IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 12, 2020, accepted June 22, 2020, date of publication July 3, 2020, date of current version July 16, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3006773

An FPGA-Based Hardware Accelerator for Real-Time Block-Matching and 3D Filtering

DONG WANG[1], (Member, IEEE), JIA XU[2], AND KE XU[2]
[1]Institute of Information Science, Beijing Jiaotong University, Beijing 100044, China
[2]Beijing Key Laboratory of Advanced Information Science and Network Technology, Beijing Jiaotong University, Beijing 100044, China

Corresponding author: Dong Wang (wangdong@bjtu.edu.cn)

This work was supported in part by the Fundamental Research Funds for the Central Universities under Grant 2020JBM020, and in part by the Beijing Natural Science Foundation under Grant 4202063.

ABSTRACT Block-matching and 3D filtering (BM3D) denoising algorithm has been employed in many application fields because of its superior image processing quality. Due to the huge computational workload, real-time implementation of this algorithm is very challenging. Recently, studies on accelerating the BM3D algorithm on GPU have presented impressive speed up over CPU-based implementations. However, GPU devices are generally inefficient in energy dissipation and, thus, are not suitable for embedded application scenarios. In this paper, we propose a dedicated hardware accelerator design to efficiently boost the BM3D algorithm with reduced power consumption on FPGA device. The proposed design is based on a deeply pipelined OpenCL kernel architecture that can efficiently speed up the compute-intensive procedures of the denoising algorithm by exploiting the intrinsic parallelism and maximizing data reuse. The final design was implemented on Intel's Arria-10 GX1150 FPGA, and achieved an average 1.2× performance boost and an outstanding 8.3× reduction in energy dissipation when compared to a state-of-the-art GPU-based software design.

INDEX TERMS FPGA, BM3D, systolic array, image denoising.

I. INTRODUCTION

Image denoising plays an important role in image and video processing and has become one of the most fundamental technologies in many fields, such as digital camera [1], medical image processing [2] and computer vision [3]. Traditional noise reduction algorithms can be divided into two broad categories, i.e., spatial-domain and transform-domain denoising. Spatial-domain schemes directly perform denoising algorithms on the raw pixels of the image, while transform domain-based methods operate on a sparse representation of the image in the transform domain and manipulate the transformed coefficients to reduce noise. Among the many approaches presented in the literature, the block-matching and 3D filtering (BM3D) algorithm [4] effectively combines non-local filtering and transform domain filtering and exhibits outstanding denoising performance, especially for additive white Gaussian noise. It is still considered as one of the state-of-the-art denoising method [9], [10].

The associate editor coordinating the review of this manuscript and approving it for publication was Qiangqiang Yuan.

However, BM3D is intrinsically compute-intensive, which makes real-time implementation of the algorithm challenging. Therefore, studies on real-time acceleration of the BM3D algorithm have received many attention in recent years. In [7], the authors presented a multi-core CPU-based implementation, which utilizes OpenMP for parallelization. The block-matching procedure was accelerated by an efficient data reuse method applied to the whole image, while the 3D transformation was accelerated by utilizing a fast software FFTW library. Later on, the work of [8] proposed the first open-source GPU-based implementation of the BM3D algorithm by using both the OpenCL and CUDA frameworks. The final design achieved 7.5× speed up compared to the CPU-based design of [7]. One of the drawbacks of this work was that it only supported grayscale images. Very recently, the study of [9] presented a highly efficient GPU-based software accelerator implemented in CUDA. The authors proposed a more fine-grained partition of the algorithm compared to [8] and optimized data caching and sharing schemes for block-matching, which removed many redundant computations and thus significantly improved the image processing speed. In [10], the authors targeted an GPU implementation

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

for VBM3D and focused on reducing the use of external memory bandwidth, which is realized by regrouping all filtering operations, including fetching the patch data, the data transpositions, the 1D filtering and the thresholding, into one kernel without requiring intermediate bufffer for patches caching.

Unfortunately, GPU devices are generally energy inefficient. For instance, a modern GPU device normally consumes more than 250W power [11], which is infeasible for embedded application scenarios, such as robotics, autonomous vehicles and surveillance systems. On the other hand, field-programmable gate array devices (FPGAs), which provide massive processing elements, reconfigurable interconnections and lower power dissipation, are naturally suitable to implement compute-intensive image processing algorithms [1], [12]–[15]. For instance, our previous study of [17] has presented an FPGA-based BM3D accelerator design which achieved 12× speed-up over the OpenCL-based GPU implementation of [8]. However, the performance was still not comparable with the CUDA-based GPU accelerator of [9] due to inefficient utilization of the on-chip logic resource and external memory bandwidth.

In this paper, we propose an performance improved FPGA accelerator design for real-time processing of the block-matching and 3D filtering algorithm based on our previous study of [17]. The detailed contribution of this study includes: (1) we present a quantitative analysis of the complexity of each functions of the BM3D algorithm and propose a accelerator architecture based on deeply-pipelined OpenCL kernels to implement the partitioned sub-algorithms; (2) A dedicated systolic-like array architecture for parallel block-matching is developed to efficiently exploit fine-grained data-level parallelism of the algorithm through pipelining, and at the same time, save large amount of hardware resources by avoiding using very wide data-buses to support high throughput computation; (3) A parallel line-buffer-based on-chip data caching scheme is also introduced such that data reuse is maximized and the demand on external memory bandwidth is greatly reduced; (4) We have implemented the proposed design on Intel's Arria-10 GX1150 FPGA device, and experiment results showed that our design gained more than 20% performance improvement and in the meantime achieved a significant 8.3× advantage in power consumption over state-of-the-art GPU-based design.

The rest of this paper is organized as follows. Section II and III briefly review the OpenCL framework and the BM3D algorithm, respectively. Section III also highlights the hardware design challenges that this study has faced in designing the FPGA accelerator. Section IV describes the detailed design of the proposed FPGA accelerator. Section V presents the hardware implementation results and compares our results with two GPU-based designs. Section VI concludes this paper.

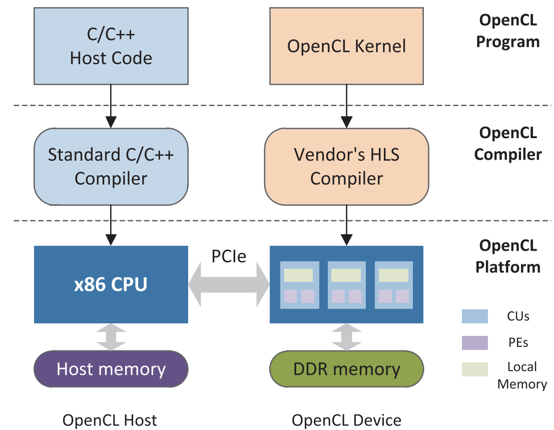


FIGURE 1. The OpenCL framework for FPGA-based accelerator development.

II. OpenCL FRAMEWORK

In this section, we briefly review the OpenCL programming framework with special focus on the methodology and basic architecture-level abstractions for FPGA-based hardware accelerator design. Our target platform is Intel's Arria-10 FPGA Development Kit and the design goal is to accelerate the most time-consuming parts of the BM3D algorithm in dedicated hardware circuits on FPGA to achieve improved real-time image processing performance.

A. DESIGN FLOW

There is a growing trend among the research community to utilize High Level Synthesis (HLS) tools to design and implement customized circuits on FPGAs [18], [19]. Compared with traditional methodology, the HLS tools provide faster hardware development cycle by automatically synthesizing an algorithm in high-level languages (e.g. C/C++) to RTL codes, which is then compiled into FPGA circuits. Fig. 1 summarizes the OpenCL-based FPGA accelerator development flow adopted by this work. Hardware circuits, which accelerate compute-intensive algorithms, are modeled in a high-level abstraction form as OpenCL kernel functions and executed in parallel on the FPGA fabric. A C/C++ code executing on the CPU side provides vendor specific application programming interface (API) to communicate with and control the implemented kernels. This work uses the Intel OpenCL SDK toolset [22] for compiling and implementing the OpenCL codes on FPGAs. Profiling function of the SDK is also used to analyze the performance and resource utilization of the final design. We have defined a set of OpenCL kernels, each of which implements one of the core functions of the BM3D algorithm, e.g. block-matching, 3D transform/inverse transform, aggregation, etc. The host program initiates and launches the hardware kernels in a pipelined manner implementing the whole image filtering process.

