Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2008, Article ID 438051, 14 pages
doi:10.1155/2008/438051

Research Article

Real-Time Target Detection Architecture Based on Reduced Complexity Hyperspectral Processing

Kyoung-Su Park,[1] Shung Han Cho,[1] Sangjin Hong,[1] and We-Duke Cho[2]

[1] *Mobile Systems Design Laboratory, Department of Electrical and Computer Engineering, College of Engineering and Apllied Sciences, Stony Brook University-SUNY, Stony Brook, NY 11794-2350, USA*
[2] *Division of Electrical & Computer Engineering, College of Information Technology, Ajou University, Suwon 443-749, South Korea*

Correspondence should be addressed to Sangjin Hong, snjhong@ece.sunysb.edu

Received 30 May 2007; Revised 10 October 2007; Accepted 28 March 2008

Recommended by Mark Kahrs

This paper presents a real-time target detection architecture for hyperspectral image processing. The architecture is based on a reduced complexity algorithm for high-throughput applications.We propose an efficient pipelined processing element architecture and a scalable multiple-processing element architecture by exploiting data partitioning. We present a processing unit modeling based on the data reduction algorithm in hyperspectral image processing and propose computing structure, that is, to optimize memory usage and eliminates memory bottleneck. We investigate the interconnection topology for the multipleprocessing element architecture to improve the speed. The proposed architecture is designed and implemented in FPGA to illustrate the relationship between hardware complexity and execution throughput of hyperspectral image processing for target detection.

1. INTRODUCTION

Hyperspectral processing technology is a powerful tool for detecting chemical substances, anomalies, and camouflaged objects, as well as for visual surveillance. These applications require low complexity and high throughput. Traditional hyperspectral image processing uses hundreds of bands to detect or classify targets, in which the computational complexity is proportional to the amount of data that needs to be processed. Thus for real-time execution, the data reduction and simplified algorithm are very critical. The computational complexity of the hyperspectral processing can be reduced by exploiting spectral content redundancy using a partial number of bands [1–3]. However, the amount of data to be processed in hyperspectral image processing is still large compared to that in a typical image processing.

General purpose processors and/or field-programmable gate arrays (FPGAs) have been used for real-time hyperspectral processing. The COMPASS hyperspectral sensor system was presented in [4]. The system has the data processing computer (DPC) and the operator display/control computer. Thus the high-performance DPC executes real-time calibration and multiple spectral detection on 13 G4-processors [4]. Principal components transformation (PCT) compresses the redundant information between hyperspectral bands [5, 6]. The PCT-based real-time compression was implemented in 12 SHARCs [7], where each SHARC has a link port for efficient data flow. The wavelet-based dimension reduction algorithm was presented in [8, 9]. Since the wavelet-based method transforms the spectrum of a pixel to a linear and weighted combination of pixels, the reduced bands have the most important features of original spectral bands. The algorithm was implemented as reconfigurable computers (RCs) in FPGA. Parallel-independent component analysis (pICA) algorithm was implemented for hyperspectral images in FPGA, in which 50 bands were selected from the maxima, the minima, and the inflection points. The system used three reconfigurable components to compensate for the performance limit of a single FPGA. However, the implemented system cannot be applied in real-time application [10]. K-means unsupervised clustering algorithm was implemented in FPGA. The algorithm was transformed to use alternative distance measures and truncation of the input data and cluster centers. The design has a 10 stage pipeline, but does not incorporate data partitioning which could improve the processing time [11, 12].

This paper presents a real-time target detection architecture as an efficient solution for hyperspectral image processing. The architecture uses our proposed complexity reduction algorithm by which the computational complexity is significantly reduced from the effective band selection and library refinement [3]. Since the index of effective bands and refined libraries are dynamically changed, the algorithm has a benefit for detection. However, the algorithm is sequential since the detection needs the updated band index and refined libraries for an image. Thus we propose an efficient pipelined processing element architecture which minimizes the effect of the sequential algorithm. We present the processing unit model based on the data reduction algorithm and then propose a computing structure that can help to optimize memory usage and eliminates memory bottleneck. Also we present a scalable multiple-processing element architecture by exploiting data partitioning. We propose an interconnection topology for the multiple-processing element architecture to improve the speed. Compared to the general purpose processor, FPGA implementations particularly suit for the implementation of hyperspectral processing since the FPGA permits parallel implementation and proper bandwidth [8, 11]. Thus the proposed architecture is designed and implemented in FPGA to illustrate the relationship between hardware complexity and execution throughput of the hyperspectral image processing for the target detection.

The remainder of this paper is organized as follows: Section 2 describes the overview of hyperspectral image processing applied to the effective band selection and the library refinement scheme. The image data structure as well as the processing data flow is described. We present an architecture design for real-time hyperspectral image processing in Section 3 and the architecture is verified in Section 4. Section 5 concludes the paper.

2. DESIGN OVERVIEW

2.1. Overall processing

While conventional image pictures are represented by 2-dimensional matrices, the hyperspectral image has one more dimension for band spectral data as shown in Figure 1. Collected data by hyperspectral image sensors are kept as one cube and each pixel which is located at (x, y) has N_z bands. Notations N_x and N_y are used for indicating the total size of pixels in accordance with the axis. Since the number of spectral bands presents high-computational complexity, the real-time hyperspectral image processing is a big challenge.

The hyperspectral image processing involves three key stages denoted as preprocessing, processing, and postprocessing [13–16]. The spectrum contents from sensors are stored in a cube memory structure as raw image data as shown in Figure 1. The raw image data is calibrated by the preprocessing [17]. Each cube contains large numbers of bands which represent the characteristics of a target material. In the processing, target images are detected by isolating the portion of data while it is highly correlated with the target library. The target library contains spectral information about the object that it is intended to detect. The objective

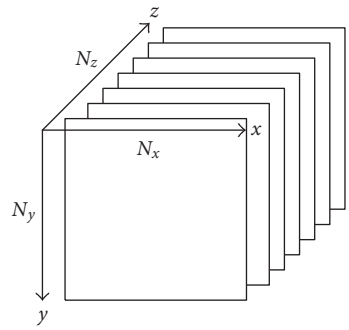


Figure 1: Illustration of cube data structure.

of the processing is to find out the target image from the input cubes that correlate with spectral information stored in the target library. The third step is the postprocessing where actual detected images are displayed with RGB colors.

The main challenges for the hyperspectral image processing are high volume and complexity of the hyperspectral image data. For the real-time processing, the complexity should be reduced. The easiest approach is to reduce the number of bands and the amount of library for processing. However, such reductions may eliminate the merit of the hyperspectral image processing. If certain bands have more characteristics to represent the object, all spectrums of bands do not need to detect the target. Thus our approach determines bands that are more effective for the target detection. The performance of detection depends on the quality of spectral information stored in the target library. A perfect target library does not exist since objects exhibit different spectral characteristics which are sensitive to environmental factors such as lighting [13–15].

The main operation in the hyperspectral image processing for target detection is to compare input cube images with the target library, which determines the correlation coefficient in terms of spectrum contents [3]. Hence the main operation in hyperspectral image processing is the calculation of correlation. The correlation coefficient is a measure of similarity between two spectrum contents which are stored in the target library and obtained from input images. High values of the correlation coefficient between two spectrum contents indicate the high degree of similarity between the two spectrum contents.

To exploit the correlation, several types of correlation functions have been introduced such as spectral angle mapper (SAM), Euclidean minimum distance (EMD), and the information theoretic approach [1, 18–20]. The distance metrics such as SAM and EMD provide a unique measure of distance from two spectrum contents. SAM is invariant to multiplicative scaling [1, 18]. Since the refined library represents the luminescent variation of the basic library, the invariance is particularly important for our proposed library refinement scheme. Thus the correlation coefficient is presented as

$$\mathrm{A} = 1 - \cos^{-1}\left(\frac{\sum_{i=1}^{N_E} t_i r_i}{\sqrt{\sum_{i=1}^{N_E} t_i^2}\sqrt{\sum_{i=1}^{N_E} r_i{}^2}}\right), \tag{1}$$

where N_E is the number of effective bands, t_i is the test spectrum of ith band, and r_i is the reference spectrum of ith band.

