FPGA implementation of 2D Convolution using OneAPI and OpenCL

Daouda Diakite
Université Paris-Saclay, CNRS, CentraleSupélec,
Laboratoire des Signaux et Systèmes
91190, Gif-sur-Yvette, France
daouda.diakite@l2s.centralesupelec.fr

Nicolas Gac
Université Paris-Saclay, CNRS, CentraleSupélec,
Laboratoire des Signaux et Systèmes
91190, Gif-sur-Yvette, France
nicolas.gac@l2s.centralesupelec.fr

***Abstract*—Thanks to High-Level Synthesis (HLS) tools, FPGAs have become an alternative to GPUs for compute-intensive applications. These tools have been developed to provide flexibility in FPGA design at a higher abstraction level than hardware description languages. Major FPGA manufacturers have proposed many HLS tools depending on the target audience. In this paper, we propose a hardware architecture on FPGA for 2D convolution designed through two software-like development tools based on oneAPI and OpenCL languages. This paper also focuses on comparing the oneAPI and OpenCL HLS tools in terms of performance and productivity with the case study of the 2D convolution operator using an Intel Stratix 10 device.**

***Index Terms*—Convolution, Local memory, OneAPI, OpenCL**

I. Introduction

Convolution is one of the most used operators in signal and image processing applications such as edge detection, image blurring, noise reduction [1]. The operator is very compute-intensive, mainly used in iterative algorithms for inverse problems such as deconvolution [2]. It is essential to have significant computing power and a consequent bandwidth to meet the high computational demand. The acceleration of 2D convolution has been explored on different computing architectures such as CPU, GPU, or FPGA [3], [4]. Many approaches have been developed to accelerate the convolution operator, such as the reduction of the arithmetic complexity notably by using Winograd method [5], simple matrix multiplication using core tensors [6] or element-wise multiplication in the Fourier domain [7]–[9]. The element-wise multiplication in the Fourier domain followed by the inverse Fourier transform delivers better throughput than the convolution in the image domain for large mask sizes [10]. These works also explored different computational accuracy (half, fixed, single, or double precision) to leverage the underlying architecture better. This paper will focus on the pipeline execution model for FPGA. We proposed a convolution accelerator with a memory access strategy to reuse the data and improve the algorithm's arithmetic intensity, allowing us to use FPGA BRAM efficiently.

II. 2D convolution

The 2D convolution represents a multiply-accumulation operation between a mask (convolution kernel) and the pixels of a given image. Let f be a 2D image of size (H, W) and h a mask of size (K, K), the convolution of f by h noted F is given by:

$$F(x,y) = \sum_{j=0}^{K-1}\sum_{i=0}^{K-1} f(x-(i-\frac{K}{2}), y-(j-\frac{K}{2}))h(i,j) \quad (1)$$

To deal with the problem of image edge, we use the zero-padding method, which allows us to maintain the size of the input image. The pixel padding technique is often the acceptable solution to handle image edges, and zero-padding is the simplest to implement among the different types of padding used [11].

III. FPGA-based Convolution

We have described equation 1 in OpenCL and oneAPI for an FPGA implementation by performing the same optimizations to evaluate the efficiency of both tools. We will compare the efficiency of these tools on this operator using several mask sizes. Our implementation uses shift register technique to leverage FPGA local memory. Traditionally, the shift register method is used on FPGA to perform the convolution using Hardware Description Languages (HDL). In that case, the shift is applied to the local buffer for the input data to add new incoming data at the bottom of the buffer. Therefore, the buffer size cannot be too large to avoid expensive shift costs. In our implementation, the shift is only applied to the mask buffer, which is implemented into FPGA registers. A local buffer is

Algorithm 1 Optimized kernel

```
for all yn do
    Prefetched new line into f_local
    Circular Shift the mask buffer
    #pragma unroll 4
    for all xn do
        result ← 0
        #pragma unroll
        for all j do
            #pragma unroll
            for all i do
                Handle image border
                result+ = f_local[xn, yn] * h[i, j]
            end for
        end for
        output_local[xn] = result
    end for
    F ← output_local
end for
```

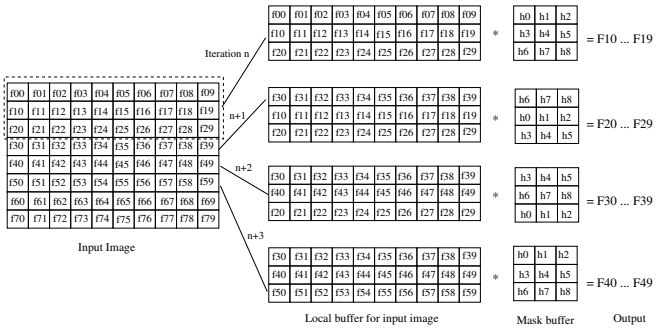


Fig. 1. Memory access strategy with circular shift applied to the mask

implemented using BRAM resources to store the input data. At each iteration of loop yn, a new image line is inserted into the local buffer replacing the buffer line whose coefficients will no longer be used for this iteration as illustrated in Fig.1.