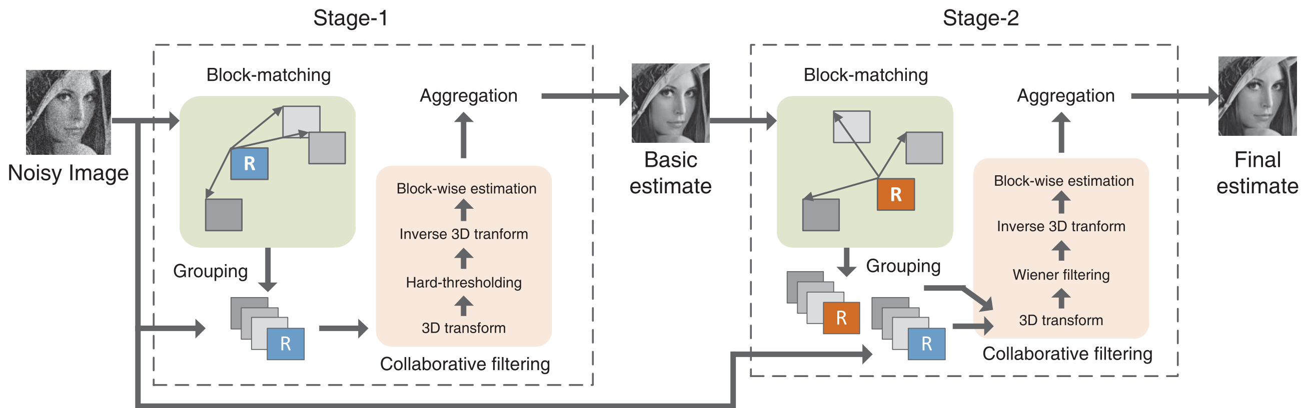


FIGURE 2. The basic processing flowchart of the BM3D algorithm. The block marked with "R" represents the reference block.

B. ACCELERATOR ARCHITECTURE ABSTRACTIONS

The OpenCL framework defines a high-level representation of a parallel hardware architecture on which the OpenCL kernel programs are accelerated. As shown in Fig. 1, the OpenCL device is divided into multiple compute units (CUs), each of which consists of one or more processing elements (PEs). For FPGA-based accelerator design, the OpenCL kernels are directly implemented as dedicated hardware units, i.e. CUs, which can be executed concurrently in parallel. The CUs normally utilize the massive on-chip DSP blocks as the main PEs to speed up the target application.

C. PROGRAMMING MODEL

There are two basic executing models for OpenCL kernel programs. The first one is called N-Dimensioned Range (NDrange) kernel. After being launched by the host, an instance of the NDRange kernel executes one point of the problem index space that is created by the OpenCL runtime system. Each instance of the executing kernel is referred to as a work-item or thread, and multiple thread can be launched concurrently to exploit coarse-grain task-level parallelism. Scheduling of the parallel tasks are normally conducted by the OpenCL runtime, however, designer has to explicitly map the computation problem onto this N-dimensional space. The NDrange programming model is generally useful for GPU-based software accelerator design. The other execution model is single work-item kernel [22], in which multiple threads are executed on replicated pipeline circuits on the device. In each clock cycle, a new thread can be launched on the single work-item kernel to achieve a very high throughput. In addition, fine-grained data-level parallelism can also be utilized through customizing the structure of the underlying pipeline circuit. Compared to NDrange kernels, single work-item programming model is more flexible in controlling the structure of the hardware circuit, and thus, is more suitable in designing accelerators on FPGA-based platform. The key design challenge we have faced in this work is to develop an appropriate hardware architecture that can efficiently exploit the intrinsic parallelism of the BM3D algorithm, while, at the same time, minimizing the on-chip hardware resources.

D. MEMORY MODEL

The OpenCL memory model defines three abstract level of memory regions in which the data can be stored and accessed by the host program and hardware kernels. On the host side, a host memory region is reserved and visible only to the host. All the program, buffer and image objects are created, initiated and stored in the host memory. On the device side, there are two distinct regions, including global memory and local/private memory. The global memory region is located in the external DDR memory on the FPGA board and permits read/write access to all kernels. Local memory region is used to allocate variables that are shared by all the threads running on a OpenCL kernel, while private memory is private to a single thread. In FPGA-based accelerator design, both the local and private memories are located on the device side. Local memory is generally implemented as on-chip embedded memory blocks, while private memory is implemented as registers. It is worth noting that, unlike GPU accelerators, the bandwidth of the global memory on FPGA board is often limited, therefore, optimizing the global memory bandwidth becomes another key challenge in improving the performance of OpenCL-based FPGA accelerators.

III. REVIEW AND ANALYSIS OF BM3D

BM3D is a novel image denosing strategy based on enhanced sparse representation in the transform domain. The enhancement of the sparsity, which is utilized as the major metric to distinguish between image and noise, is achieved by grouping similar 2D image fragments into 3D data arrays. Moreover, grouping is realized by block-matching, i.e., gathering similar patches of the image into groups by measuring the patch distances. After transforming the 3D data array into the frequency domain, collaborative filtering is accomplished by shrinkage of the transformed 3D data array. As illustrated in Fig. 2, the general procedure of BM3D is implemented in two phases, namely the basic estimation phase (stage-1)

and the Wiener filtering phase (stage-2). In order to maximize resource sharing in the hardware accelerator design, we have partitioned the whole denosing flow into six sub-procedures, including block-matching (grouping), 3D transform, inverse 3D transform, block-wise estimation (filtering) and aggregation, each of which is implemented as an OpenCL kernel function and is reused in successive filtering stages.

To guide the hardware design, we have measured the execution time of a software-based implementation [6] of the BM3D algorithm on CPU. The detailed processing time can be used as a quantitative metric to evaluate the computational complexity of each procedure. As shown by Fig. 3, the block-matching and collaborative filtering steps consists up to 98% of the total execution time, which means that the FPGA accelerator must allocate as much hardware resource as possible for these two procedures to support parallel acceleration of the computation. In what follows, we will briefly review all the computational steps of these procedures and identify the intrinsic parallelism that can be utilized to boost the computation. Data transmit and reuse patterns will also be analyzed with a specific emphasis on global memory bandwidth optimization.

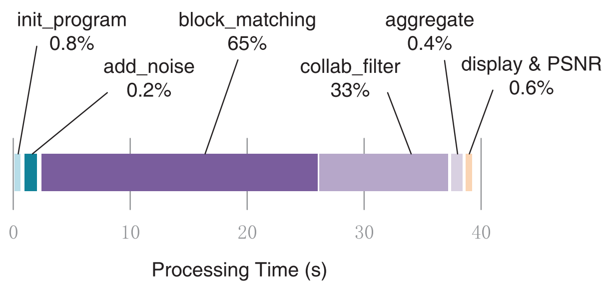


FIGURE 3. Breakdown of the execution time of a software implementation of the BM3D algorithm on CPU [6].

A. GROUPING BY BLOCK-MATCHING

Grouping is a repetitive procedure that is performed on sub-blocks (patches) of the input image. As shown in Fig. 4, for each reference block R, it finds the patches that have high similarity with the currently processed one within a predefined searching window and stacks them together in a 3D array (group). The center of the search window is at the top-left corner of the reference patch, and the window and the block sizes are defined as W_s and K, respectively. The reference patch slides within the whole image with a fixed step of p_{step}, and in [4], p_{step} is set as 3. For a more elaborate filtering process, p_{step} can be set as small as 1.