Figure 2 illustrates the overall algorithm for detecting and isolating target images. We apply the effective band selection scheme to reduce the number of bands applied to the detection. For the scheme of effective band selection, we have defined a contribution factor to represent the isolation effectiveness in terms of the target libraries [3]. To obtain the contribution factor, we need randomly selected background samples which represent the spectral property of background images. Since the correlation represents the variation of differences between two spectrum contents, the effective bands are selected to get the maximally separated contribution value. The algorithm has two processing flows. The right side is mainly related to the detection which compares the input image with the library. The left side is the update which has the library refinement and the effective band selection.

Each operation is specified by Steps. Step 0, 1, and 2 exist for the detection and others perform the update. Step 0 loads the index of effective bands from Step 5 and then chooses spectrum contents of an input pixel and a library for effective bands. Step 1 has a loop to get the correlation coefficient (A) and the loop size is $N_{\mathrm{LIB}}N_E$. Step 2(a) is for target detection and Step 2(b) is for background detection. In Step 2(a), if the correlation coefficient (A) is over the minimum correlation coefficient between the library and the input image (A_t), the pixel is detected as a target and the spectrum contents in the pixel are reserved for the library refinement. On the other hand, in order to choose background samples, the correlation coefficient (A) is compared to the maximum correlation coefficient between input image and background (A_b) in Step 2(b). Step 3 corrects samples for background and target. For the representation of the spectrum of background area, the background samples are randomly selected. The library is refined in Step 4 and the effective bands are selected by using the contribution coefficient in Step 5.

There are several floating point operations such as root and arccosine functions in Step 1. The detection function in Step 2 also has floating point operations to compare the correlation coefficient (A) with A_t and A_b. However, the output of Step 2 has integer data type. Step 4 and Step 5 have floating point operations.

As discussed in [3], Step 1 has the highest computational complexity. However, Step 3, 4, and 5 have less complexity than Step 1. The complexity is proportional to the number of effective bands and the number of libraries. However, the number of target and background samples does not relate to the overall complexity.

2.2. Design issues and approach

The objectives of design are to assure high-speed operation for detection and update. Our algorithm has two kinds of data dependency. First, update and detection are sequential. The results of the update are the index of effective bands and refined libraries which is used in the detection. Thus the detection cannot start until the operation of the previous cube image is completed. The sequential property prevents the parallel operation of update and detection. Second, the update needs target and background samples, but the type of samples is decided after the detection. Therefore, update and detection require two different cube images. Once we construct the pipeline structure, the architecture is insensitive to these data dependencies and improves the execution speed. However, the pipeline structure increases resource usage. We investigate two types of pipeline structure denoted pixel-based and cube-based pipelines. Thus the proposed architecture optimizes the memory usage and eliminates the memory bottleneck.

The correlation function has both fixed and floating point operations. The computational complexity of floating point operation is important for high-speed operations. We investigate the implementation constraints from the timing relationship between floating and fixed point operations. Since our target architecture has limited floating point units, we verify the sharing of floating point units in update and detection.

To improve the execution time, we use data partitioning which exploits a scalable architecture. We present an interconnection topology and update sharing among the processing elements.

3. ARCHITECTURAL DESIGN SPACE

3.1. Algorithm characteristics

3.1.1. Execution dependency

To express the execution dependency of the hyperspectral image processing, a functional graph is shown in Figure 3. Step 0 has two functions of load() and init(). The function load() corrects spectrum contents for effective bands from the preprocessing and refined library. The function init() loads the index of effective bands from the get_eb() in Step 5. Thus the function init() operates on each cube, but the function load() works on each pixel. Step 1 has the function acc() and the function corr(). The function acc() accumulates the inputs for effective bands, where the accumulator has two kinds of operations denoted multiplication and summation in (1). Thus the outputs of the function acc() are three fixed point numbers for $\sum t_i^2$, $\sum r_i^2$, and $\sum t_i r_i$. The function corr() calculates the correlation coefficient A which is a floating point number. In Step 2, the function detect() verifies the type of pixel using correlation coefficient A. Thus the main operations of the function detect() are floating point operations. In Step 3, once the type of pixel is decided in the function detect(), the function sample() corrects the sample in Step 0. The function choose_samples() chooses target and background samples. In Step 4, the functions loads(), accs(), corrs(), dets(), and saves() refine the library from the target samples. These functions are very similar to load(), acc(), corr(), detect(), and sample(). The functions diff(), cont(), and get_eb() in Step 5 find the index of the effective bands using the contribution factor. These functions require both floating point and fixed point operations.

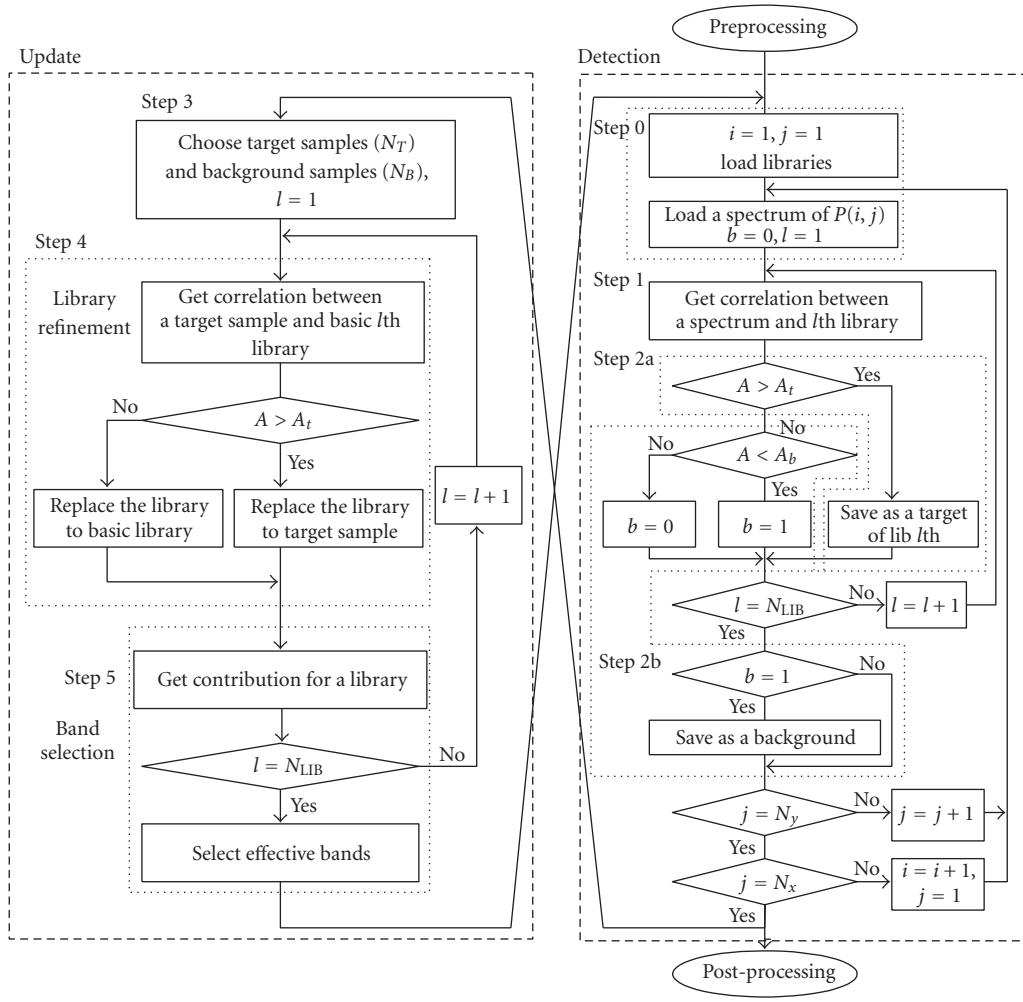


Figure 2: Flowchart of the processing.

