(x+r, y+r, c) - \mathrm{inte}(x-r-1, y+r, c)$
                $-\mathrm{inte}(x+r, y-r-1, c) + \mathrm{inte}(x-r-1, y-r-1, c) * s)$
        end for
      end for
    end for
  end for
end for
```

3. PROPOSED METHOD

Figure 1 shows an example of a general Halide code execution. Usually, the Halide compiler performs JIT compilation and function execution by calling the realize-function. Let src and dest are input and output images, r is the kernel radius, and output is a Halide Function. This description must be in Halide rule, i.e., computing scheduling cannot change the computing result by changing the computational order. This description cannot generate efficient tiling scheduling for recursive filtering.

Figure 2 shows the proposed tiling method. Let subsrc and subdest are tiled input and output images. In our implementation, we divided an input image into multiple tiles at first, and then each tile is processed with Halide code. In this example, we divide an image into 16 tiles. At first, the code forcefully bounds image as tiles, the outer part of the tile image is cutoff. Therefore, the scheduling can reduce redundant computation in this tiling scheduling. Also, each processing written in Halide is parallelized by OpenMP. This tiling method cut off redundant computation of recursive filtering. The writing method of separating tiles before writing Halide code overcome the rule of Halide restriction.

The processing, however, has an approximated computation in boundary conditions; thus, the resulting image has slightly degraded. To control the degradation, we add the tile expansion process. Figure 3 shows the outline of the tile expansion in the one-dimensional case. When the amount of expansion is 0, the accessed pixel, which is outside of the tile, uses a copy of the pixel at the edge of the tile itself. When the amount of expansion is full size, the outside pixels is copied from input image at generating the tile. We can control the amount of the copy. If the size of the tile expansion is small, we can obtain high cache efficiency. If we have the larger tile expansion, cache efficiency becomes low but have fewer approximated computation. Also, by extending the tile to the same radius as the kernel radius, we can obtain the same calculation results as when we do not perform tiling strategy, except the rounding error of the floating-point number due to the loop or computing order changing.

```
output = BoxFilter(src, r);
output.realize(dest);//Execution
```

Figure 1. General execution description in Halide.

```
for (int i = 0; i < 16; i++)
  output[i] = BoxFilter(src[i], r);
#pragma omp parallel for
for (int i = 0; i < 16; i++)
  output[i].realize(dest[i]);//Execution
```

Figure 2. Proposed method for tiling description in Halide with OpenMP.

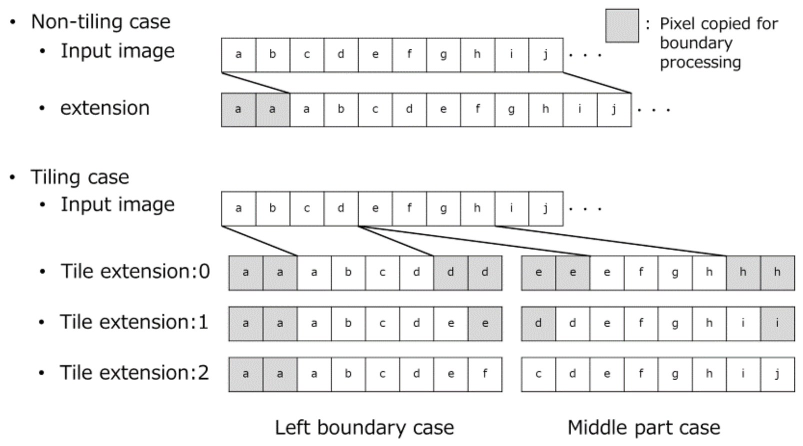

Figure 3. Tile expansion for one-dimensional signals. Assuming radius is 2. In the non-tiling case, the left boundary is expanded from the most left pixel by 2 pixels. For the tiling case, we divide the input image into tile size, and then copy the values. Next, we copy input image for boundary as adequate extension size. Note that, in the case of left boundary, the processing is the same case of the non-tiling case. Note that the expansion is copying, i.e., replicating. We can also utilize wrapping and reflecting on the boundary condition.