Similarity is generally calculated as the inverse of the block-distance which means that patches with smaller distances are matched as similar, whereas the ones with larger such distance are left out. As did in most previous studies [5], [9], [10], we adopt the normalized Euclidean ($\mathcal{L}_2$) metric to quantitatively measure the distance. Assuming R denotes the reference patch and P is a block in its neighborhood, then the distance between R and P can be calculated by

$$dist(R, P) = \frac{\mathcal{L}_2^2(R, P)}{K^2} = \frac{\sum_x (R_x - P_x)^2}{K^2} \tag{1}$$

Then, a hard threshold τ_{match} is applied such that the patches whose inverse distances are larger than this threshold are grouped and stored in a 3D array $G_{3D}(R)$.

Assuming that the width and height of the input image are represented by W and H, there is a total number of $(W - K)/p_{step} \times (H - K)/p_{step}$ reference blocks and each block requires an order of $W_s^2 \cdot K^2 \cdot N_{step}$ multiply-accumulate (MAC) operations, where N_{step} denotes the number of the selected similar patches. Fortunately, searching and distance calculation of different reference blocks, such as the blocks R_1 and R_2 shown in Fig. 4, can be conducted in parallel since there is no data dependence among them. Therefore, the proposed FPGA accelerator will exploit fine-grained data-level parallelism of the algorithm to boost the computation.

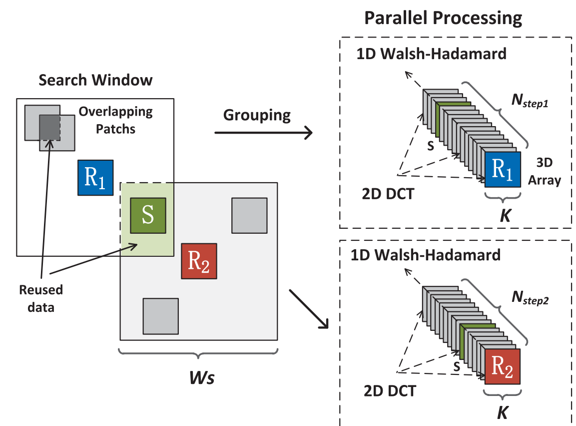


FIGURE 4. Grouping and 3D transformation on two reference patches.

Beside the intrinsic data-level parallelism, the block-matching algorithm also has a high data reuse rate. As shown in Fig. 4, both the searching windows of adjacent reference blocks and the compared patches corresponds to one reference block share a large overlapping area. Therefore, in hardware design, we will propose an efficient on-chip data buffering approach that can maximize reuse of the image data to relieve the pressure on global memory bandwidth. Moreover, we also propose to cache the 3D patches in dedicated on-chip buffers until the collaborative filtering is finished. However, due to the limited memory resource of the FPGA device, the number of the stored patches should be restricted to a fixed constant N_{step}. This means that a hardware distance sorting unit, which can find the first N_{step} blocks with the lowest distance, is also needed after distance calculation. Furthermore, when the number of similar patches is smaller than N_{step}, we propose to fill up the empty entries of the 3D array with the reference block (distance is zero). In this way, the 3D transform operation can have a fixed input length, which greatly reduces the complexity of the

controlling circuit. We have verified that this algorithm optimization has no obvious impact on denoising quality.

B. COLLABORATIVE DENOISE FILTERING

Collaborative filtering consists of three sub-steps: transformation of the 3D data array, hard-thresholding or Wiener-based filtering and inverse 3D transformation. The key idea behind this scheme is that images have much sparse representation in the transform domain than noise because of the intrafragment correlation which appears between the pixels of each patch and, thus, noise can be easily attenuated after frequency thresholding. Compared to 2D transform based denoising, the 3D transform can further take advantage of the interfragment correlation which appears between the corresponding pixels of different patches within the same group and produce a even more sparse representation of the true image. As illustrated in Fig. 2, the first stage of collaborative filtering applies hard-thresholding denoise filter whereas the second stage adopts a Wiener filter.

1) HARD-THRESHOLD FILTERING

By denoting the threshold as λ^{3D}, the hard-threshold filtering function $hd(x)$ can be expressed as

$$hd(x) = \begin{cases} 0, & |x| < \sigma \cdot \lambda^{3D} \cdot \sqrt{N_{step1}} \\ x, & other \end{cases} \tag{2}$$

where σ represents the estimated standard deviation of the noise signal. Normally, a stronger noise is detected, a larger value of σ is set during denose filtering. The first stage denoise filtering procedure can then be summarized by the following formula:

$$\mathcal{P}^{1st}(R) = \mathcal{T}_{3D}^{-1}(hd\ (\mathcal{T}_{3D}(G_{3D}(R)))) \tag{3}$$

where $\mathcal{T}_{3D}$ and $\mathcal{T}_{3D}^{-1}$ present the forward and inverse 3D transforms, respectively. $G_{3D}(R)$ denotes the 3D data array obtained from block matching, and $\mathcal{P}^{1st}(R)$ is the 3D array generated by the first stage collaborative filtering procedure.

2) WIENER FILTERING

In the second stage, the non-linear hard-threshold filtering function $hd(x)$ is replaced by a Wiener filter-based spectrum shrinkage operation as follow

$$\mathcal{P}^{2nd}(R) = \mathcal{T}_{3D}^{-1}(wr(\mathcal{T}_{3D}(G_{3D}(R)))) \tag{4}$$

and the coefficients of the Wiener filter are computed as

$$wr(x) = \frac{\left|\mathcal{T}_{3D}(G_{3D}(R^{basic}))(x)\right|^2}{\left|\mathcal{T}_{3D}(G_{3D}(R^{basic}))(x)\right|^2 + \sigma^2} \tag{5}$$

where R^{basic} is the reference patch on the basic image obtained through the first stage. From above definitions, we can see that the coefficients array has the same size with the reference block and needs to be computed on-the-fly for each reference patch in the basic image. However, it will only be used one time for the current 3D group, which consumes a very small amount of global memory bandwidth when compared to the data access pattern of block-matching. Therefore, as shown by Fig. 5, we choose to store the obtained coefficients in global memory and reserve as much on-chip memory resource as possible to maximize the throughput of the block-matching computation in the proposed FPGA accelerator.

C. AGGREGATION

The aggregation operation combines the filtered 3D arrays to generate an estimation of denoised image. According to [4], patches from different positions may have overlapped pixels with each other, and the same patch may also be included in multiple 3D arrays. As the example shown in Fig. 4, the block S is assigned to both the R_1 and R_2 groups. Considering that homogeneous patches and patches containing edges and corners should be treated differently to avoid undesired distortions, the BM3D algorithm applies a weighted average at these overlapped pixels position as follows:

$$P_{est}(x) = \frac{\sum_R w_{agg} \sum_{P \in \mathcal{P}(R)} M(P,x) \cdot u_{P,R}(x)}{\sum_R w_{agg} \sum_{P \in \mathcal{P}(R)} M(P,x)} \tag{6}$$

where $u_{P,R}(x)$ denotes the pixel value at position x from the patch P produced by the denoised 3D data array $\mathcal{P}(R)$ through inverse transformation. Function $M(P, x)$ is a mask signal which indicates whether the pixel x belongs to the patch. w_{agg} represents the weights of the aggregation operation, which is calculated differently in the two filtering stages as follows:

1) BASIC ESTIMATION PHASE

In the first step, the aggregation weight is calculated as the inverse value of the number of retained elements (pixels) of the 3D group $\mathcal{P}(R)$. Assuming the number of non-zero elements of $\mathcal{P}(R)$ after the hard-threshold filtering as N_R^{1st}, then the weight can be calculated by the following equation:

$$w_{agg}^{1st} = \begin{cases} \dfrac{1}{\sigma^2 N_R^{1st}} & N_R^{1st} > 0 \\ 1 & N_R^{1st} = 0 \end{cases} \tag{7}$$

Considering a patch that contains rich detail information such as edges and texture, after hard-threshold filtering, there are generally more retained elements, so the weight is smaller. In contrast, a homogeneous patch group is more sparse and thus has larger weight. Therefore, the first filtering step favors pixel estimates belonging to the more homogeneous patches.