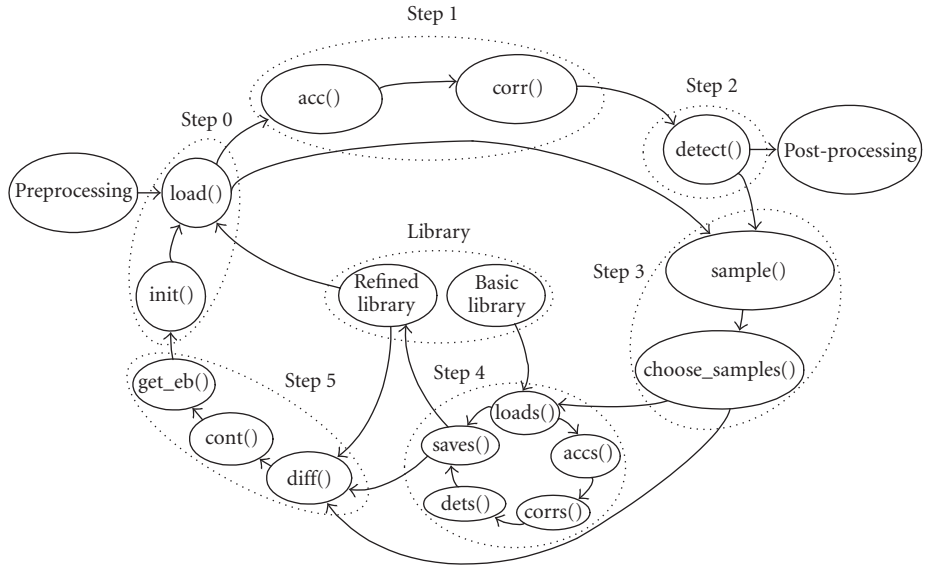


Figure 3: Illustration of functional graph.

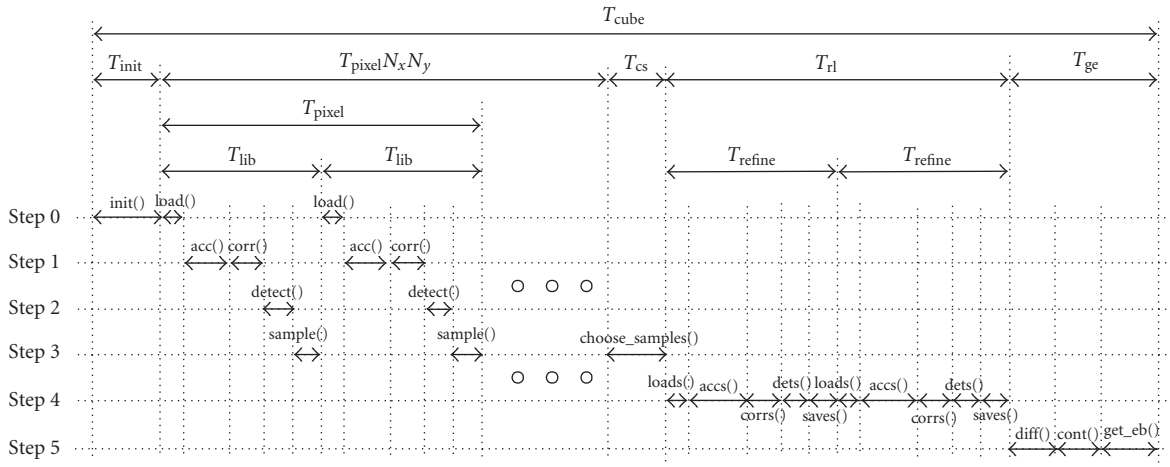


Figure 4: Timing flow in the processing where $N_{LIB} = 2$.

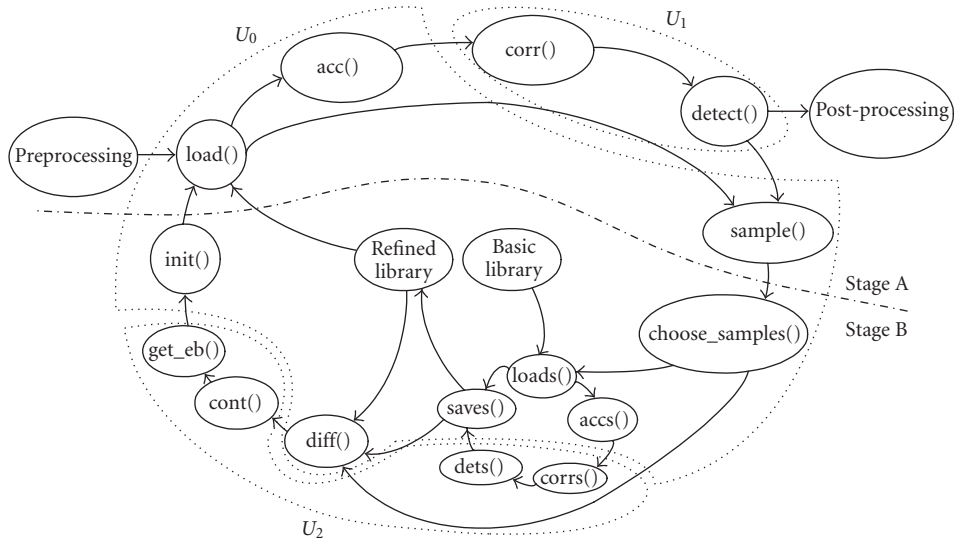


Figure 5: Illustration of functional partitioning.

Figure 4 shows the overall execution timing flow of our algorithm. T_{load}, T_{acc}, T_{corr}, T_{detect}, and T_{sample} denote the execution time for the function load() in Step 0, acc() in Step 1, corr() in Step 1, detect() in Step 2, and sample() in Step 3, respectively. The execution time for a pixel denoted as T_{pixel} is represented as $(T_{load} + T_{acc} + T_{corr} + T_{detect} + T_{sample})N_{LIB}$. T_{init}, T_{rl}, T_{cs}, and T_{ge} denote the execution time for the function init() in Step 1, all functions in Step 4, the function choose_samples() in Step 3, and all functions in Step 5, respectively. Note that the execution time for a cube (T_{cube}) is represented as $T_{init} + T_{pixel}N_xN_y + T_{cs} + T_{rl} + T_{ge}$, where N_xN_y represents the spatial resolution of the hyperspectral image cube. Therefore, bigger spatial resolution requires longer execution time.

The functions in the processing have execution dependencies. The functions acc() and sample() use the same data from the preprocessing, but the operations of function samples() cannot be completed before the function detect() has results. The functions in Step 4 load data from the function choose_sample(), but the function in Step 5 cannot start before the operations in Step 4 are done. Therefore, after the detection is completed, the functions in the update can be processed.

There are two types of operation denoted by a pixel-based function and a cube-based function. The functions load(), acc(), corr(), detect(), and sample() are pixel-based functions and the functions init(), choose_samples(), loads(), accs(), corrs(), dets(), saves(), diff(), cont(), and get_eb() are cube-based. The pixel-based operations execute N_xN_y times for a cube, but the cube-based functions execute only once for a cube. Therefore, the pixel-based operations are more significant than the cube-based operations for high-speed execution.

The functions have two kinds of operations presented as fixed and floating point operations. Figure 5 illustrates the functional partitioning, where fixed point operations

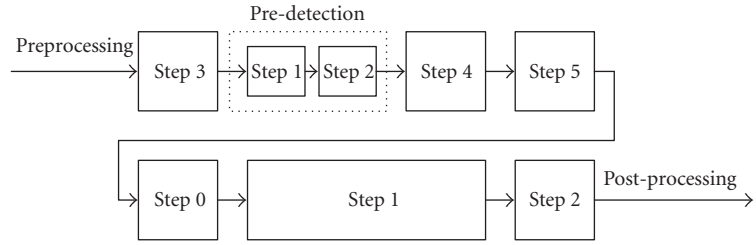


Figure 6: Illustration of block diagram of the processing without cube delay.

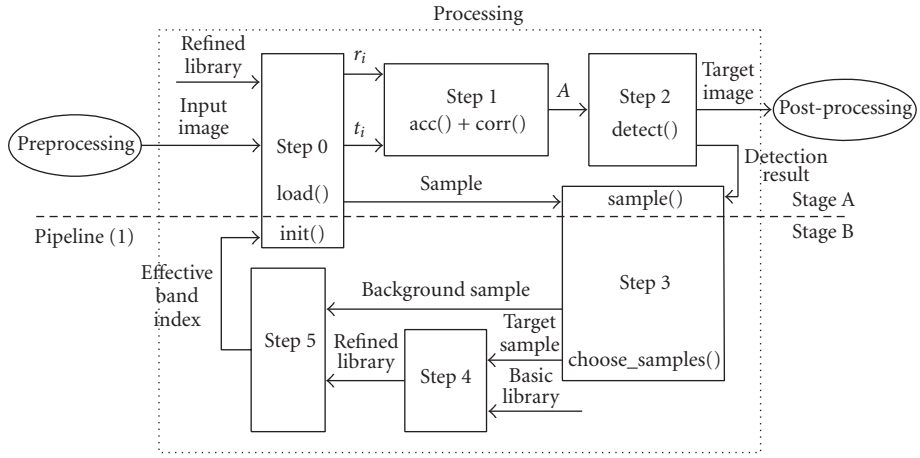


Figure 7: Illustration of block diagram of the processing which uses the two-stage pipeline.

and floating point operations are separated with U_0, U_1 and U_2. U_1 contains the function corr() and detect() which are mainly related to the detection. U_2 contains the functions of dets(), corrs(), cont(), and get_eb().