4. EXPERIMENTAL RESULTS

In the experiments, we compared the proposed method with the non-tiling scheduling code for box filtering. The input image was a 512×512 color image. The parameter of the filter is r = 10. CPU was Intel Core i9-9900K 3.60 GHz compiled with Visual Studio 2017. For comparing the approximation accuracy in tile expansion, we regarded the result of the non-tiling scheduling code as the ground truth and measured it by peak signal-to-noise ratio (PSNR) between the output of the proposed scheduling and the ground truth. The schedule of the non-tiling code is depicted in Figure 4.

Figure 5 shows that the proposed method has better computational time performance than the non-tiling scheduling code. Figure 6 indicates that the tile expansions in the proposed method are directly proportional to PSNR and computational time. Therefore, the trade-off between approximated accuracy and computational time can be balanced by adjusting the size of the tile expansion. It can also be seen that the amount of tile expansion beyond the kernel radius does not contribute to accuracy. Usually, PSNR of casting from a 32bit-float image to a 8bit integer image is about 58 dB; thus, the approximation reaches enough accuracy for 8 bit images.

```
inte.compute_root().split(y, yo, y, 16).vectorize(x, 4).parallel(yo);
inte.update(0).parallel(R.y);
inte.update(1).vectorize(R.x);
output.compute_root().split(y, yo, y, 16).parallel(yo);
```

Figure 4. The schedule of the non-tiling code.

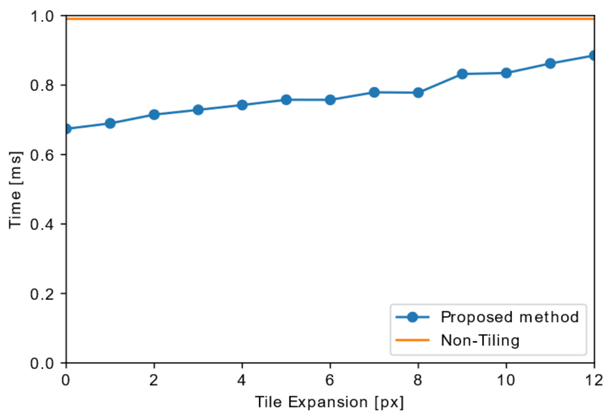

Figure 5. Computational time of the proposed method and the non-tiling scheduling code. The measurement of the proposed method is from 0 to 12 for tile expansion.

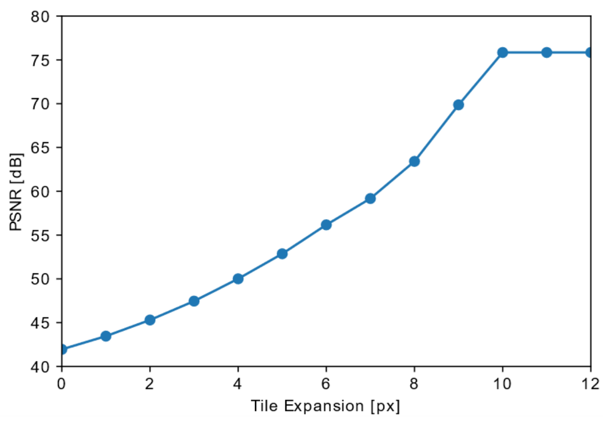

Figure 6. PSNR of the proposed method with respect to the amount of tile expansion from 0 to 12.

5. CONCLUSION

In this paper, we proposed a high-performance scheduling method for recursive filtering of box filtering by using Halide and OpenMP. Beyond the restriction of the Halide compiler, the proposed method accelerates the recursive filtering by scheduling with approximated computing. The experimental results show that the proposed method has a better computational time performance than the non-tiling scheduling code. Also, the trade-off between approximated accuracy and computational time can be balanced by the amount of the tile expansion.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI (JP17H01764, JP18K19813).

REFERENCES

[1] Ragan-Kelley, J., Adams, A., Paris, S., Levoy, M., Amarasinghe, S., and Durand, F., "Decoupling algorithms from schedules for easy optimization of image processing pipelines," ACM Transactions on Graphics, 31, 4, 32 (2012).

[2] Ragan-Kelly, J., Barnes, C., Adams, A., Paris, S., Durand, F., and Amarasinghe, S., "Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," ACM Programming Language Design and Implementation, 48, 6, 519-530 (2013).