2) FINAL ESTIMATION PHASE

In the second step, the aggregation produces the final estimate of the denoised image. The weight is calculated as the inverse squared $\mathcal{L}_2$-norm of the Wiener filter coefficients as follows:

$$w_{agg}^{2nd} = \frac{1}{\sigma^2 \|wr\|_2^2} \tag{8}$$

From (7) and (8) we can see that the aggregation process requires complicated floating-point arithmetic operations,

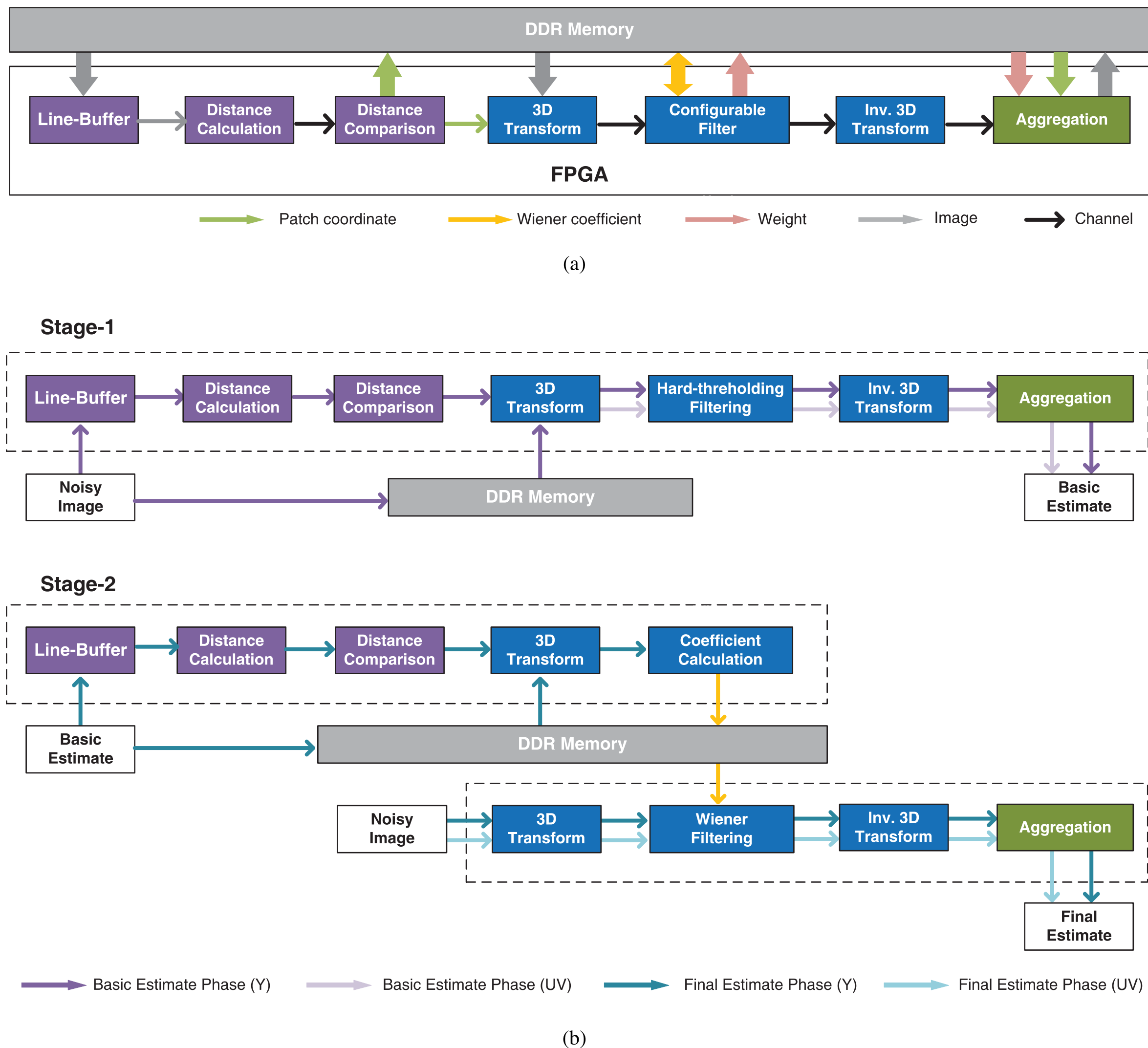


FIGURE 5. (a) Top-level architecture of the proposed FPGA accelerator for BM3D image denoising. (b) Data flow configuration schemes for the first and second filtering stages.

such as square and reciprocal (division), which will cost a large number of the hardware resource on FPGA. To address this design issue, we have proposed several arithmetic level optimizations to reduce the logic and DSP resource consumption. The detailed hardware design is discussed in Section-IV-C.

IV. HARDWARE ARCHITECTURE

Fig. 5-(a) shows the top-level architecture of the proposed FPGA accelerator, which includes a Distance Calculation unit, a Distance Comparison unit, a MemRead unit, a pair of 3D Transform/Inverse Transform units, a Configurable Filtering unit and an Aggregation unit. All these hardware units are implemented as OpenCL kernel functions with configurable hardware parameters (i.e., user defined macros) to control the level of parallelism of the implemented algorithm and balance between the computational performance and hardware cost. The proposed accelerator architecture concatenates the kernels in a way that forms a deep circuit pipeline by using Intel's OpenCL extension Channel. Intermediate computation results can thus be directly passed from one kernel to another resulting in very low frequency of data transmission between FPGA accelerator and global memory.

The Line-Buffer kernel fetches image data from the global memory (i.e., the on-board DDR memory) and caches frequently reused data in on-chip memory to further reduce the pressure on global memory bandwidth. It also converts the serial input data stream into parallel outputs to support high throughput computation in the following kernels. The block-matching procedure is implemented as two kernels, one of which is in charge of searching of the patches and calculating the distances, while the other performs quick sorting of the distance to obtain the 3D patch array with the smallest distance. Collaborative filtering is carried out by the pair of 3D Transform and Configurable Filter kernels. As illustrated by Fig. 5-(b), the Line-Buffer, Distance

Calculation/Comparison, 3D Transform and Configurable Filtering kernels are reused between the first and second stage of the denoising algorithm. For colored image, the block-matching procedure is only performed on Y channel, and the U/V channel reuses the same coordinates with Y channel.

Detailed hardware designs and corresponding optimization schemes are as follow:

A. PARALLEL BLOCK-MATCHING

As discussed in previous section, block-matching is the most compute-intensive procedure of the BM3D algorithm, however, parallel processing of multiple reference patches can be utilized to boost the processing speed since there is no data dependence among different 3D groups. As illustrated in Fig. 6, assuming that we need to process a total number of $M \times N$ reference blocks concurrently in parallel, then the corresponding searching space covers an area of $[W_s + p_{step} \times (M-1)] \times [W_s + p_{step} \times (N-1)]$ image pixels. Since the image data will be repeated used because of the data access pattern of the sliding-window based searching scheme, we propose to cache the entire area (referred to as the prefetch window) of the input image in the on-chip data buffer. In order to efficiently support the $M \times N$ parallel block-matching computations, two optimized hardware architectures are proposed in this design: i) a parallel line-buffer-based on-chip data cache that can simultaneously provide high throughput data stream to support parallel computation and maximize data reuse to improve the global memory bandwidth utilization. ii) a systolic-like 2D PE array architecture is developed to exploit the fine-gained data-level parallelism of the block-matching algorithm by efficiently pipelining the distance calculation and comparison computations on the PE array. Due to the high data reuse rate among the neighborhood PEs, very wide data busses are also avoided resulting in reduced logic resource consumption over the design of [17].