3.1.2. The effect of cube delay

In Figure 5, the pixel-based functions and the cube-based functions are separated into Stage A and Stage B. The update requires $N_T N_{\text{LIB}}$ target samples and N_B number of background samples. Since the results of the update can be used from the next cube, the processing flow requires a two-stage pipeline. However, the two-stage pipeline is the reason of cube delay (i.e., the time corresponding for transferring one cube image data). The cube delay decreases the performance of detection since the refined libraries and the effective bands index may not be available in the next cube image. However, since consecutive cube images have similar spectrum properties, if the cube delay time ($T_{\text{cube}} N_{\text{cube\_delay}}$) is faster than the change of spectral properties (T_{spectral}), the cube delay can be allowed. $N_{\text{cube\_delay}}$ denotes the number of cube delay. Note that, T_{spectral} depends on the application of hyperspectral image processing. For example, once the hyperspectral image is used to detect a person in a surveillance system, the basic properties of the light source is not abruptly changed. Therefore, the cube delay is not significant.

In order to remove the cube delay, the predetection step can be used in the processing. Figure 6 illustrates the block diagram of the processing without a cube delay, where all steps use the same cube image. Since the complex reduction schemes require background and target samples, the predetection composed of Step 1 and Step 2 is necessary to verify the randomly selected samples for all bands. Since N_T number of target samples are required for the library refinement, the predetection chooses and verifies bigger number samples than N_T target samples to get N_T number of target samples. However, if the number of detected target pixels in the entire cube image is smaller than N_T target samples, a cube image is necessary for the predetection. In this case, the benefit of the effective band selection disappears. Therefore, we use the two-stage cube-based pipeline.

3.2. Single-processing execution model

3.2.1. Two-stage pipeline

To remove the data dependency between the update and detection, we use the two-stage pipeline structure. Figure 7 shows the block diagram of the processing. *Pipeline(1)* separates the block diagram of the processing into Stage A and Stage B.

In the two-stage pipeline structure, the execution time for a cube (T_{cube}) is represented as

$$T_{\text{cube}} = \max\left(T_A, T_B\right), \tag{2}$$

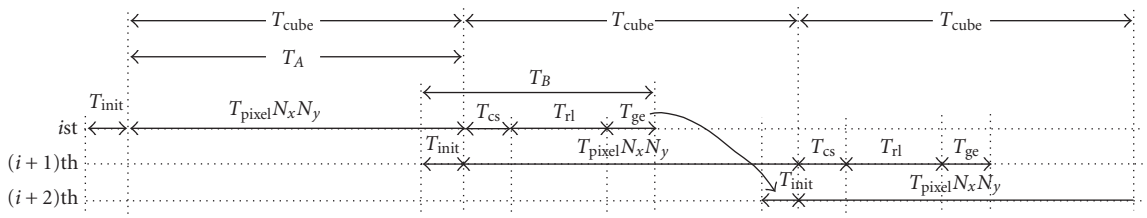


Figure 8: Illustration of timing flow in the two-stage pipeline.

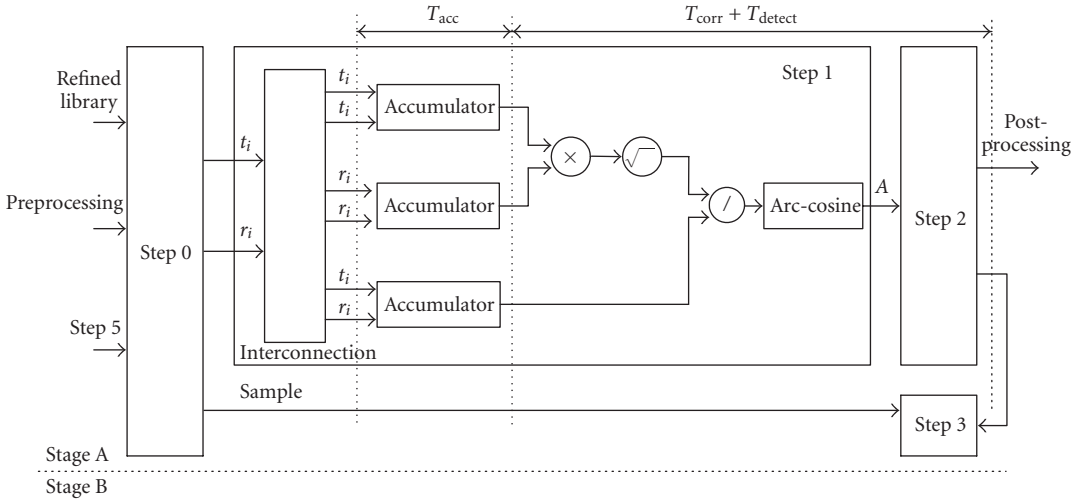


Figure 9: Illustration of pixel-based execution pipeline structure in Stage A where three accumulators are used.

where T_{A} and T_{B} represent the execution time for Stage A and Stage B, respectively. T_{A} is expressed as $T_{\text{pixel}}N_xN_y$ and T_{B} is equal to $T_{\text{init}} + T_{\text{cs}} + T_{\text{rl}} + T_{\text{ge}}$. Once reduced number of bands are used for the detection, T_{pixel} is reduced. Thus T_{A} is getting shorter, but T_{B} is not changed. Therefore, once minimum number of effective bands is used, the overall execution time is improved, but the enhanced time is limited by T_{B} since T_{B} is invariant about the number of bands. Figure 8 shows the timing flow in the two-stage pipeline structure. The effective bands and refined library from the ith cube can be applied in $(i+2)$th cube. Thus the two-stage execution pipeline has a two-cube delay.

3.2.2. Pixel-based pipeline

In order to improve the execution time of Stage A, we use an internal pipeline structure presented by the pixel-based execution pipeline. The pixel-based execution pipeline does not have the cube delay, but has a pixel delay between stages. In the pixel-based execution pipeline, the execution time for a pixel (T_{pixel}) is critical for the execution time of Stage A (T_{A}). For example, in Figure 9, the pixel-based execution pipeline structure is used and three accumulators are used. In this figure, the minimum execution time for a pixel is the same as the execution time for an accumulator (T_{acc}).

The objective of the pixel-based execution pipeline is to minimize the execution time for a pixel (T_{pixel}). Figure 10 shows the execution time for a pixel (T_{pixel}). Thus once the pixel-based execution pipeline structure is used, the execution time for a pixel is represented as the following:

$$T_{\text{pixel}} = \left(\frac{3T_{\text{acc}}}{N_{\text{acc}}}\right)N_{\text{LIB}} = (T_{\text{corr}} + T_{\text{detect}})N_{\text{LIB}}, \qquad (3)$$

where $1 \le N_{\text{acc}} \le 3$.

The execution time for floating point operations ($T_{\text{corr}} + T_{\text{detect}}$) can limit the execution time for a cube (T_{cube}) as well as the execution time of the accumulation (T_{acc}). Once the effective band selection algorithm is applied, the execution time of an accumulator (T_{acc}) can be reduced. Therefore, the execution time for the floating point operations ($T_{\text{corr}} + T_{\text{detect}}$) is significant in reduced complexity hyperspectral image processing.