[3] Mullapudi, R. T., Adams, A., Sharlet, D., Ragan-Kelley, J., and Fatahalian, K., "Automatically scheduling Halide image processing pipelines," ACM Transactions on Graphics, 35, 4, 83 (2016).

[4] Viola, P. and Jones, M., "Rapid object detection using a boosted cascade of simple features," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 511–518 (2001).

[5] Crow, F. C., "Summed-area tables for texture mapping," ACM SIGGRAPH, 207–212 (1984).

[6] Deriche, R., "Fast algorithms for low-level vision," IEEE Transactions on Pattern Analysis Machine Intelligence, 12, 78-87 (1990).

[7] Van Vliet, L. J., Young, I. T., and Verbeek, P. W., "Recursive Gaussian derivative filters," International Conference on Pattern Recognition (ICPR) (1998).

[8] Sugimoto, K. and Kamata, S., "Fast image filtering by DCT-based kernel decomposition and sequential sum update," IEEE International Conference on Image Processing (ICIP), 125–128 (2012).

[9] Sugimoto, K. and Kamata, S., "Fast Gaussian filter with second-order shift property of DCT-5," IEEE International Conference on Image Processing (ICIP), 514–518 (2013).

[10] Sugimoto, K. and Kamata, S., "Efficient constant-time Gaussian filtering with sliding DCT/DST-5 and dual-domain error minimization," ITE Transactions on Media Technology and Applications, 3, 1, 12–21 (2015).

[11] Sugimoto, K., Kyochi, S., and Kamata, S., "Universal approach for DCT-based constant-time Gaussian filter with moment preservation," IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 1498–1502 (2018).

[12] Yano, T., Sugimoto, K., Kuroki, Y., and Kamata, S., "Acceleration of Gaussian filter with short window length using DCT-1," Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), 129–132 (2018).

[13] Fukushima, N., Maeda, Y., Kawasaki, Y., Nakamura, M., Tsumura, T., Sugimoto, K., and Kamata, S., "Efficient computational scheduling of box and gaussian FIR filtering for CPU microarchitecture," Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), 875–879 (2018).

[14] Porikli, F., "Constant time o(1) bilateral filtering," IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2008).

[15] Yang, Q., Tan, K. H., and Ahuja, N., "Real-time o(1) bilateral filtering," IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2009).

[16] Chaudhury, K. N., Sage, D., and Unser, M., "Fast o(1) bilateral filtering using trigonometric range kernels," IEEE Transactions on Image Processing, 20, 12, 3376–3382 (2011).

[17] Sugimoto, K. and Kamata, S., "Compressive bilateral filtering," IEEE Transactions on Image Processing, 24, 11, 3357–3369 (2015).

[18] Sugimoto, K., Fukushima, N., and Kamata, S., "200 FPS constant-time bilateral filter using SVD and tiling strategy," IEEE International on Image Processing (ICIP) (2019).

[19] He, K., Sun, J., and Tang, X., "Guided image filtering," European Conference on Computer Vision (ECCV) (2010).

[20] Ishikawa, A., Fukushima, N., Maruoka, A., and Iizuka, T., "Halide and GENESIS for generating domain-specific architecture of guided image filtering," IEEE International Symposium on Circuits and Systems (ISCAS) (2019).

[21] Fukushima, N., Sugimoto, K., and Kamata, S., "Guided image filtering with arbitrary window function," IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2018)

[22] Tsubokawa, T., Nakamura, M., Maeda, Y., and Fukushima, N., "Tiling parallelization of guided image filtering," International Workshop on Frontiers of Computer Vision (IW-FCV) (2019).

[23] Gastal, E. S. L. and Oliveira. M. M., "Domain transform for edge-aware image and video processing," ACM Transactions on Graphics, 30, 4 (2011).

[24] Yang, Q., "Recursive bilateral filtering," European Conference on Computer Vision (ECCV) (2012).

[25] Chaurasia, G., Ragan-Kelly, J., Paris, S., Drettakis, G., and Durand, F., "Compiling high performance recursive filters," High-Performance Graphics (HPG), 85-94 (2015)