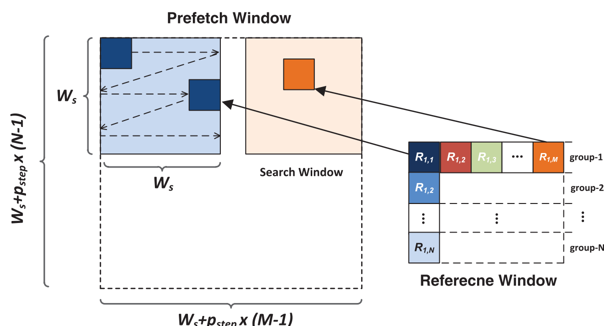


FIGURE 6. Sliding window based patches searching scheme. Searching of the similar patche within each window is carried out in a "zig-zag" sequence.

1) LINE-BUFFER KERNEL

Fig. 7 shows the internal structure of the proposed Line-Buffer kernel. It consists of M parallel line-buffer units, each of which is formed by eight concatenated 2-port RAMs

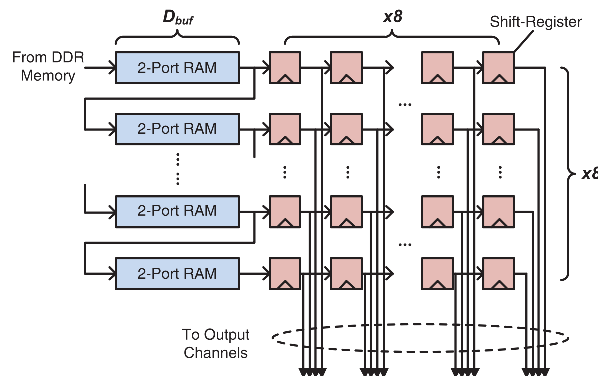


FIGURE 7. Hardware architecture of the proposed parallel line-buffer.

and a 2D array of 8×8 shift-registers. During block-matching, the noisy image pixels of the search window are pushed into the 2-port RAMs in a line-by-line manner starting from the upper-left corner. After the eight line-buffers are filled up with data, the shift-register array reads out eight lines of image pixels from the RAMs and moves the data stream through pipe-lined registers. In every clock cycle, one patch, i.e., 64 pixels, can then be read out from one search window and processed by following kernels. As the line-buffers are updated by each line of the search window, the patches are read from the register array in zig-zag sequence as shown by Fig. 6. The depth of the line-buffer D_{buf} should be larger than the width of the prefetch window.

The proposed line-buffer-based search window prefetching approach significantly reduces the global memory bandwidth by avoiding repetitive memory access of the same image pixel in the overlapped region of the patches. For instance, assuming the search window is configured as 33×33 and the size of the reference block is 8×8, non-optimized design will need to fetch 43K pixels of image data for one search window from the global memory when no data prefetching scheme is adopted, whereas our approach only requires to read around 1K pixels, which satisfactorily reduced the global memory bandwidth by more than $40\times$. This optimization is critical for efficient FPGA-based accelerator designs, since the physical width of the data bus of the FPGA on-chip memory controller is usually shorter than GPUs.

2) DISTANCE CALCULATION KERNEL

In this work, we propose a systolic-like 2D array structure to efficiently support parallel distance calculation for multiple reference blocks. As illustrated by Fig. 8, the systolic array consists of $N \times M$ interconnected PEs. During computation, the image patches of the reference window are divided into N groups, and in each cycle, the image pixels of M reference blocks of the same group are pushed into the PE array simultaneously through the data bus on the top side, while the data stream of the corresponding similar patches are propagated through the data bus on the left side. Both data streams flow through the 2D systolic array in a pipelined way and a total

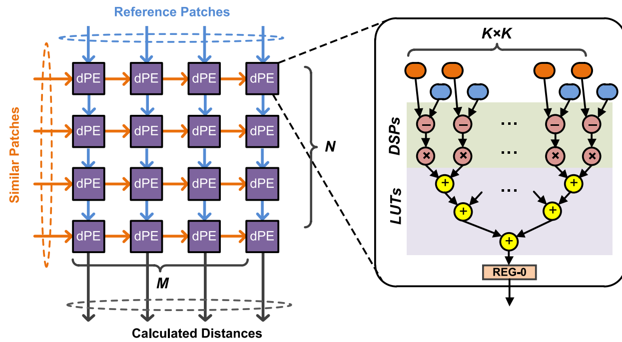


FIGURE 8. Architecture of the Distance Calculation kernel.

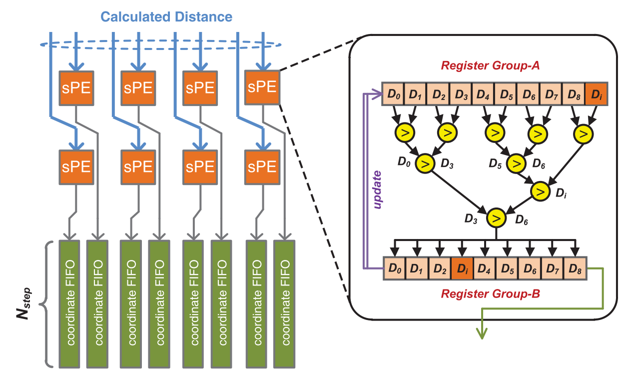


FIGURE 9. Architecture of the Distance Comparison kernel.

number of $N \times M$ distance calculation tasks in the reference window as defined by Fig. 6 can be executed in parallel. The proposed systolic-like architecture has an advantage of avoiding using extremely wide data bus for transmitting image pixel data between the on-chip line-buffer-based data cache and the computational unit. In previous works [17], to support $N \times M$ parallel distance computaion tasks, both of the two input data buses of the PE array were designed as wide as $N \times M \times 64 \times 8$ bits. Such wide data buses can cause increased logic utilization, severe routing problems and degradation in F_{max} of the final implemented design. In the proposed architecture, the input data stream are divided into multiple groups and pushed into the 2D PE array in a wave-by-wave manner, which successfully saved the reference and similar patch data buses by N and M times, respectively.

Each of the distance calculation PE (dPE) consists of 64 multipliers plus a adder tree to implement one complete Euclidean distance calculation defined by Equation (1) when $K = 8$. Division by K^2 is realized by bit-shifting. To improve the computational throughput of the Distance Calculation kernel, the adder tree is divided into eight pipeline stages, and in each clock cycle, the input similar patches are updated with new ones, while the reference blocks remain the same until all the patches in the search window have been compared. Since each DSP block of the target FPGA device can support two $16b \times 16b$ fixed-point multiplication, one dPE of the proposed 2D array will consume 32 DSP blocks.

3) DISTANCE COMPARISON KERNEL

Sorting of the grouped patch distances is another time consuming operation. In OpenCL kernel code, the distance sorting computation is implemented in a loop structure which is hard to be fully unrolled due to the intrinsic data dependency. There are several studies, such as [20], [21], which has developed dedicated hardware sorting circuits on FPGA, however, the reported implementations are either too costly to be applied in our design or only suitable for large data set.

To reduce the hardware implementation cost, we relax the constraint on the sorting procedure of the 3D array by allowing that the N_{step} patches, which have the shortest distance to reference block, can be stored in random order. Then a cost-efficient distance comparison circuit is developed as shown by Fig. 9. The sorting PE (sPE) consists of a parallel comparator tree and a pair of register arrays. In each clock cycle, a newly received distance D_i is inserted into the last position of the input register array, and then the comparator tree finds the maximum distance and returns the index of the corresponding register. If the new distance D_i is not the maximum one, it will be switched with the maximum distance found, and a new patch group is generated and stored in the output register array. In order to maintain a short critical path delay, we have limited the depth of the grouped 3D array to 16. During comparison both the distance data and patch indexes are stored in the pair of register arrays. Note that we have verified that the proposed hardware-oriented algorithm-level optimization schemes have no obvious impact on the quality of the denoised image.