3.2.3. Floating point unit sharing

In Figure 5, both stages have fixed and floating point operations. U_1 and U_2 are the parts of floating point operations in Stage A and Stage B, respectively. For example, if we have two available floating point units (FPUs), then the FPUs can support the floating point operations of Stage A or Stage B. T_{D1} and T_{D2} denote the required times of fixed point operation in Stage A and Stage B. Also, T_{F1} and T_{F2} denote the required times for the floating point operations in Stage A and Stage B, respectively. Note that, as shown in Figure 10, T_{D1} is the same as T_{acc}, and T_{F1} is the summation of T_{corr} and

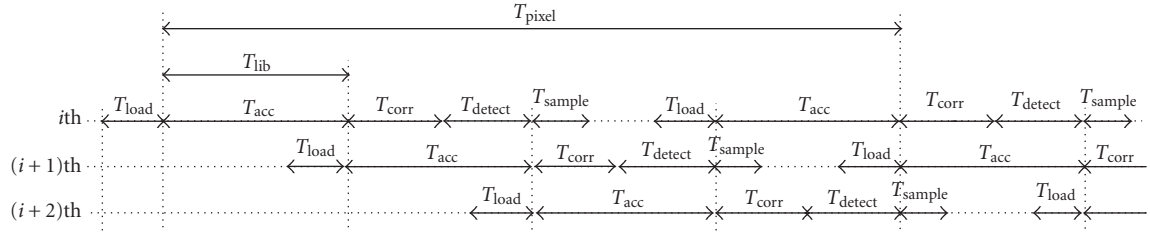
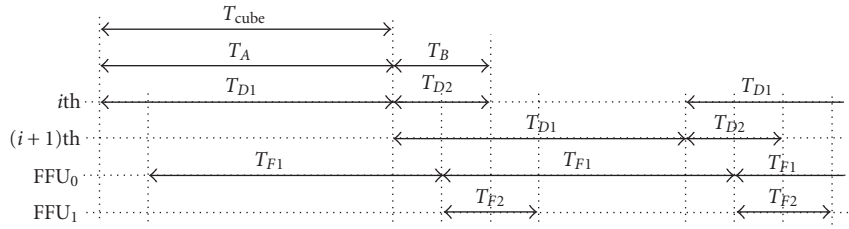
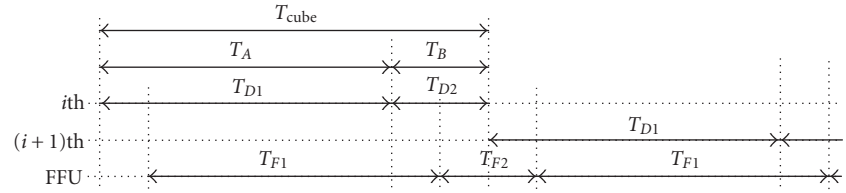


Figure 10: Execution time for a pixel (T_{pixel}) in the pixel-based pipeline structure ($N_{\text{LIB}} = 4$).



(a) The case of two FPUs



(b) The case of one FPU

Figure 11: Execution flow with FPUs.

T_{detect}. In the case of two FPUs, T_{cube} is identified to T_{A} and T_{D1}.

The computational complexity of Stage B is much smaller than that of Stage A. Besides, our target architecture has limited FPUs. Thus we consider the sharing of floating point units. Figure 11 illustrates the execution flow with floating point units (FPUs), where Figures 11(a) and 11(b) use one FPU or two, respectively. When one FPU is available, the execution time for a cube (T_{cube}) is the same as $T_{\text{A}} + T_{\text{B}}$.

3.2.4. Input capacity

The input capacity limits the overall execution time. We define the input capacity $N_{\text{bit}}F_m$, where N_{bit} and F_m denote the input bitwidth and the maximum input frequency, respectively. To assure the execution of the processing, the input capacity ($N_{\text{bit}}F_m$) is bigger than $N_xN_yN_zN_{\text{re}}N_{\text{Th}}$, where N_{re} represents the resolution of a spectrum content and N_{Th} is the throughput, that is, the number of cubes per second.

3.3. Multiple-processing execution model

3.3.1. Data partitioning

The objective of the data partitioning is to reduce the execution time by using the multiple-processing elements (PEs). The type of data partitioning depends on the cube memory structure. Figure 12 shows three kinds of cube data partitioning which have four numbers of PEs. Figures 12(a) and 12(b) separate the area of cubes into 4 banks. Since each PE is connected to a bank memory, the limitation of input capacity is the same as in the single-processing execution model. In Figure 12(c), each pixel is allocated to a different PE. Thus the cube image allocated in a PE is a low-resolution cube image.

3.3.2. The effect of partitioning

The execution time for a cube (T_{cube}) in the multiple-processing execution model is represented as

$$T_{\text{cube}} = \max\left(T_{\text{A}}/N_{\text{PE}}, T_{\text{B}}\right). \tag{4}$$

The data partitioning can improve the execution time. The increased number of PEs affects Stage A since the spatial image area of a PE is proportionally reduced to N_{PE}. Therefore, once N_{PE} is increased, the overall execution time (T_{cube}) is finally limited to T_{B}. Even if the cube is partitioned into several banks, the data type of each PE is still a cube as shown in Figure 13. The cube size in Figures 12(a) and 12(c) is $(N_x/2)(N_y/2)N_z$ and the cube size in Figure 12(b) is $(N_x/4)N_yN_z$.

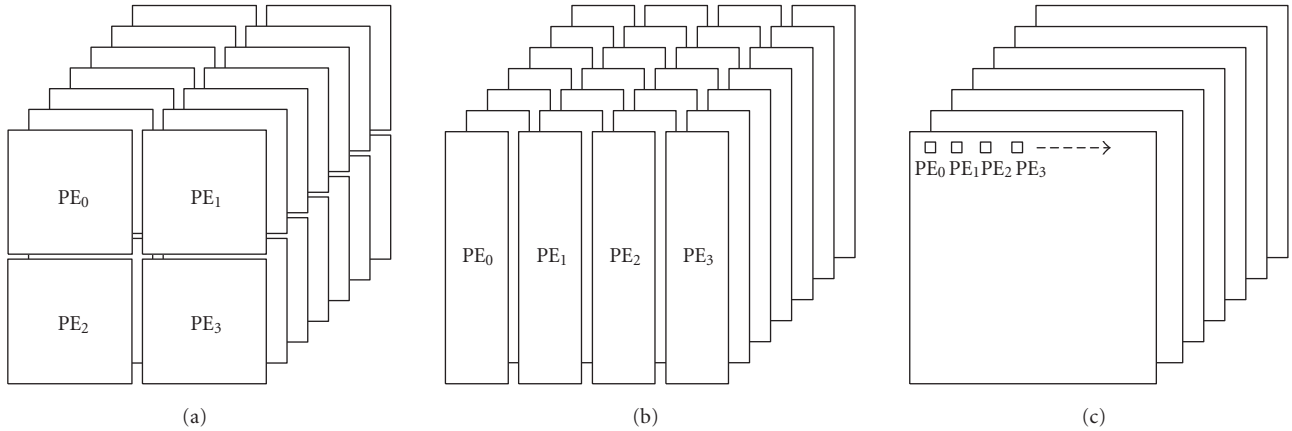


Figure 12: Cube data partitioning where the number of processing elements (N_{PE}) is 4.

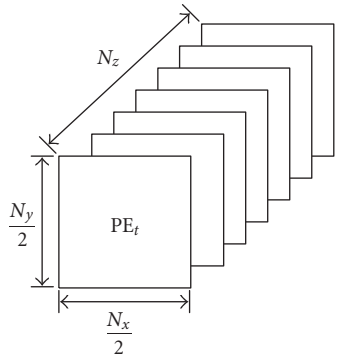


Figure 13: Illustration of cube partitioning.

3.3.3. Update sharing

Figure 14 shows the block diagram in the multiple-data partitioning without the update sharing where a PE is connected with the preprocessing and the postprocessing through the interconnection networks. Stage A has three signals, *initial*, *sample*, and *refined library* that send the index of effective bands and refined libraries and it receives the samples for the update. Thus the update is independent of processing elements.

In the multiple-processing execution model, the interconnection network is one of the reasons for the speed bottleneck since the input capacity follows the increased requirement. The input capacity is related to both the input frequency and the input bitwidth (N_{bit}). Since the input frequency is dedicated to the implemented architecture, the bigger bitwidth increases the speed. However, the bigger input bitwidth also increases the complexity of interconnection. Finally, the speed from applying multiple-processing elements is limited by the interconnection network.

Once the update sharing is necessary, Stage B is shared as shown in Figure 15. To transfer the index of effective bands and the refined library from Stage B to all processing elements, all processing elements need to stop their execution and then execute Stage B for a cube. T_{A} can be improved by N_{PE}, but T_{B} is not changed. The execution time of Stage B limits the speed.

