Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2010, Article ID 454506, 11 pages
doi:10.1155/2010/454506

Research Article

Parameterized Hardware Design on Reconfigurable Computers: An Image Processing Case Study

Miaoqing Huang,[1] Olivier Serres,[2] Tarek El-Ghazawi,[2] and Gregory Newby[3]

[1] *Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR 72701, USA*
[2] *Department of Electrical and Computer Engineering, The George Washington University, Washington, DC 20052, USA*
[3] *Arctic Region Supercomputing Center, University of Alaska Fairbanks, Fairbanks, AK 99775, USA*

Correspondence should be addressed to Miaoqing Huang, mqhuang@uark.edu

Received 1 July 2009; Accepted 10 February 2010

Academic Editor: Elías Todorovich

Reconfigurable Computers (RCs) with hardware (FPGA) co-processors can achieve significant performance improvement compared with traditional microprocessor (μP)-based computers for many scientific applications. The potential amount of speedup depends on the intrinsic parallelism of the target application as well as the characteristics of the target platform. In this work, we use image processing applications as a case study to demonstrate how hardware designs are parameterized by the co-processor architecture, particularly the data I/O, i.e., the local memory of the FPGA device and the interconnect between the FPGA and the μP. The local memory has to be used by applications that access data randomly. A typical case belonging to this category is image registration. On the other hand, an application such as edge detection can directly read data through the interconnect in a sequential fashion. Two different algorithms of image registration, the exhaustive search algorithm and the Discrete Wavelet Transform (DWT)-based search algorithm, are implemented on hardware, i.e., Xilinx Vertex-IIPro 50 on the Cray XD1 reconfigurable computer. The performance improvements of hardware implementations are 10× and 2×, respectively. Regarding the category of applications that directly access the interconnect, the hardware implementation of Canny edge detection can achieve 544× speedup.

1. Introduction

Reconfigurable Computers (RCs) are traditional computers extended with co-processors based on reconfigurable hardware like FPGAs. Representative RC systems include SGI RC100 [1], SRC-6 [2], and Cray XD1 [3]. These enhanced systems are capable of providing significant performance improvement for scientific and engineering applications [4]. The performance of a hardware design on an FPGA device depends on both the intrinsic parallelism of the design as well as the characteristics of the FPGA co-processor architecture, which consists of the FPGA device itself and the surrounding data interface. Due to the limited size of the internal Block RAM memory, it is not applicable to store large amounts of data inside the FPGA device. Therefore external, however, local SRAM modules are generally connected to the hardware co-processor for data storage, such as the example shown in Figure 1. Furthermore, an FPGA co-processor can directly access the host memory through the interconnect, which generally provides a sustained bandwidth up to several GB/s. The available number and data width of local memory banks and the interconnect channels play important roles in the hardware implementation on an FPGA device and decide the parallelism a design can achieve in many cases. In this work, image processing algorithms are adopted as a case study to demonstrate how hardware designs can be parameterized by the data I/O of the FPGA co-processor in order to achieve the best performance.

Image processing applications are capable of gaining a significant performance improvement with hardware design [5, 6]. Two categories of image processing applications are selected to represent hardware designs that present different data I/O requirements. The first category of applications is image registration, which requires the use of local memory

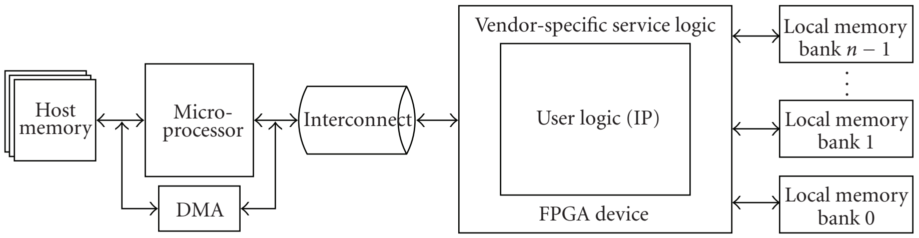

Figure 1: General architecture of a reconfigurable computer.

for data access. The second category of applications is edge detection, which directly reads from or writes to the interconnect in a streaming fashion. Image registration is a very important image processing task. It is used to align or match pictures taken in different conditions (at a different time, angle, or from different sensors). A vast majority of automatic image processing systems require the use of image registration processes. Common image registration applications include target recognition (identification of a small target inside a much bigger image), satellite image alignment in order to detect changes such as land usage or forest fires [7], matching stereo images to recover depth information, and superposing medical images taken at different moments for diagnosis [8, 9]. As an example of the applications in the second category, edge detectors encompass image processing algorithms that identify the positions of edges in an image. Edges are discontinuities in terms of intensity or orientation or both and generally represent meaningful characteristics of the image (boundaries of objects, e.g.). Commonly, edge detectors are used to filter relevant information in an image. Thus, they greatly reduce the amount of processing needed for interpreting the information contents of an image. One of the most important edge detection algorithms is the Canny edge detection [10]. The Canny edge detection operator was developed by Canny in 1986 and uses a multistage algorithm to detect a wide range of edges in images. It remains until now, as a state-of-the-art edge detector used in many applications.

The implementation of image registration and edge detection on reconfigurable computers has been previously reported in [11, 12], respectively. In this paper, we not only present the detail of hardware design itself, but also exploit the role of data I/O of the FPGA co-processor in the design process. More precisely, we demonstrate how the design of image processing applications is parameterized by local memory architecture and DMA interface on reconfigurable computers. Furthermore, the hardware processing time of both applications are formalized in terms of clock cycles. The remaining text is organized as follows. In Section 2, we discuss the related work based on literature survey. In Section 3, two related image registration algorithms, the exhaustive search algorithm and the DWT-based search algorithm, and their hardware implementations are discussed. Section 4 focuses on the design of Canny edge detection in hardware. The implementation of all applications on the Cray XD1 reconfigurable computer is presented in Section 5. Finally, Section 6 concludes this work.

2. Related Work

Since image registration is a computation-demanding process in general, hardware (e.g., FPGA device) is leveraged to improve the processing performance. In [8], Dandekar and Shekhar introduced an FPGA-based architecture to accelerate mutual information-(MI)-based deformable registration during computed tomography-(CT)-guided interventions. Their reported implementation was able to reduce the execution time of MI-based deformable registration from hours to a few minutes. Puranik and Gharpure presented a multilayer feedforward neural network (MFNN) implementation in Xilinx XL4085 for template search in standard sequential scan and detect (SSDA) image registration [13]. In [14], Liu et al. proposed a PC-FPGA geological image processing system in which the FPGA was used to implement Fast Fourier Transform-(FFT)-based automatic image registration. In [15], El-Araby et al. prototyped an automatic image registration methodology for remote sensing using a reconfigurable computer. However, these previous work only emphasized the image registration algorithm itself. The FPGA data I/O as a factor in the design was not discussed in detail.