B. 3D TRANSFORM/INVERSE TRANSFORM KERNELS

The collaborative filtering procedure is implemented by three pipelined kernels, including the 3D Transform kernel, Configurable Filter kernel and Inverse 3D Transform kernel. Due to limited on-chip memory resources, the pixel values of the similar patches are not cached on FPGA. Instead of storing $64 \times 3 \times 16$-bit data for each colored image patche, the proposed accelerator only stores the upper-left corner coordinates in two 16-bit words. After receiving the patch coordinates from the Distance Comparison kernel, the 3D Transform kernel loads the corresponding data from global memory and feeds the pixel stream to the internal 3D transformation circuit.

As shown in Fig. 4, the 3D transformation is normally realized by the combination of a 2D DCT and a 1-D Walsh-Hadamard transformations in the hardware design. The 2D DCT is applied separately to each individual patch within the 3D group, while the Walsh-Hadamard transform is applied among the multiple patches. In order to reduce the arithmetic complexity of the 2D DCT procedure, we have implemented two fast 1-D DCT units [16], each of which only requires 11 multiplications, to implement the required 2D DCT transformation. Fig. 10 shows the internal hardware

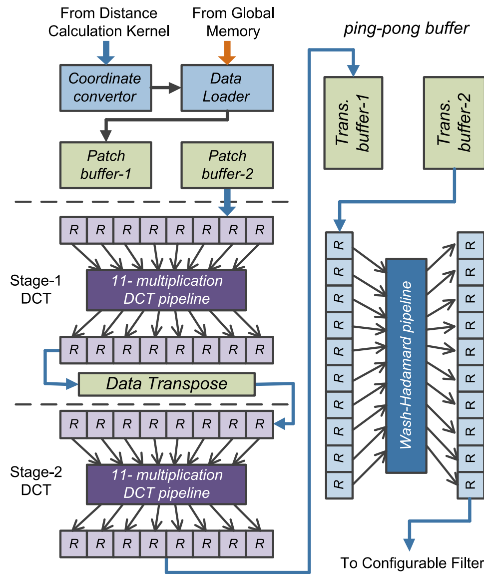


FIGURE 10. The hardware architecture of the 3D transform/inverse transform circuit.

structure of the 3D Transform/Inverse Transform kernels. Since a series of filtering operations have to be conducted in the pipeline, both the 1-DCT and Walsh-Hadamard transform/inverse transform are performed in single-precision floating-point format to guarantee the precision of the final results. In order to maintain a high throughput data flow, the two DCT units are fully pipelined, each of which consumes 37 DSP blocks (11 for multiplication and 26 for additions/subtractions), while the Walsh-Hadamard unit consumes 64 DSP blocks. Moreover, we have designed a pair of ping-pong buffers before each of the transform unit to support concurrent image data loading and processing, and improve the efficiency of the hardware pipelines.

C. AGGREGATION KERNEL

Division operation is required in both generation of the weights as defined in (7) and (8), and the weighted average of the overlapped pixels. Implementing division operation in FPGA logic is very costly and it may also have a negative impact on the working frequency of the accelerator. Hence, we propose to use a look-up table based low-cost approach to approximate the division operation. Our strategy is to store the reciprocal function $\frac{1}{n}$ into a look-up table, and convert the division operations to multiplications. For instance, in the basic estimation stage, the 3D group is of the size $8 \times 8 \times 16$, and consequently the largest possible non-zero value of N_R^{1st} is 1024. Since σ^2 is a predefined constant, we could store the value of $\frac{1}{\sigma^2}, \frac{1}{2\sigma^2}, \frac{1}{3\sigma^2}, \ldots, \frac{1}{1024 \cdot \sigma^2}$ in a look-up table with 1024 entries. During the aggregation computation, the input divisor $\sigma^2 N_R^{1st}$ is first normalized to the range of [0, 1024) by adding a bias to its exponent, and then converted from the floating-point format to an integer with 10-bit precision by using the OpenCL build-in convert function. Finally, the table lookup is addressed by the converted integer to fetch the corresponding coefficient.

V. IMPLEMENTATION RESULTS

A. EXPERIMENTAL SETUP

To evaluate the performance of the proposed accelerator, we have implemented the design on Intel's A10 FPGA development board. The on-board FPGA is an Arria-10 GX1150 device, which has 1150K logic elements, 1518 DSP blocks and 66Mb on-chip memory resources. A single DDR4 SDRAM is attached to the FPGA providing 19.2 GB/s global memory bandwidth. The host machine is equipped with an Intel i7-6700K CPU and 64GB memories. The proposed architecture was modeled in OpenCL kernel codes and compiled by using Intel OpenCL SDK v20.1. The FPGA accelerator executes the entire block-matching and 3D filtering algorithm, while the host program loads image files from the hard drive and sends the data to FPGA through OpenCL APIs. A NVIDIA GeForce Titan-Xp GPU was also installed on the host machine to implement two reference GPU-based software accelerators of [8] and [9]. The hardware specifications of the FPGA and GPU boards are listed and compared in Table 1.

TABLE 1. Hardware specifications of the FPGA and GPU boards.

Board	Intel A10 Dev. Kit	Nvidia Titan-Xp
Technology	20 nm	16 nm
Frequency	200 ∼ 400 MHz	1582 MHz
DSP/CUDA Cores	1518	3840
DDR Bandwidth	19.2 GB/s	547.7 GB/s
Power	30W	250W

In the final design, the size of the search window was configured as 33×33 pixels, while the group size N_{step} in both stages was set with a constant value of 16. The threshold τ_{match} for grouping was different between the first and second phases, i.e., $\tau_{match}^{1st} = 2500$ and $\tau_{match}^{2nd} = 400$. In addition, the step size p_{step} of the reference blocks was 3 within the whole image, and the patch size K was set with an value of 8. For collaborative filtering, we select $\lambda^{3D} = 2.7$ when $\sigma < 40$, otherwise $\lambda^{3D} = 2.8$.

B. PERFORMANCE AND RESOURCE ANALYSIS

As discussed in Section IV-A3, two design parameters M and N are defined in the proposed architecture to control the number of parallel PEs used in the Distance Calculating and Comparison kernels. The highest attainable performance of the final design is bounded by the FPGA DSP resource and the corresponding parameter configuration is $M = 8$ and $N = 5$. The working frequency of the best implemented design on the Intel Arria-10 GX1150 FPGA is 233 MHz. We have measured the average processing time for both gray and color images with different input sizes and report the performance results in Fig. 11. It can be observed that color image generally takes around $2.1\times$ the processing time than gray image. In theory, our implementation can also

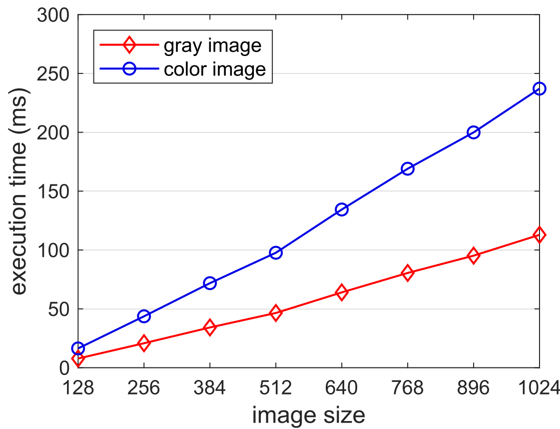
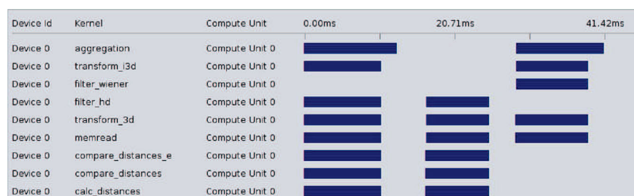


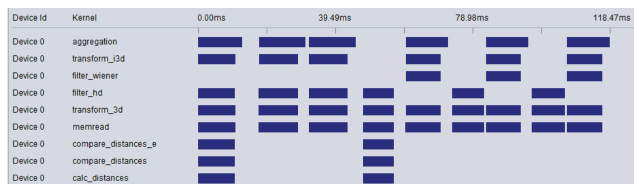
FIGURE 11. Average processing times of images of a different size ($\sigma = 25$).

be used for larger images, and the execution time increases proportionally with the image resolution. However, due to bandwidth and on-chip RAM constraints, the implemented design cannot handle arbitrarily large images like GPUs.