4. ARCHITECTURE EVALUATION

4.1. Single-processing element model

We choose Xilinx FPGA (XC2VP100) device to implement the architecture. The floating point operations are implemented with a Power PC Core (PPC) in FPGA, where the PPC has 350 MHz execution speed. The device has a 7,992 kB block ram which is big enough to support the memory requirement of our algorithm.

Figure 16 describes the overall processing in the single-processing execution model. Most functional blocks of the block diagram are matched with the introduced step functions. FU_0, FU_1, FU_3, FU_4, and FU_5 correspond to Step 0, Step 1, Step 3, Step 4, and Step 5, respectively. PPC_1 takes the operation of Step 2 and the floating point operation in Step 1. Similarly, PPC_2 takes the floating point operations in Step 4 and Step 5.

The fixed point operations and floating point operations are implemented in FPGA logic blocks and the dedicated PPCs in our target architecture, respectively. Besides, if an operation is control driven and the computation complexity of the operation does not highly affect the operation of a PPC, the operations are implemented in the PPC.

Figure 17 illustrates the timing flow in the single-processing execution model. T_{f1}, T_{p1}, T_{f3}, T_{f4}, $T_{p2a,b,}$ and T_{f5} are the execution times of FU_1, PPC_1, FU_3, FU_4, PPC_2, and FU_5, respectively. Since our design is based on the two-stage pipeline structure, the overall execution time (T_{cube}) is the same as T_{A}. Note that our algorithm is sequential, but the timing flow of the final architecture shows that the operations of functional units are parallel.

Figure 18 shows the throughput in terms of the number of effective bands and libraries, where one PPC or two are used. In the fastest case, the throughput is 2.1099 or 3.2196. However, when the number of bands and libraries is increased, the advantage of two PPCs disappears since the computational complexity of PPC_1 depends on the number

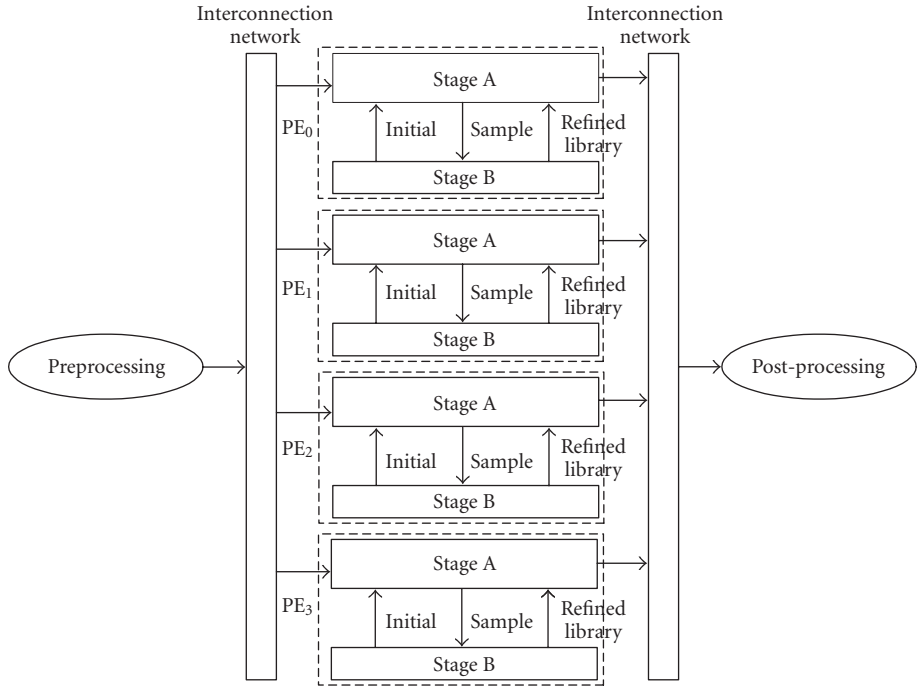


Figure 14: Block diagram in multiple data partitioning without separated library.

Table 1: Throughput comparison.

	Type	Resolution	Used bands	Second/cube
Estlick et al. [11]	Wildstar (Vertex 1000)	614 × 512	10	0.41
Moigne et al. [8]	SRC-6E (XC2V8000)	217 × 512	192	1.47
Du and Qi [10]	Pilchard (Vertex1000E)	614 × 512	50	500
Single PE (Proposed)	Xilinx Vertex-II pro	614 × 512	56	0.9
Multiple PE (Proposed)	Xilinx Vertex-II pro	614 × 512	56	0.14

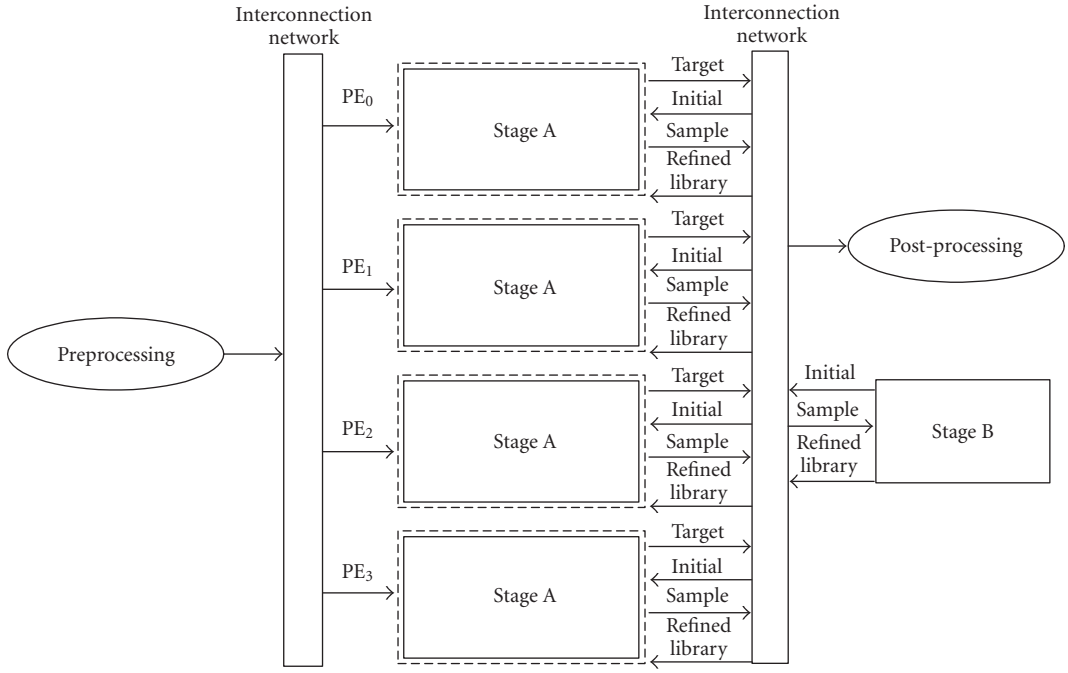


Figure 15: Block diagram in multiple-data partitioning with shared library.