Low-level image processing operators, such as digital filters, edge detectors and digital transforms are good candidates for hardware implementation. In [16], a generic architectural model for implementing image processing algorithms of real-time applications was proposed and evaluated. In [17], a Canny edge detection application written in Handel-C and implemented in the FPGA device was discussed. The proposed architecture is capable of producing one edge-pixel every clock cycle. The work in [18] illustrated how to use design patterns in the mapping process to overcome image processing constraints on FPGAs. However, most of the previous works, for example, [16–18], only focused on the algorithms alone and did not consider the platform characteristics as a factor in the design.

3. Image Registration

In this section, we discuss how to implement image registration algorithms in hardware to exploit the processing parallelism, which is bounded by the local memory architecture.

3.1. Background. Image registration can be defined as a mapping between two images, the reference image R and the test image T, both spatially and with respect to the intensity

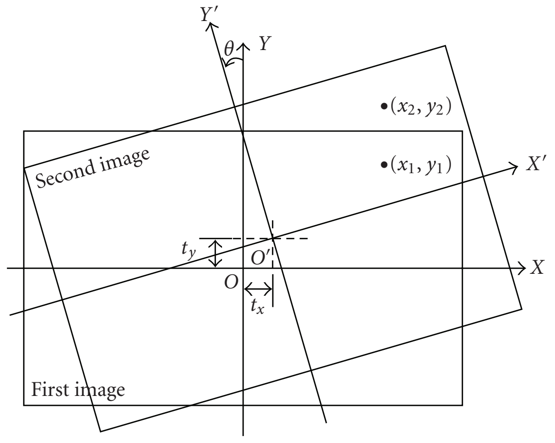

FIGURE 2: Rigid-body transformation (scale = 1).

[19]. If these images are defined as two 2D arrays of a given size denoted by I_1 and I_2 where $I_1(x, y)$ and $I_2(x, y)$ each map to their respective intensity values, then the mapping between images can be expressed as

$$I_2(x, y) = g(I_1(f(x, y))), \tag{1}$$

where f is a 2D spatial-coordinate transformation and g is a 1D intensity or radiometric transformation. More precisely, f is a transformation that maps two spatial coordinates, x and y, to new spatial coordinates x' and y':

$$(x', y') = f(x, y). \tag{2}$$

g is used to compensate gray value differences caused by different illuminations or sensor conditions.

According to [19], image registration can be viewed as the combination of four components:

(1) *a feature space*, that is, the set of characteristics used to perform the matching and which are extracted from the reference and test images;

(2) *a search space*, that is, the class of potential transformations that establish the correspondence between the reference and test images;

(3) *a search strategy*, which is used to decide how to choose the next transformation from the search space;

(4) *a similarity metric*, which evaluates the match between the reference image and the transformed test image for a given transformation chosen in the search space.

The fundamental characteristic of any image registration technique is the type of spatial transformation or mapping used to properly overlay two images. The most common transformations are rigid-body, affine, projective, perspective, and global polynomial. Rigid-body transformation is composed of a combination of a rotation (θ), a translation (t_x, t_y), and a scale change (s). An example is shown in Figure 2. It typically has four parameters, t_x, t_y, s, θ, which map a point (x_1, y_1) of the first image to a point (x_2, y_2) of the second image as follows:

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} t_x \\ t_y \end{pmatrix} + s\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \tag{3}$$
$$\text{or} \quad \overline{p}_2 = \overline{t} + s\mathcal{R}\overline{p}_1,$$

where $\overline{p}_1$ and $\overline{p}_2$ are the coordinate vectors of the two images; $\overline{t}$ is the translation vector; s is a scalar scale factor; $\mathcal{R}$ is the rotation matrix. Since the rotation matrix $\mathcal{R}$ is orthogonal, the angles and lengths in the original image are preserved after the registration. Because of the scalar scale factor s, rigid-body transformation allows changes in length relative to the original image, but they are the same in both x and y axes. (Please note both (x_1, y_1) and (x_2, y_2) are coordinates in the same Cartesian coordinate system with the origin O.)

Computing the correlation coefficient is the basic statistical approach to registration and is often used for template matching or pattern recognition. A correlation coefficient is a similarity measure or match metric, that is, it gives a measure of the degree of similarity between a template (the reference image) and an image (the transformed test image). The correlation coefficient between the reference image R and the image T', which is the test image after rigid-body transformation, is given as

$$\frac{\sum_{x,y}(R(x,y)-\mu_R)(T'(x,y)-\mu_{T'})}{\sqrt{\sum_{x,y}(R(x,y)-\mu_R)^2\sum_{x,y}(T'(x,y)-\mu_{T'})^2}}, \tag{4}$$

where μ_R and $\mu_{T'}$ are mean of the image R and T'. If the image R matches T', the correlation coefficient will have its peak with the corresponding transformation. Therefore, by computing correlation coefficients over all possible transformations, it is possible to find the transformation that yields the peak value of the correlation coefficient.

In this work, rigid-body transformation is selected for the registration between two images and the correlation coefficient is used to measure the similarity. Further, we assume that both the reference image and the test image are 8-bit grayscale and share the same size.

Given a search space, $(\Delta\Theta, \Delta X, \Delta Y)$ theoretically all tuples of (θ, t_x, t_y) are to be tested to find the tuple that generates the maximum correlation coefficient between the reference image and the transformed test image (In this work, the scale factor s is fixed at 1.). Figure 3 shows the two steps to test each tuple. The first step is to apply a rigid-body transformation on the test image T to get T'. The second step is to calculate the correlation coefficient between T' and the reference image R. As shown in Figure 3(b), only the pixels of both images within the shaded region are used during the calculation. Since the test image T is rotated and translated to obtain T', some parts of T' are beyond the shaded region. In other words, some portions of the shaded region (shown as crossed regions in the left Cartesian coordinate system of Figure 3(b)) do not belong to T'. For the pixels belonging to these crossed regions, their values are treated as zeros in the calculation.