The circular shift intends to switch the rows of the convolution mask so that the element-wise multiplication of the input image and the mask is done with correct correspondence between their lines. At a given iteration n, the circular buffer contains all the input data to compute a line of the output image without access to global memory. However, for the next iteration $n + 1$, the line at the top of the buffer is no longer required to compute the next output line. Therefore this line is replaced by a new image line, and then the bottom line of the mask buffer becomes the top to match coefficients. Then, for iteration $n + 2$, the second line of the local buffer is replaced by the new image line, and at the same time, the mask buffer is shifted. The convolution is performed this way until the output image is completely calculated. Compared to the prefetched data buffer, the mask buffer is smaller and is implemented using registers instead of BRAM resources. Therefore, this circular shift register will be less expensive and energy-efficient for the overall design. The pseudo-code of this version is presented in Algorithm 1.

IV. Results and discussion

We notice better performance for the OpenCL tool in almost all cases for the convolution operator. This performance advantage of OpenCL lies in the difference in depth of the generated pipeline. Indeed, we noticed a significant difference in these two tools' global memory access latencies (201 versus 818 cycles). This memory latency impacts the number of pipeline stages, especially in the blocks where access to global memory is required. This has little impact on performance for an ideal pipeline as long as the pipeline can produce a result at each clock cycle. Regardless of the depth, once filled, the pipeline can perform an update at every clock cycle (or at regular intervals), given that we are working on massive data. On the other hand, all the pipeline stages will be impacted if the computation is stalled due to memory accesses (high stall percentage). This delay will propagate in pipeline depth, which will prevent an update from being produced at each clock cycle. Synthesis tool with oneAPI is less advantageous in this regard, with a $4\times$ higher global memory latency than using OpenCL, and this latency will affect the generated pipeline. In any case, a reduction of global memory access is necessary to guarantee better performance, and for this reason, local memory usage must be privileged. Our optimized versions reduce the memory access, and the stall percentage is alleviated. The results of the optimized versions are presented in table I for the oneAPI and OpenCL kernels. We can observe similar performance with a slight advantage for OpenCL due to the performance loss for oneAPI kernels.

Mask size	OpenCL Time (ms)	oneAPI Time (ms)	OneAPI lost of performance (%)
3×3	0.28	0.31	10.7
5×5	0.29	0.36	24.1
7×7	0.32	0.44	37.5
9×9	0.35	0.49	40
15×15	0.52	×	×

TABLE I
Performance comparison between OpenCL and oneAPI

V. Conclusion

This paper deals with the acceleration of the 2D convolution operator on FPGA with high level languages. Several implementations have been implemented to evaluate the OpenCL and oneAPI synthesis tools from Intel. These tools allow to have an implementation on FPGA with a reduced development time, but a powerful design requires a deep analysis of the application in order to apply consequent FPGA-specific optimizations. We find that OpenCL offers the best trade-off between development time and performance. However, oneAPI is gaining more and more maturity and could become an alternative to OpenCL.

References

[1] W. Burger and M. J. Burge, *Digital image processing: an algorithmic introduction using Java*. Springer, 2016.

[2] J. Idier, *Bayesian approach to inverse problems*. John Wiley & Sons, 2013.

[3] J. Fowers, G. Brown, J. Wernsing, and G. Stitt, "A performance and energy comparison of convolution on gpus, fpgas, and multicore processors," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 9, no. 4, pp. 1–21, 2013.

[4] Z. Jin and H. Finkel, "Exploration of OpenCL 2d convolution kernels on intel FPGA, CPU, and GPU platforms," in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 4460–4465.

[5] S. Winograd, *Arithmetic complexity of computations*. Siam, 1980.

[6] M. Seznec, N. Gac, F. Orieux, and A. S. Naik, "A new convolutions algorithm to leverage tensor cores," GPU Technology Conference (GTC), May 2020, poster.

[7] E. O. Brigham, *The fast Fourier transform and its applications*. Prentice-Hall, Inc., 1988.

[8] D. Kolba and T. Parks, "A prime factor fft algorithm using high-speed convolution," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 25, no. 4, pp. 281–294, 1977.

[9] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, "Fast convolutional nets with fbfft: A gpu performance evaluation," *arXiv preprint arXiv:1412.7580*, 2014.

[10] M. Seznec, N. Gac, A. Ferrari, and F. Orieux, "A study on convolution using half-precision floating-point numbers on gpu for radio astronomy deconvolution," in *2018 IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2018, pp. 170–175.

[11] H. Ström, "A parallel fpga implementation of image convolution," 2016.