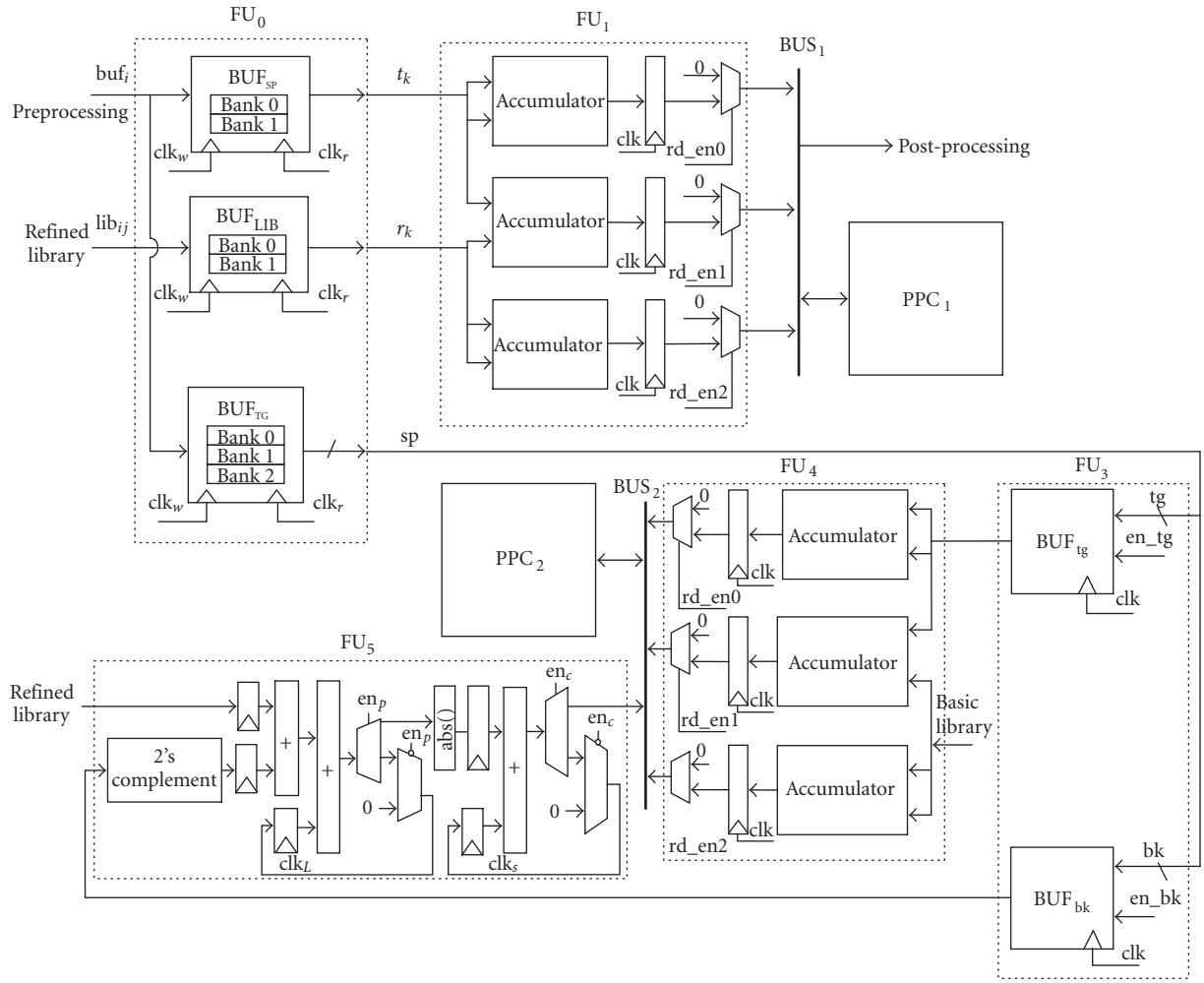


Figure 16: Illustration of a block diagram of a functional unit.

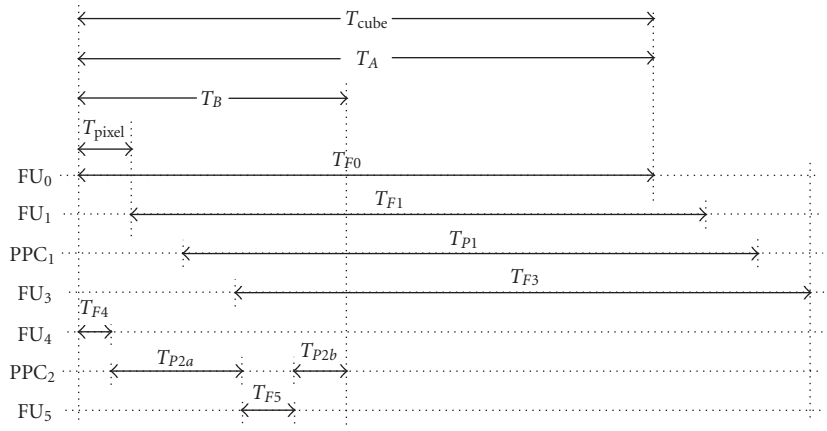
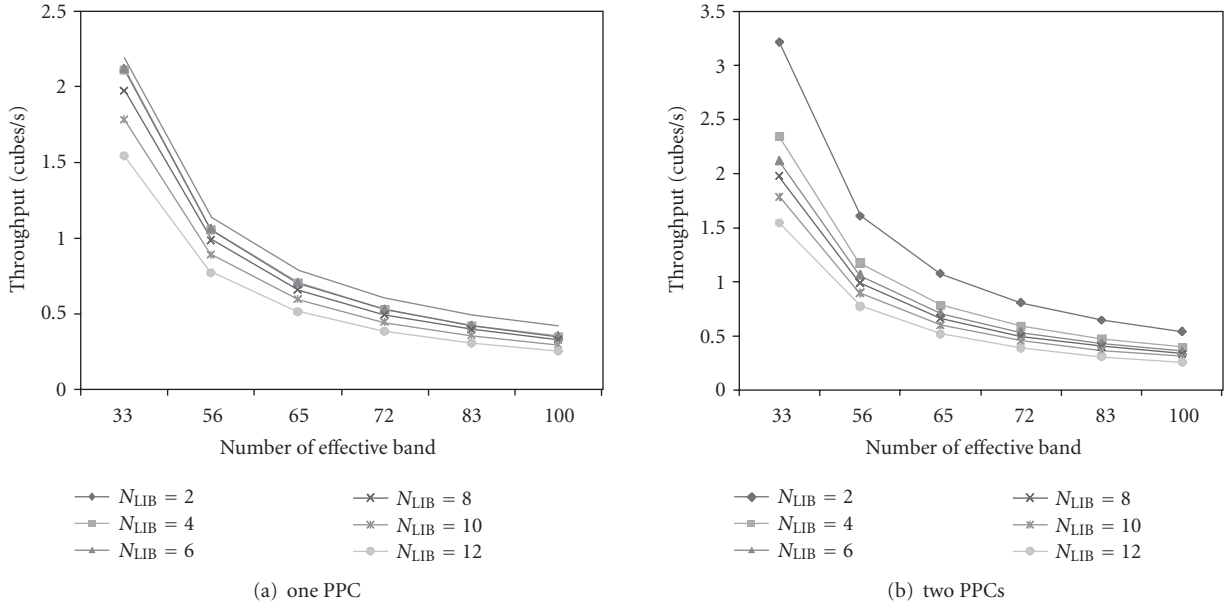


Figure 17: Illustration of timing flow in the single-processing execution model.

of effective bands and libraries, but the complexity of PPC_2 is invariant. Note that since our target architecture has the limited number of PPCs and the computation complexities of PPC_2 is much smaller than PPC_1, we use one PPC for a processing element.

4.2. Multiple-processing execution model

To enhance performance, the multiple-processing execution model is introduced in Figure 19. The incoming image from a bank is shared between two processing elements.



(a) one PPC (b) two PPCs

Figure 18: Illustration of throughput in terms of the number of effective bands and libraries in 614 × 512 resolution and 224 band hyperspectral images.

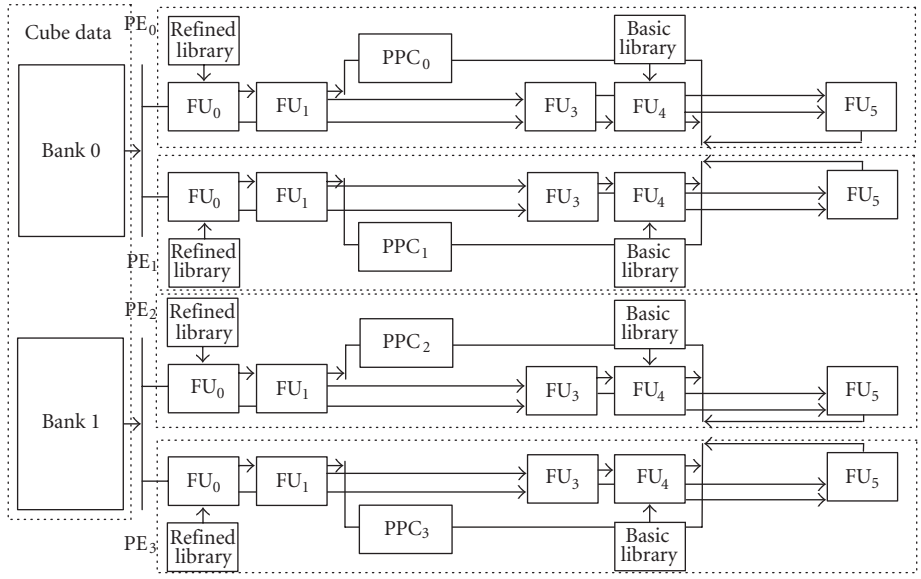


Figure 19: Illustration of the overall diagram.