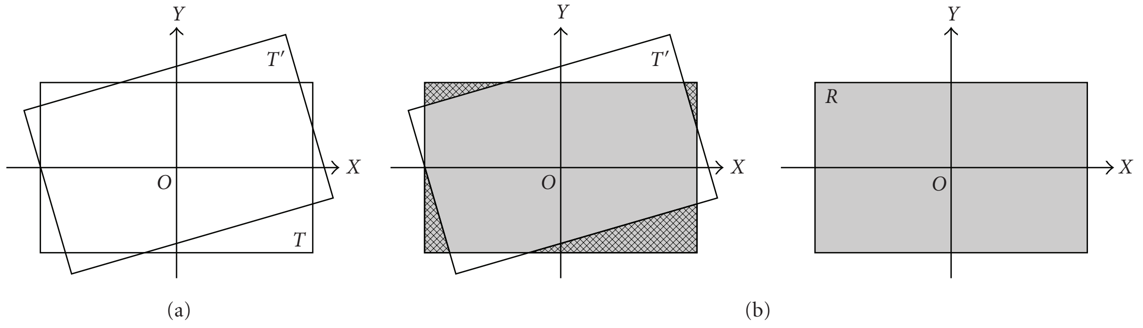

FIGURE 3: Two steps in image registration: (a) rigid-body transformation on the test image T, (b) calculate correlation coefficient between the transformed test image T' and the reference image R.

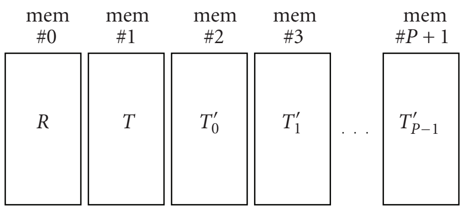

FIGURE 4: Local memory data storage layout in the exhaustive search algorithm for image registration.

In the remaining part of this section, two different approaches based on rigid-body transformation are discussed in details. The first approach literally tests the whole search space to find the best tuple of (θ, t_x, t_y). The second approach applies DWT on both the reference image and the test image to reduce the search resolutions in order to improve the search efficiency.

3.2. Exhaustive Search Algorithm. As its name implies, the exhaustive search algorithm tests all possible tuples of (θ, t_x, t_y) with a fixed search resolution, δ_θ, δ_x, and δ_y, on each dimension, respectively, in order to find the tuple that produces the highest correlation coefficient between the transformed test image and the reference image. If this algorithm is implemented on a scalar microprocessor, these tuples have to be tested in sequence. However, if the same algorithm is implemented in hardware, multiple tuples can be tested in parallel to improve the performance.

Since the size of an image is normally bigger than the amount of available Block RAM inside an FPGA device, the local external memory is used to store images. Assuming there are $P + 2$ individual local memory banks connected directly to the FPGA device, and one bank keeps the reference image R, one bank keeps the test image T. The other remaining P banks are used to store P transformed test images $T's$ using different tuples of (θ, t_x, t_y), as shown in Figure 4. If we further assume that each memory bank has its own independent read and write ports, P transformations of the test image can be carried out concurrently. The calculation of correlation coefficients between the image R and P different $T's$ can be performed in parallel as well.

Given the coordinate of one pixel in the original image, (3) is used to calculate the coordinate of the corresponding pixel in the transformed image. Then, the intensity of the pixel (x_1, y_1) in T can be written into T' at the coordinate of (x_2, y_2). If we assume that there are S pixels in the original image and the hardware implementation is fully pipelined, the transformation step would take approximately S clock cycles. Furthermore, some extra clock cycles are needed to initialize the intensities of all pixels within the shaded region to zero due to two reasons. First, there are several regions within which the pixels do not belong to T', as shown in Figure 3(b). Second, there may exist artifacts whose coordinates are within both the shaded region and T', but are not calculated due to discretization, as shown in Figure 5. If the intensities of these pixels are left randomly, it may affect the accuracy of the correlation coefficient. Therefore, it is necessary to initialize the intensities of all pixels in the shaded region to zero in the first place. Since the data width of the memory bank's access ports is multiple-byte, multiple pixels can be initialized in one clock cycle. If we assume that the data width is D-byte, then the initialization process would take roughly S/D clock cycles. Overall, the transformation step of P tuples would take $((D + 1)/D)S$ clock cycles. The mean intensity $\mu_{T'}$ of each T' can be calculated during the transformation step, hence it takes no extra time. The mean intensity μ_R can be precalculated by the microprocessor and forwarded to the FPGA device later since it remains unchanged during the whole image registration process.

Although the calculation of the correlation coefficient as (4) between R and $T's$ is more complicated than the transformation step, it takes the same time as the initialization process since D pixels can be read and processed in the same clock cycle. Altogether, these three steps, including initialization, transformation and correlation coefficient calculation, would take $((D + 2)/D)S$ clock cycles for testing P tuples of (θ, t_x, t_y). If the entire search space consists of $\Delta\Theta * \Delta X * \Delta Y$ tuples, the whole registration process would take

$$\frac{\Delta\Theta * \Delta X * \Delta Y * (D+2)}{PD} S \quad (5)$$

clock cycles. Apparently, the image registration time in hardware can be significantly reduced by increasing the number of local memory banks. Widening the data width of

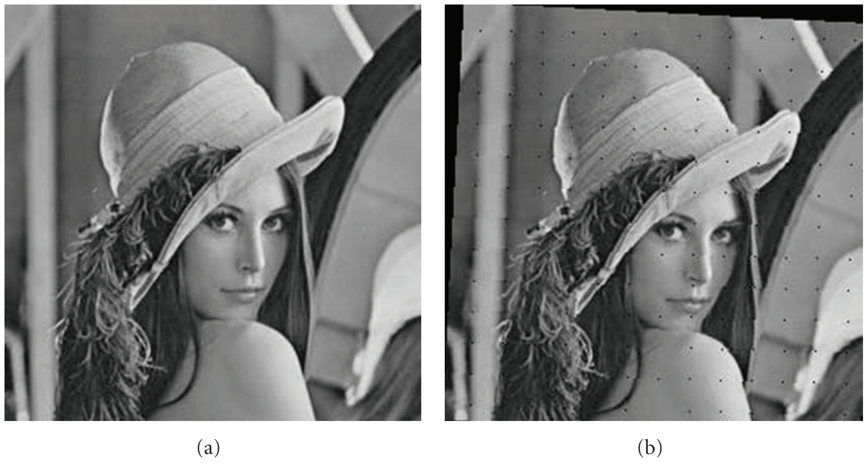

Figure 5: Artifacts due to discretization in rigid-body transformation ((a) the original image; (b) the transformed image).