We further used the Intel OpenCL SDK [22] to profile the detailed execution time that each kernel has spend when performing the whole BM3D denoising algorithm. Fig. 12 reports the results for a design with parameter configuration of $M = 4, N = 2$ and a 256×256 input color image. Note that the real processing time without kernel profiling will be significantly lower than which is shown in Fig. 12 because of the delay involved with kernel profiling itself. It can be seen that all kernels run in parallel forming a deep kernel pipeline, which exhibits a very high utilization of the hardware resource (measured occupancy of the PE is 80% $\sim$ 85%) and avoids unnecessary transmitting of intermediate filtering result between FPGA and external memory. Due to the latency of the hardware pipeline, the aggregation kernel takes a slightly longer time to finish than other kernels. In



(a) gray image.



(b) color image.

FIGURE 12. Profiled execution time of each kernel. The example shown is for design with hardware parameters $M = 4$, $N = 2$, and $\sigma = 25$.

the first stage, block-matching (i.e., distance calculation and comparison) is only conducted on the channel Y, while in the second stage, block-matching is performed on the basic estimate.

Fig. 13 reports the detailed hardware resource utilizations, including logic, DSP and on-chip memory blocks, of each kernel type of the best implemented design. It can be seen from the result that 99% of the on-chip DSP resource have been consumed, while there are still plenty of logic and memory resources available. The PE array in the Distance Calculation kernel contributes the largest portion (82%) of the total DSP usage. We could conclude that the block-matching procedure, more specifically the distance calculation computation, is the bottleneck in accelerating the BM3D algorithm on FPGA.

C. COMPARISON WITH GPU ACCELERATORS

In order to compare with GPU-based software accelerators, we have measured the performance of two open-source designs of [8] and [9] by using the Nvidia GeForce Titan-Xp GPU installed on the same host machine. The following parameters, including K (patch size), p_{step} (stride of the patches), τ_{match} (distance threshold) and λ^{3D} (hard-thresholding limit), were set with the same values as the FPGA accelerator. The only difference it that the GPU designs adopt a search window of 39×39 due to the limitation of the software design, whereas our design used a window size of 33×33. Table 2 compares the measured performance data with the proposed FPGA-based accelerator. It can be observed that the proposed design achieves an average speedup of 20% for both gray and color images when compared to the CUDA-based GPU design of [9]. When compared to the OpenCL-based GPU design, our approach has a $14\times$ notable advantage on processing speed. On the other hand, by taking into account the power consumption data shown in Table 1, one could see that our design has a significant $8.3\times$ advantage in energy reduction over GPU-based accelerators. the proposed FPGA accelerator operates at a $6.8\times$ lower working frequency and consists of $2.5\times$ computational resource, which reflects that the proposed systolic-like pipelining architecture is more efficient than the SIMD-based GPU architecture. This makes our work

TABLE 2. Comparison of the execution time ($\sigma = 25$).

Image type	Image size	Method	Execution time (ms)	
			One-step	Two-step
gray	256×256	GPU (OpenCL)	163.8	362.7
		GPU (CUDA)	15.3	25.1
		FPGA	7.5	20.8
	512×512	GPU (OpenCL)	207.7	662.1
		GPU (CUDA)	25.2	58.8
		FPGA	17.1	46.5
color	256×256	GPU (CUDA)	19.8	46.3
		FPGA	14.5	42.1
	512×512	GPU (CUDA)	39.8	118.2
		FPGA	32.9	97.4

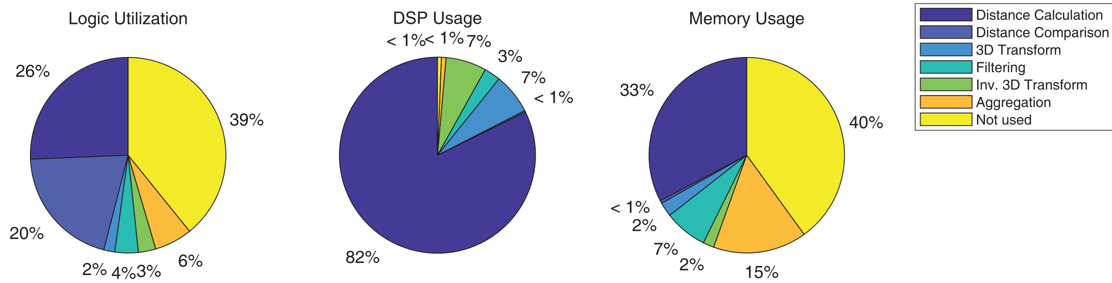


FIGURE 13. **Breakdown of the normalized hardware resource utilization of the best implemented design.**

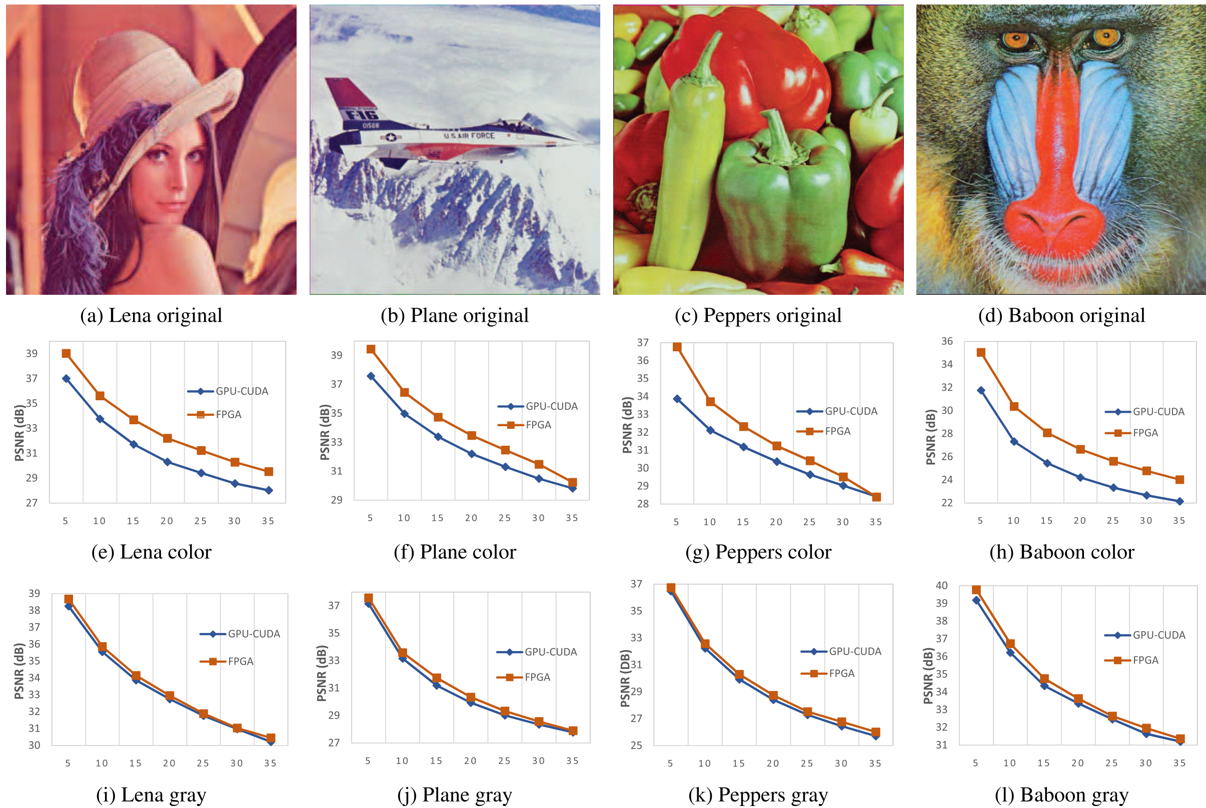


FIGURE 14. **Comparison of the measured PSNR of denoised grayscale and color images.**

an idea choice for embedded applications, such as surveillance system, robotics, etc.