Each processing element uses one PPC for floating point operations.

Figure 20 illustrates the throughput in the multiple-processing execution model.

4.3. Throughput comparison and discussion

To minimize the effect of sequential operation, two kinds of pipeline structures have been investigated. The functional units in Stage A can be parallel and the memory usage is to be optimized. We also implemented the floating point operation in the dedicated PPCs. To improve the execution time, the multiple-processing execution model, whose design is scalable, has been proposed.

Table 1 compares the throughput with other FPGA designs. To compare our design with other designs, we chose the case of 56 bands and 4 libraries in the single- and multiple-processing element. The throughput is 0.9 or 0.14 seconds per cube. Estlick used the Annapolis Microsystems Wildstar PCI board which has three Xilinx Virtex 1000 FPGAs and a 500 MHz Pentium III workstation. However, for processing, one FPGA was used as 50 MHz clock frequency for 614 × 512 images with 10 channels [11]. Wavelet-based dimension reduction was implemented

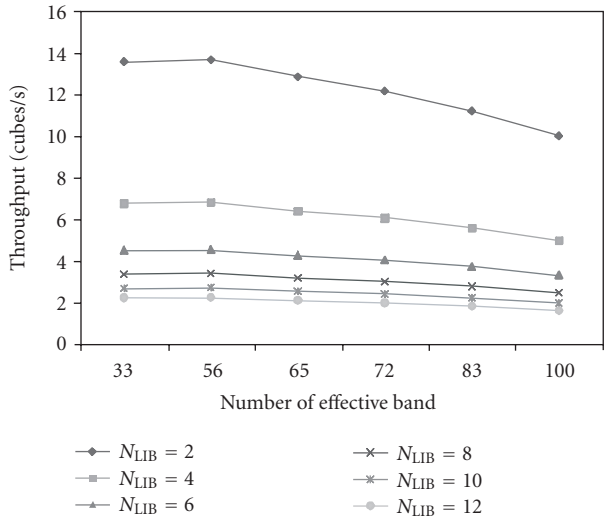


Figure 20: Illustration of throughput in the multiple-processing execution model.

in SRC-6E, where 217 × 512 resolution and a 192 band image was used [8]. The SRC-6E architecture has two 1 GHz Pentium microprocessors and two Xilinx Virtex II-6000-4 [9]. Du presented the parallel ICA algorithm in the Pilchard board which has Xilinx Virtex V1000E FPGA, where 614 × 512 resolution hyperspectral image and 50 selected bands were used, in which if the amount of weight vectors is four, the computational time requires 500 seconds [10].

5. CONCLUSION

A real-time target detection architecture for hyperspectral image processing is proposed in this paper. The architecture is based on a reduced complexity algorithm for high throughput applications. Multilevel pipelining of the architecture enhances the overall throughput, and the architecture is scalable so that the execution speed improves with the number of processing elements. The proposed pipelining optimizes overall memory usage and eliminates the memory bottleneck. The proposed architecture has been designed and implemented in FPGA to verify the relationship between the hardware complexity and the execution throughput of the reduced complexity hyperspectral image processing.

ACKNOWLEDGMENTS

This research is supported by Foundation of Ubiquitous computing and Networking (UCN) Project, the Ministry of Information and Communication (MIC) 21st Century Frontier R\&D Program in Korea.

REFERENCES

[1] N. Keshava, "Distance metrics and band selection in hyperspectral processing with applications to material identification and spectral libraries," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 42, no. 7, pp. 1552–1565, 2004.

[2] P. Bajcsy and P. Groves, "Methodology for hyperspectral band selection," *Photogrammetric Engineering and Remote Sensing*, vol. 70, no. 7, pp. 793–802, 2004.

[3] K.-S. Park, S. Hong, and P. Park, "Spectral contents characterization for efficient image detection algorithm design," in *Proceedings of the 6th IEEE International Conference on Computer and Information Technology (CIT '06)*, p. 137, Seoul, Korea, September 2006.

[4] W. E. Schaff, A. Copeland, M. Steffen, et al., "Real-time data processor for the COMPASS hyperspectral sensor system," in *Imaging Spectrometry IX*, vol. 5159 of *Proceedings of SPIE*, pp. 1–13, San Diego, Calif, USA, August 2003.

[5] X. Liu, W. L. Smith, D. K. Zhou, and A. Larar, "Principal component-based radiative transfer model for hyperspectral sensors: theoretical concept," *Applied Optics*, vol. 45, no. 1, pp. 201–209, 2006.

[6] M. R. Goupta and N. P. Jacobson, "Wavelet principal component analysis and its applocation to hyperspectral images," in *Proceedings of the IEEE International Conference on Image Processing (ICIP '06)*, pp. 1585–1588, Atlanta, Ga, USA, October 2006.

[7] S. Subramanian, N. Gat, A. Ratcliff, and M. Eismann, "Real-time hyperspectral data compression using principal component transform," in *Proceedings of the AVIRIS AirBorne Geosciences Workshop*, Pasadena, Calif, USA, February 2000.

[8] J. Le Moigne, P.-S. Yeh, J. Joiner, et al., "Dimension reduction of hyperspectral data on reconfigurable computers," in *Proceedings of the 4th Annual Earth Science Technology Conference (ESTC '04)*, Palo Alto, Calif, USA, June 2004.

[9] E. El-Araby, T. El-Ghazawi, J. Le Moigne, and K. Gaj, "Wavelet spectral dimension reduction of hyperspectral imagery on a reconfigurable computer," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT '04)*, pp. 399–402, Brisbane, Australia, December 2004.

[10] H. Du and H. Qi, "A reconfigurable FPGA system for parallel independent component analysis," *EURASIP Journal on Embedded Systems*, vol. 2006, Article ID 23025, 12 pages, 2006.

[11] M. Estlick, M. Leeser, J. Theiler, and J. J. Szymanski, "Algorithmic transformations in the implementation of K-means clustering on reconfigurable hardware," in *Proceedings of the 9th ACM/SIGDA International Sysmposium on Field Programmable Gate Arrays (FPGA '01)*, pp. 103–110, Monterrey, Calif, USA, February 2001.

[12] M. Leeser, P. Belanovic, M. Estlick, M. Gokhale, J. J. Szymanski, and J. Theiler, "Applying reconfigurable hardware to the analysis of multispectral and hyperspectral imagery," in *Imaging Spectrometry VII*, vol. 4480 of *Proceedings of SPIE*, pp. 100–107, San Diego, Calif, USA, August 2001.

[13] G. A. Shaw and H.-H. K. Burke, "Spectral imaging for remote sensing," *Lincoln Laboratory Journal*, vol. 14, no. 1, pp. 3–28, 2003.

[14] M. L. Nischan, R. M. Joseph, J. C. Libby, and J. P. Kerekes, "Active spectral imaging," *Lincoln Laboratory Journal*, vol. 14, no. 1, pp. 131–144, 2003.

[15] M. K. Griffin and H.-H. K. Burke, "Compensation of hyperspectral data for atmospheric effect," *Lincoln Laboratory Journal*, vol. 14, no. 1, pp. 29–54, 2003.

[16] T. W. Fry and S. Hauck, "Hyperspectral image compression on reconfigurable platforms," in *Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '00)*, pp. 251–260, Napa, Calif, USA, April 2000.

[17] B.-C. Gao, M. J. Montes, Z. Ahmad, and C. O. Davis, "Atmospheric correction algorithm for hyperspectral remote sensing of ocean color from space," *Applied Optics*, vol. 39, no. 6, pp. 887–896, 2000.

[18] G. Girouard, A. Bannari, A. Harti, and A. Desrochers, "Validated spectral angle mapper algorithm for geological mapping: comparative study between quickbird and landsat-tm," in *Proceedings of the 20th International Society for Photogrammetry and Remote Sensing Congress (ISPRS '04)*, pp. 599–605, Istanbul, Turkey, July 2004.

[19] L. M. Bruce and J. Li, "Wavelets for computationally efficient hyperspectral derivative analysis," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 39, no. 7, pp. 1540–1546, 2001.

[20] C.-I. Chang, "An information-theoretic approach to spectral variability, similarity, and discrimination for hyperspectral image analysis," *IEEE Transactions on Information Theory*, vol. 46, no. 10, pp. 1927–1932, 2000.