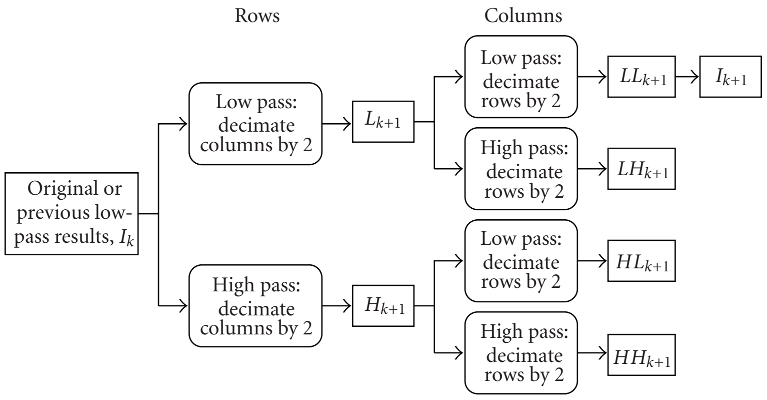
Rows
Columns
Original or previous low-pass results, I_k
Low pass: decimate columns by 2
High pass: decimate columns by 2
L_{k+1}
H_{k+1}
Low pass: decimate rows by 2
High pass: decimate rows by 2
Low pass: decimate rows by 2
High pass: decimate rows by 2
LL_{k+1}
I_{k+1}
LH_{k+1}
HL_{k+1}
HH_{k+1}

Figure 6: DWT decomposition of an image.

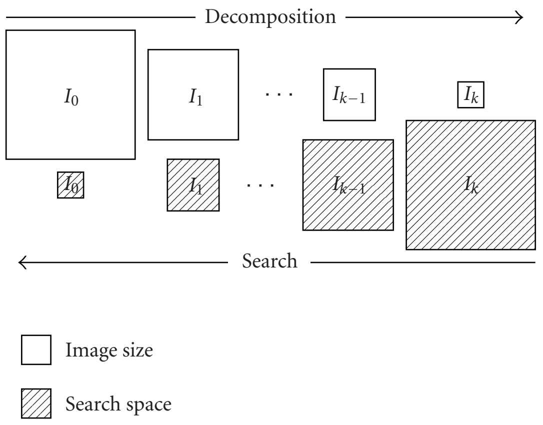

Figure 7: Decrease the search space and increase the search resolution in the search process.

access port of local memory can improve the performance as well; however, it can also hit the upper bound very quickly due to the fact that

$$\lim_{D \to \infty} \frac{D+2}{D} = 1. \tag{6}$$

3.3. DWT-Based Search Algorithm. Although the exhaustive search algorithm is quite straightforward, it is computation-demanding as well. In [20], a DWT-based image registration approach was proposed. As shown in Figure 6, both the test image and the reference image go through several levels of Discrete Wavelet Transform before applying image registration. After each level of DWT, the image size is shrunk to 1/4 of the previous level. In the meantime the image resolution is reduced to half. For example, if k levels of DWT are applied on both the test image T_0 and the reference image R_0, two series of images, $T_0, T_1, \ldots, T_k$, and $R_0, R_1, \ldots, R_k$, are obtained. The registration process starts from the exhaustive search between T_k and R_k among the search space of $(\Delta\Theta, \Delta X, \Delta Y)$ with the search resolution, $2^k * \delta_\theta$, $2^k * \delta_x$, and $2^k * \delta_y$, on each dimension. The registration result between T_k and R_k, $(\theta_k, t_{x_k}, t_{y_k})$ becomes the center of the search space of the registration between T_{k-1} and R_{k-1}. In other words, the registration between T_{k-1} and R_{k-1} is among the search space of $(\theta_k \pm 2^k * \delta_\theta,\ t_{x_k} \pm 2^k * \delta_x,\ t_{y_k} \pm 2^k * \delta_y)$. However, the search resolution is increased to $2^{k-1} * \delta_\theta$, $2^{k-1} * \delta_x$, and $2^{k-1} * \delta_y$, on each dimension. In general, when the registration process traces back one level, the search scope is reduced to half on

Table 1: Search strategy summary for rotation.

Decomposition Level	Search Space	Search Resolution	Result
k	$\Delta\Theta$	$2^k * \delta_\theta$	θ_k
$k-1$	$[\theta_k - 2^k * \delta_\theta; \theta_k + 2^k * \delta_\theta]$	$2^{k-1} * \delta_\theta$	θ_{k-1}
$k-2$	$[\theta_{k-1} - 2^{k-1} * \delta_\theta; \theta_{k-1} + 2^{k-1} * \delta_\theta]$	$2^{k-2} * \delta_\theta$	θ_{k-2}
...	...	...	...
2	$[\theta_3 - 2^3 * \delta_\theta; \theta_3 + 2^3 * \delta_\theta]$	$2^2 * \delta_\theta$	θ_2
1	$[\theta_2 - 2^2 * \delta_\theta; \theta_2 + 2^2 * \delta_\theta]$	$2 * \delta_\theta$	θ_1
0	$[\theta_1 - 2 * \delta_\theta; \theta_1 + 2 * \delta_\theta]$	δ_θ	θ_0^*

* The final output.

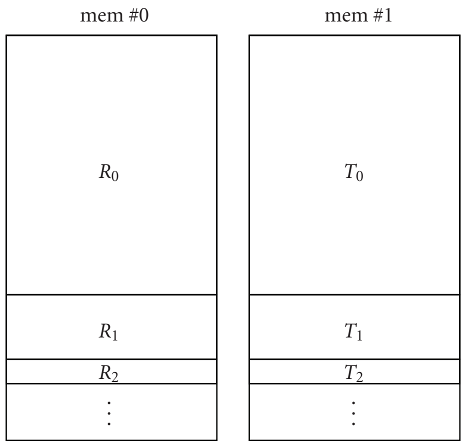

Figure 8: Store the original and decomposed images in the same memory bank in the DWT-based image registration.

each dimension, and the search resolution is increased two times on each dimension, respectively. This search strategy is illustrated in Figure 7. Table 1 details the search space and search resolution at each step in which rotation is taken as an example.

Different DWT decomposition processes have to be carried out in a sequence. Similarly the search processes at different levels need to be performed one after the other. Due to these two reasons, the original image and the decomposed images can be stored in the same memory bank, as shown in Figure 8 in which R_k or T_k denote the decomposed image at the level k.