Both the proposed FPGA accelerator and the GPU-based reference designs have conducted several approximations on the original BM3D algorithm to reduce the cost of the final implementation. For instance, in the our design, the original floating-point division operation is approximated by fixed-point table look-up, which may introduce computational errors to the denoised image. On the other hand, the GPU design of [9] rounded the basic estimate of the first aggregation step to 8-bit integers immediately after computation, while our design kept the result in 16-bit fixed-point format. These differences in detailed harware/software implementations will result in a slightly different image processing quality.

Therefore, we have also measured the PSNR of the denoised images processed by both GPU and FPGA-based designs and compare the results in Fig. 14. The comparison only covers the case of $\sigma < 40$ since the GPU reference design does not support further larger σ. It is clear that our scheme achieves around $2 \sim 3$ dB higher PSNR than the GPU reference design for all the color images tested, while the image quality for gray images are vey close. The result indicates that the precision of the basic estimate may has a large impact on the image quality of the U/V channels.

VI. CONCLUSION

In this paper, we propose an FPGA-based hardware accelerator for block-matching and 3D filtering algorithm. A deeply pipelined OpenCL kernel architecture together with a line-buffer-based on-chip data caching scheme were developed to maximize data reuse and reduce external memory bandwidth. The compute-intensive block-matching procedure was accelerated by a systolic-like parallel PE array structure, which efficiently exploits fine-grained data-level parallelism of the block-matching algorithm. The best implemented design on an Arria-10 GX1150 FPGA achieved 20% processing speed improvement and slightly better denoising quality when compared to a state-of-the-art software design on Nividia Titan-Xp GPU. The most distinguish advantage of the proposed scheme is the 8.3× improvement in power dissipation. Thus, the proposed approach is especially suitable for embedded application scenarios, in which energy efficiency is the most critical design concern.

REFERENCES

[1] S. Zhang and R. L. Stevenson, "Inertia sensor aided alignment for burst pipeline in low light conditions," in *Proc. 25th IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2018, pp. 3953–3957.

[2] V. Estellers, S. Soatto, and X. Bresson, "Adaptive regularization with the structure tensor," *IEEE Trans. Image Process.*, vol. 24, no. 6, pp. 1777–1790, Jun. 2015.

[3] T. Yu, X. Wang, and A. Shami, "UAV-enabled spatial data sampling in large-scale IoT systems using denoising autoencoder neural network," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1856–1865, Apr. 2019.

[4] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-D transform-domain collaborative filtering," *IEEE Trans. Image Process.*, vol. 16, no. 8, pp. 2080–2095, Aug. 2007.

[5] F. Chen, L. Zhang, and H. Yu, "External patch prior guided internal clustering for image denoising," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 603–611.

[6] *C-Programm for a BM3D Denoising Algorithm*. Accessed: Aug. 1, 2019. [Online]. Available: https://github.com/20logTom/BM3D

[7] M. Lebrun, "An analysis and implementation of the BM3D image denoising method," *Image Process. Line*, vol. 2, pp. 175–213, Aug. 2012.

[8] S. Sarjanoja, J. Boutellier, and J. Hannuksela, "BM3D image denoising using heterogeneous computing platforms," in *Proc. Conf. Design Archit. Signal Image Process. (DASIP)*, Sep. 2015, pp. 1–8.

[9] D. Honzátko and M. Kruliš, "Accelerating block-matching and 3D filtering method for image denoising on GPUs," *J. Real-Time Image Process.*, vol. 16, no. 6, pp. 1–15, 2017.

[10] A. Davy and T. Ehret, "GPU acceleration of NL-means, BM3D and VBM3D," *J. Real-Time Image Process.*, pp. 1–18, Feb. 2020.

[11] *Nvidia TESLA v100 Architecture*. Accessed: Jun. 20, 2019. [Online]. Available: https://im ages.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper

[12] M. Ravi, A. Sewa, S. T. G., and S. S. S. Sanagapati, "FPGA as a hardware accelerator for computation intensive maximum likelihood expectation maximization medical image reconstruction algorithm," *IEEE Access*, vol. 7, pp. 111727–111735, 2019.

[13] C. Li, Y. Bi, F. Marzani, and F. Yang, "Fast FPGA prototyping for real-time image processing with very high-level synthesis," *J. Real-Time Image Proc.*, vol. 16, pp. 1795–1812, Apr. 2017.

[14] S. Nakasone, L.-G. Ofverstedt, G. Wilken, and U. Skoglund, "An OpenCL implementation of an image filter on FPGA," in *Proc. IEEE 5th Int. Conf. Comput. Commun. (ICCC)*, Dec. 2019, pp. 272–276.

[15] J. Wang, B. Li, and K. Xing, "A new real-time lucky imaging algorithm and its implementation techniques," *IEEE Access*, vol. 8, pp. 52192–52208, 2020.

[16] C. Loeffler, A. Ligtenberg, and G. S. Moschytz, "Practical fast 1-D DCT algorithms with 11 multiplications," in *Proc. Int. Conf. Acoust., Speech, Signal Process.*, May 1989, pp. 988–991.

[17] X. Wang, K. Xu, and D. Wang, "Accelerating block-matching and 3D filtering-based image denoising algorithm on FPGAs," in *Proc. 14th IEEE Int. Conf. Signal Process. (ICSP)*, Aug. 2018, pp. 235–240.

[18] I. Firmansyah and Y. Yamaguchi, "OpenCL implementation of FPGA-based signal generation and measurement," *IEEE Access*, vol. 7, pp. 48849–48859, 2019.

[19] F. B. Muslim, L. Ma, M. Roozmeh, and L. Lavagno, "Efficient FPGA implementation of OpenCL high-performance computing applications via high-level synthesis," *IEEE Access*, vol. 5, pp. 2747–2762, 2017.

[20] Y. Pu, J. Peng, L. Huang, and J. Chen, "An efficient KNN algorithm implemented on FPGA based heterogeneous computing system using OpenCL," in *Proc. IEEE 23rd Annu. Int. Symp. Field-Program. Custom Comput. Mach.*, May 2015, pp. 167–170.

[21] *Bitonic Sorting*. Accessed: Jun. 20, 2020. [Online]. Available: https://github.com/mediroozmeh/Biton-ic-Sorting

[22] *Intel FPGA SDK for OpenCL Programming Guide*. Accessed: Jun. 26, 2020. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/documentation/-mwh1391807965224.html

DONG WANG (Member, IEEE) was born in Xianyang, Shanxi, China, in 1981. He received the B.S. and Ph.D. degrees in information engineering and control science from Xi'an Jiaotong University, China, in 2004 and 2010, respectively.

From 2010 to 2013, he was a Postdoctoral Researcher with the Institute of Microelectronics, Tsinghua University. Since 2013, he has been an Associate Professor with the Institute of Information Science, Beijing Jiaotong University. He was a Visiting Scholar with the Department of Electrical and Computer Engineering, University of California at Davis, from 2018 to 2019. His research interests include computer arithmetic for reconfigurable devices and high performance and energy efficient computing architectures for embedded and machine learning applications.

JIA XU was born in Chifeng, Nei Mongol, China, in 1997. He received the B.S. degree from Beijing Jiaotong University, China, in 2019. He is currently pursuing the M.E. degree in signal and information processing. His current research interests include heterogeneous computation on reconfigurable devices and neural network accelerators.

KE XU was born in Weifang, Shandong, China, in 1993. He received the B.S. degree from the Hefei University of Technology, China, in 2016. He is currently pursuing the Ph.D. degree with the Institute of Information Science, Beijing Jiaotong University, Beijing, China. His research interests include neural network compression, high-performance computing architectures for embedded applications, and computer vision.