If we use the same assumptions as in Section 3.2, that is, both original images consist of S pixels, the data width of the local memory is D-byte, and there are $P+2$ independent local memory banks, then the DWT decomposition step alone will take

$$\frac{S}{D}\left(1+\frac{1}{4}+\cdots+\left(\frac{1}{4}\right)^{k-1}\right) = \frac{4S}{3D}\left(1-\left(\frac{1}{4}\right)^{k}\right) \quad (7)$$

clock cycles.

The search between the decomposed reference image and the test image at each level can use the same method described in Section 3.2. By observing Table 1, it is found that the search space at each level, except the level k, is 125 tuples of (θ,t_x,t_y). Therefore, the search from level 0 to level $k-1$ will take

$$\begin{aligned}\frac{125S(D+2)}{PD}&\left(1+\frac{1}{4}+\cdots+\left(\frac{1}{4}\right)^{k-1}\right)\\ &= \frac{500S(D+2)}{3PD}\left(1-\left(\frac{1}{4}\right)^{k}\right)\end{aligned} \quad (8)$$

clock cycles. The search at the level k itself takes

$$\frac{\Delta\Theta * \Delta X * \Delta Y * (D+2)}{32^k \delta_\theta \delta_x \delta_y PD} S \quad (9)$$

clock cycles.

4. Edge Detection

Edge detection aims at identifying pixels in a digital image at which the image brightness changes sharply, that is, having discontinuities. Most edge detection algorithms involve the convolution process between the image and a kernel. Convolution provides a way of "multiplying together" two arrays of numbers, generally of different sizes, but of the same dimensionality, to produce a third array of numbers of the same dimensionality. This can be used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values. In an image processing context, one of the input arrays is normally just a grayscale image. The second array is usually much smaller, and is also two dimensional (although it may be just a single coefficient). The second array is always known as the *kernel*, as shown in Figure 9. If the image has M rows and N columns, and the kernel has m rows and n columns, then the size of the output image will consist of $M-m+1$ rows and $N-n+1$ columns. Mathematically we can write the convolution between the image I and the kernel K as

$$O(x_1,y_1) = \sum_{x_2=0}^{m-1}\sum_{y_2=0}^{n-1} I(x_1+x_2, y_1+y_2) \times K(x_2,y_2), \quad (10)$$

where x_1 runs from 0 to $M-m$ and y_1 runs from 0 to $N-n$.

From Figure 9 and (10), we can find that the calculation of different pixels in the output image is independent to each other. Therefore, the intensities of multiple output pixels can be computed in parallel in a hardware design. Since the input

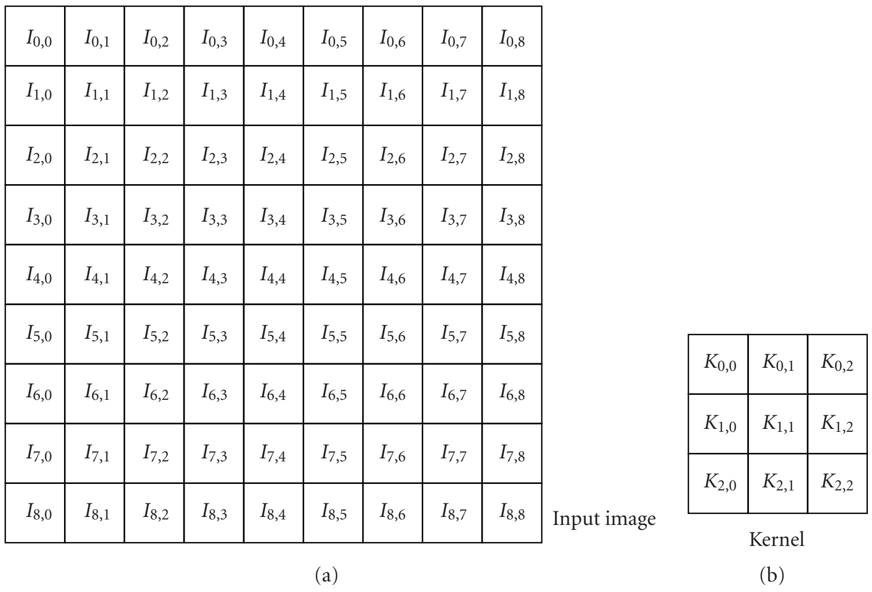
$I_{0,0}$ $I_{0,1}$ $I_{0,2}$ $I_{0,3}$ $I_{0,4}$ $I_{0,5}$ $I_{0,6}$ $I_{0,7}$ $I_{0,8}$
$I_{1,0}$ $I_{1,1}$ $I_{1,2}$ $I_{1,3}$ $I_{1,4}$ $I_{1,5}$ $I_{1,6}$ $I_{1,7}$ $I_{1,8}$
$I_{2,0}$ $I_{2,1}$ $I_{2,2}$ $I_{2,3}$ $I_{2,4}$ $I_{2,5}$ $I_{2,6}$ $I_{2,7}$ $I_{2,8}$
$I_{3,0}$ $I_{3,1}$ $I_{3,2}$ $I_{3,3}$ $I_{3,4}$ $I_{3,5}$ $I_{3,6}$ $I_{3,7}$ $I_{3,8}$
$I_{4,0}$ $I_{4,1}$ $I_{4,2}$ $I_{4,3}$ $I_{4,4}$ $I_{4,5}$ $I_{4,6}$ $I_{4,7}$ $I_{4,8}$
$I_{5,0}$ $I_{5,1}$ $I_{5,2}$ $I_{5,3}$ $I_{5,4}$ $I_{5,5}$ $I_{5,6}$ $I_{5,7}$ $I_{5,8}$
$I_{6,0}$ $I_{6,1}$ $I_{6,2}$ $I_{6,3}$ $I_{6,4}$ $I_{6,5}$ $I_{6,6}$ $I_{6,7}$ $I_{6,8}$
$I_{7,0}$ $I_{7,1}$ $I_{7,2}$ $I_{7,3}$ $I_{7,4}$ $I_{7,5}$ $I_{7,6}$ $I_{7,7}$ $I_{7,8}$
$I_{8,0}$ $I_{8,1}$ $I_{8,2}$ $I_{8,3}$ $I_{8,4}$ $I_{8,5}$ $I_{8,6}$ $I_{8,7}$ $I_{8,8}$
Input image
$K_{0,0}$ $K_{0,1}$ $K_{0,2}$
$K_{1,0}$ $K_{1,1}$ $K_{1,2}$
$K_{2,0}$ $K_{2,1}$ $K_{2,2}$
Kernel
(a)
(b)

Figure 9: An example small image (a) and kernel (b) to illustrate convolution.

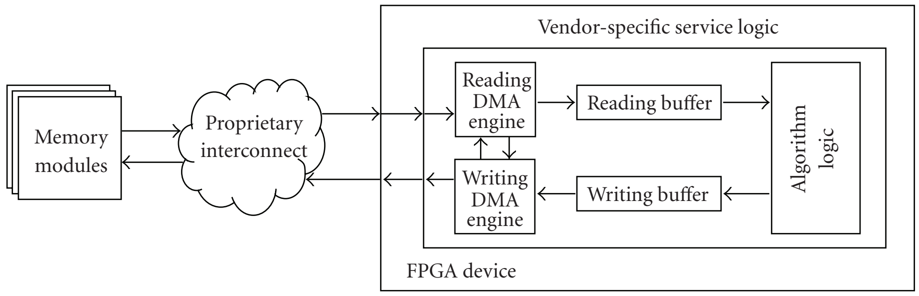

Figure 10: Streaming data transfer mode on reconfigurable computers.

image reading and the output image writing are both in a sequential mode, the data storage in local memory can be avoided. Instead, the user logic can access the interconnect directly for fetching the source image and storing the result image. In order to optimize the hardware performance, the interconnect and the user logic are chained into a pipeline, and the data run through the pipeline as a stream. We call this architecture Streaming Data Transfer Mode, shown in Figure 10. Two DMA engines work in parallel to retrieve raw data blocks from and return result data blocks to the main memory. Under ideal circumstances, the reading DMA engine receives one raw data block from the input channel and the writing DMA engine puts one result data block to the output channel every clock cycle.

Being one stage of the overall pipeline, the design of the algorithm logic is parameterized by the characteristics of other components in the pipeline, that is, the data width of the interconnect between the FPGA device and the μP. Because the data width of the interconnect fabric is multiple-byte wide in general, several pixels are fed into the algorithm logic in the same clock cycle. To maximize the throughput of the overall architecture, the algorithm logic has to be capable of performing the operations of multiple pixels concurrently and taking new data input every clock cycle. In the following discussion, we assume that (1) the image is in 8-bit grayscale, (2) the image size is $M \times N$ and the kernel size is $m \times n$, and (3) the data width of the interconnect is D-byte. Furthermore, pixels in the original image are delivered into the algorithm logic in a stream, starting with the pixel at the top-left corner, ending with the pixel at the bottom-right corner.

The diagram of the algorithm logic is shown in Figure 11. The architecture consists of four components, one Line Buffer, one Data Window, an array of PEs, and the Data Concatenating Block.

The quantity of PEs is D, that is, the data width of the interconnect in byte. Every PE is fully pipelined and is capable of taking a new input, that is, one block of $m \times n$ pixels, every clock cycle. The output of one PE is the intensity of one pixel in the result image.

The Data Window is a two-dimension register array of $m \times (n + D - 1)$ in charge of providing image blocks to the downstream PEs. Analogously, the Data Window scans the original image from left to right and from top to bottom

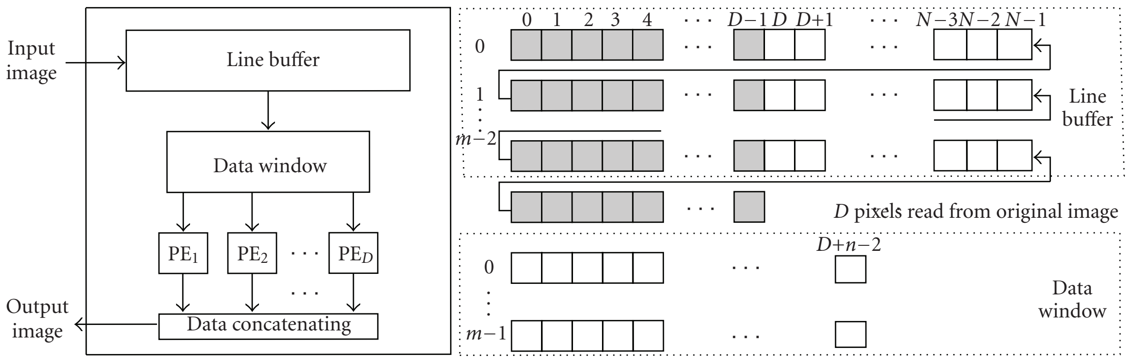

Figure 11: Diagram of the algorithm logic in edge detection.

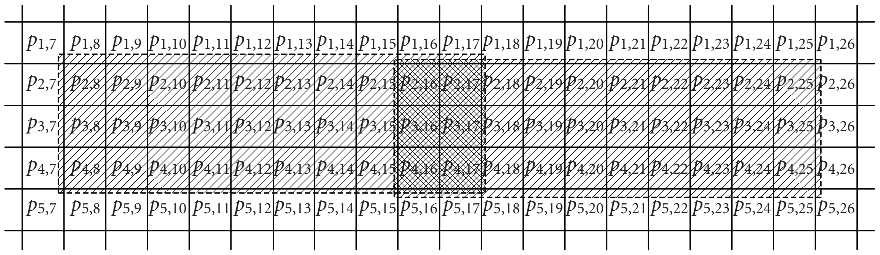

Figure 12: The content of Data Window in two consecutive steps (assume the kernel size is 3×3 and D is 8).

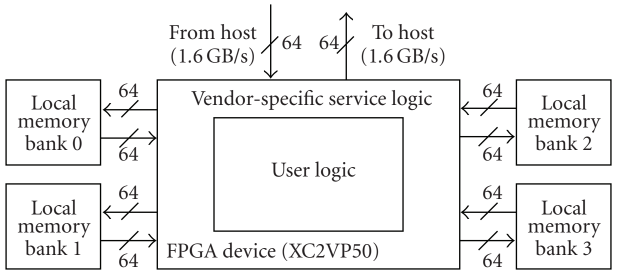

Figure 13: Cray XD1 FPGA local memory architecture.

with a horizontal stride of D pixels and a vertical stride of 1 pixel. Figure 12 demonstrates the scanning from left to right and shows the contents in two consecutive steps of the data window. Once the data window receives a valid input, that is, one block of $m \times D$ pixels, it does a D-byte left shift for all rows with the input tailing the right most $n - 1$ columns.

The Line Buffer is a two-dimension register array of $(m-1) \times N$. The original image is transferred into the FPGA as a stream. However, PEs request image blocks that spread different rows. The purpose of the line buffer is to keep all pixels in registers until one $m \times D$ block forms. One $m \times D$ block, shown in gray in Figure 11, comprises the new arrived D pixels and another $(m-1) \times D$ pixels that reside at the head of every row of the line buffer. Every time the line buffer receives D pixels of the original image, it performs two actions simultaneously. The first action is to deliver the new formed image block to the data window. The second action is to do a D-byte left shift in a zigzag form, in which two neighbor rows are linked together by connecting their tail and head.

The design of the Data Concatenating Block is straightforward because what it does is to concatenate the outputs of the upstream PEs together. Once it receives valid output from the PEs, that is, D consecutive pixels in the output image, it sends them into the output channel.

In general, the four components in Figure 11 form a pipeline chain and each of them is fully pipelined as well. This architecture is able to accept D pixels of the original image every clock cycle and output D pixels of the result image every clock cycle at the same time. In case of a multiple-stage algorithm, such as the four-stage Canny edge detector, various stages can be chained together and each stage consists of these components with different parameters and functionalities. Under ideal scenario, it would take $(M \times N)/D$ clock cycles to perform an edge detection operation on an input image. However, the real performance is upper-bounded by the sustained bandwidth of the interconnect in general.

5. Implementation and Results

These two image registration algorithms and a Canny edge detection algorithm have been implemented on the Cray XD1 reconfigurable computer with Xilinx XC2VP50 FPGA devices. On the Cray XD1 platform, each FPGA device is connected to four local SRAM modules, 4 MB each, as shown in Figure 13. Every local memory module has separate reading and writing ports connected to the FPGA device, and is able to accept reading or writing transactions every

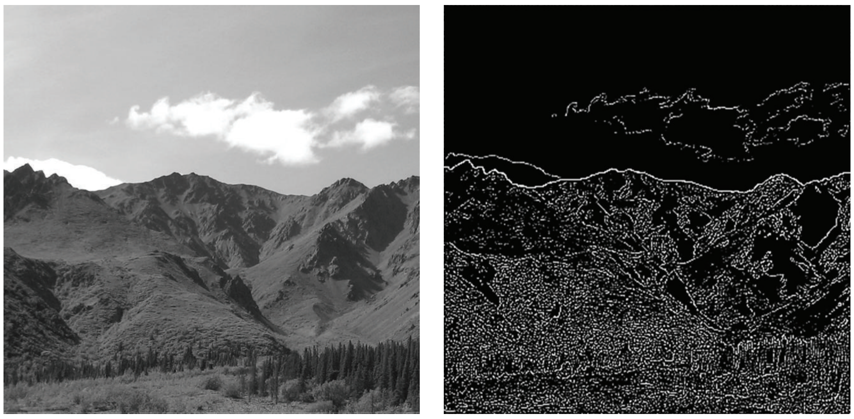
(a) (b)

Figure 14: The original image and the image after applying Canny edge detection.

Table 2: Performance improvement of image registration and Canny edge detection on Cray XD1.

Algorithm name		Computing time (s)		Speedup	Resource utilization	
		Opteron 2.4G	Cray XD1		Slices	Built-in multipliers
Image registration*	Exhaustive search	157.347	16.193	9.72	10.766 (45%)	42 (18%)
	DWT-based search†	1.298	0.829	1.57	20.205 (85%)	108 (55%)
		End-to-end throughput (MB/s)				
Canny edge detection		2.20	1.196	543.6	15.015 (63%)	200 (86%)

*The sizes of reference and test image are both 1024×1024. The search spaces of $\Delta\Theta$, ΔX, and ΔY are all from -8 to $+8$.
†Three levels of DWT are performed before the search, which is based on LL coefficients only.

clock cycle. Furthermore, the interconnect has separate input and output channels, both of them are 64-bit wide. The maximum operating clock rate for the user logic is 200 MHz, which is achieved in all our hardware implementations.

For the exhaustive search approach, the reference image and the test image are stored in two separate local memory modules. Every time, two possible tuples of (θ, t_x, t_y) are tested, that is, two different rigid-body transformations are performed on the test image simultaneously and produce two transformed test images, T' and T'', which are stored in two additional local memory modules separately. The calculation of the correlation coefficients between these two images and the reference image is carried concurrently as well. All the hardware modules are fully pipelined in our design in order to realize the full potential of a hardware implementation. Compared with the performance of the software implementation running on a single microprocessor, an AMD Opteron 2.4 GHz, the performance of the hardware implementation on a single FPGA device is approximately 10× better, as shown in Table 2. The performance improvement of hardware implementation compared with software implementation is mainly due to the fully pipelined hardware design. For example, during the correlation coefficient calculation step, it only takes one clock cycle for hardware to calculate 8 pixels. On the other hand, the software implementation has to calculate one pixel each time. The measured time on the Cray XD1 is the end-to-end time including data transfer time between the μP and the FPGA and the hardware processing time on the FPGA. In the 16.193 s, data transfer time is merely 6 ms. Therefore, the performance of hardware implementation in this case is almost upper-bounded by the available number of local memory banks since only 45% of the FPGA slices are used. As shown in (5), the hardware processing time is reciprocal to the number of memory banks that are used to store transformed test images. Therefore the speedup of hardware implementation compared with software version is linearly proportional to the number of local memory banks.

For the DWT-based approach, three levels of DWT are applied on both the test image and the reference image before the registration process starts. All the original image and decomposed images of different levels are stored in local memory modules, as depicted in Figure 8. Since the whole process consists of two steps, the DWT decomposition and the search, the hardware utilization almost doubles in this case. More built-in multipliers are used in the DWT decomposition as well. Compared with the software implementation, the hardware is barely 2× faster for the DWT-based search approach. The comparatively low speedup achieved by hardware in this case is due to the fact that the amount of computation is significantly reduced because of the DWT decomposition.

For both cases, the hardware implementation is mainly characterized by the local memory architecture, and the performance can be improved if more processing concurrency is allowed, given that more independent local memory modules are available.

For the category of applications that use the interconnect for data access, a Canny edge detection algorithm is implemented following the architecture in Figure 11. The Canny edge detector comprises four stages in which each stage takes the output from the preceding stage and feeds its output to the following stage as follows:

$$\begin{aligned}\text{Input Image} &\longrightarrow \text{Gaussian Smoothing}\\ &\longrightarrow \text{Compass Edge Detecting}\\ &\longrightarrow \text{Non-maximum Suppression}\\ &\longrightarrow \text{Edge Tracingand Thresholding.}\end{aligned} \tag{11}$$

Because the interconnect is 8-byte wide, 8 edge detection operators are implemented and execute in parallel. Figure 14 shows the original image and the corresponding output after applying the Canny edge detection algorithm. In Canny edge detection algorithm, the processing of each pixel in the output image involves neighbor pixels in the input image. Furthermore, this processing consists of 4 stages and takes hundreds of clock cycles as latency. Due to the fully pipelined design in hardware implementation, the FPGA device is capable of computing 8 pixels in the output image every clock cycle no matter how complicated the computation of each pixel is. On the other hand, the pixels in the output image are computed one by one in the software implementation. Further, it would take thousands of cpu cycles to compute one pixel. Therefore hardware implementation of the Canny edge detection algorithm achieves 544× speedup compared with the corresponding software implementation. Higher speedup can be achieved if multiple images can be processed simultaneously given several interconnect channels are connected to the same FPGA co-processor and there are enough hardware resources to implement multiple instances of edge detection operators.

6. Conclusions

In this paper, we demonstrate how the parallelism of a hardware design on reconfigurable computers is parameterized by the co-processor architecture, particularly the number and the data width of local memory banks and the interconnect. Image registration algorithms based on rigid-body transformation are adopted as a case study to represent applications that use local memory. Two related; however, different algorithms, the exhaustive search algorithm and the DWT-based search algorithm, are described in detail. For the exhaustive search algorithm, the performance is linearly proportional to the available number of local memory banks. On the other hand, the DWT-based search algorithm improves the efficiency by applying DWT decomposition on both the reference and test images before the search in order to reduce the search scope. Compared with software implementations, hardware implementations of exhaustive search and DWT-based search achieve 10× and 2× speedup, respectively. For the category of applications that directly access the host interconnect, edge detection is selected as a case study. A streaming data transfer mode in which the user logic and the interconnect are chained into a pipeline is proposed. A user logic hardware architecture whose parallelism is decided by the data width of the interconnect is discussed in detail. A Canny edge detection application following the proposed architecture is capable of achieving 544× speedup compared with the corresponding software design.

As a future work, we will extend our work to the Tile 64 platform [21]. Parameters such as the external memory bandwidth and the intertile bandwidth will be taken into account to implement image processing algorithms. We expect to take advantages from Tile 64's capability of creating multicore pipelines internally.

Acknowledgment

This work was supported in part by Arctic Region Supercomputing Center (ARSC) at the University of Alaska Fairbanks.

References

[1] *Reconfigurable Application-Specific Computing User's Guide (007-4718-007)*, Silicon Graphics, 2008.

[2] *SRC Carte C Programming Environment v2.2 Guide (SRC-007-18)*, SRC Computers, 2006.

[3] *Cray XD1 FPGA Development (S-6400-14)*, Cray, 2006.

[4] T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. Kindratenko, and D. Buell, "The promise of high-performance reconfigurable computing," *Computer*, vol. 41, no. 2, pp. 69–76, 2008.

[5] C. Torres-Huitzil and M. Arias-Estrada, "FPGA-based configurable systolic architecture for window-based image processing," *Eurasip Journal on Applied Signal Processing*, vol. 2005, no. 7, pp. 1024–1034, 2005.

[6] J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, "FPGA-Based face detection system haar classifiers," in *Proceedings of the 7th ACM SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '09)*, pp. 103–111, Monterey, Calif, USA, February 2009.

[7] J. Le Moigne, W. J. Campbell, and R. F. Cromp, "An automated parallel image registration technique based on the correlation of wavelet features," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 40, no. 8, pp. 1849–1864, 2002.

[8] O. Dandekar and R. Shekhar, "FPGA-accelerated deformable image registration for improved target-delineation during CT-guided interventions," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 1, no. 2, pp. 116–127, 2007.

[9] B. Zitová and J. Flusser, "Image registration methods: a survey," *Image and Vision Computing*, vol. 21, no. 11, pp. 977–1000, 2003.

[10] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.

[11] M. Huang, O. Serres, T. El-Ghazawi, and G. Newby, "Implementing image registration algorithms on reconfigurable computer," in *Proceedings of the 10th Military and Aerospace Programmable Logic Devices Conference (MAPLD '08)*, September 2008.

[12] M. Huang, O. Serres, S. Lopez-Buedo, T. El-Ghazawi, and G. Newby, "An image processing architecture to exploit I/O

bandwidth on reconfigurable computers," in *Proceedings of the 4th Southern Conference on Programmable Logic (SPL '08)*, pp. 257–260, San Carlos de Bariloche, Argentina, March 2008.

[13] M. S. Puranik and D. C. Gharpure, "FPGA implementation of MFNN for image registration," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (ICFPT '02)*, pp. 364–367, December 2002.

[14] S. Liu, N. Li, and W. Wang, "Efficient codesign for geology image processing," in *Proceedings of the 49th Midwest Symposium on Circuits and Systems (MWSCAS '06)*, vol. 1, pp. 272–275, San Juan, Puerto Rico, USA, August 2006.

[15] E. El-Araby, M. Taher, T. El-Ghazawi, and J. L. Moigne, "Automatic image registration for remote sensing on reconfigurable computers," in *Proceedings of the 9th Military and Aerospace Programmable Logic Devices Conference (MAPLD '06)*, 2006.

[16] M. A. de Barros and M. Akil, "Low level image processing operators on FPGA: implementation examples and performance evaluation," in *Proceedings of the 12th IAPR Intarnational Conference on Pattern Recognition*, vol. 3, pp. 262–267, October 1994.

[17] V. Muthukumar and D. V. Rao, "Image processing algorithms on reconfigurable architecture using HandelC," in *Proceedings of the EUROMICRO Systems on Digital System Design (DSD '04)*, pp. 218–226, Rennes, France, August-September 2004.

[18] K. T. Gribbon, D. G. Bailey, and C. T. Johnston, "Using design patterns to overcome image processing constraints on FPGAs," in *Proceedings of the 3rd IEEE International Workshop on Electronic Design, Test and Applications (DELTA '06)*, pp. 47–53, Kuala Lumpur, Malaysia, January 2006.

[19] L. G. Brown, "Survey of image registration techniques," *ACM Computing Surveys*, vol. 24, no. 4, pp. 325–376, 1992.

[20] T. A. El-Ghazawi, P. Chalermwat, and J. L. Moigne, "Wavelet-based image registration on parallel computers," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC '97)*, pp. 20–28, November 1997.

[21] http://www.tilera.com/.