Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2009, Article ID 162078, 10 pages
doi:10.1155/2009/162078

Research Article

FPGA Accelerator for Wavelet-Based Automated Global Image Registration

Baofeng Li, Yong Dou, Haifang Zhou, and Xingming Zhou

National Laboratory for Parallel and Distributed Processing, National University of Defense Technology, Changsha 410073, China

Correspondence should be addressed to Baofeng Li, lbf@nudt.edu.cn

Received 14 February 2009; Accepted 30 June 2009

Recommended by Bertrand Granado

Wavelet-based automated global image registration (WAGIR) is fundamental for most remote sensing image processing algorithms and extremely computation-intensive. With more and more algorithms migrating from ground computing to onboard computing, an efficient dedicated architecture of WAGIR is desired. In this paper, a BWAGIR architecture is proposed based on a block resampling scheme. BWAGIR achieves a significant performance by pipelining computational logics, parallelizing the resampling process and the calculation of correlation coefficient and parallel memory access. A proof-of-concept implementation with 1 BWAGIR processing unit of the architecture performs at least 7.4X faster than the CL cluster system with 1 node, and at least 3.4X than the MPM massively parallel machine with 1 node. Further speedup can be achieved by parallelizing multiple BWAGIR units. The architecture with 5 units achieves a speedup of about 3X against the CL with 16 nodes and a comparative speed with the MPM with 30 nodes. More importantly, the BWAGIR architecture can be deployed onboard economically.

1. Introduction

With the rapid innovations of remote sensing technology, more and more remote sensing image processing algorithms are enforced to be finished onboard instead of at ground station to meet the requirement of processing numerous remote sensing data realtimely. Image registration [1, 2] is the basis of many image processing operations, such as image fusion, image mosaic, and geographic navigator. Considering the computation-intensive and memory-intensive characteristics of remote sensing image registration and the limited computing power of onboard computers, to implement image registration efficiently and effectively with dedicated architecture is of great significance.

In the past twenty years, FPGA technology has been developed significantly. The volume and performance of FPGA chip have increased greatly to adapt many large-scale applications. Due to its excellent reconfigurability and convenient design flow, FPGA has been the most popular choice for hardware designers to implement kinds of application-specific architectures. Therefore, to implement the remote sensing image registration in FPGA efficiently is just the point of this paper. Though Carstro-Pareja et al. [3, 4] have proposed a fast automatic image registration (FAIR) architecture of mutual information-based 3D image registration for medical imaging applications, few works addressing hardware acceleration of the remote sensing image registration have been reported.

Many approaches have been proposed for remote sensing image registration. As for hardware implementation, only the automated algorithms are suitable because onboard computing demands that the algorithms should be accurate and robust and operate without manual intervention. Proposed automated remote sensing image registration algorithms can be classified into two categories, CPs-based algorithms [5–12] and global algorithms [13–22]. In the former, some matched control points (CPs) are extracted from both images automatically to

decide the final mapping function. However, the problem is that it is difficult to automatically determine efficient CPs. The selected CPs need to be accurate, sufficient, and with even distribution. Missing or spurious CPs make CPs-based algorithms unreliable and unstable [23]. Hence, CPs-based algorithms are not in our consideration.

Automated global registration, however, is an approach that does not rely on point-to-point matching. The final mapping function is computed globally over the images. Therefore, the algorithms are stable and robust and easy to be automatically processed. One of the disadvantages of global registration is that it is computationally expensive. Fortunately, the wavelet decomposition helps to relieve this situation because it provides us a way to obtain the final result progressively. A wavelet-based automated global image registration (WAGIR) algorithm for the remote sensing application proposed by Moigne et al. [13–15] has been proved to be efficient and effective. In WAGIR, the lowest-resolution wavelet subbands are firstly registered with a rough accuracy and a wider search interval, and a local best result is obtained. Nextly, this result is refined repeatedly after the iterative registrations on the higher-resolution subbands. The final result is obtained at the highest-resolution subbands, viz. the original images.

Many parallel schemes of WAGIR are proposed in previous works, such as parameter-parallel scheme (PP), image-parallel (IP) scheme, hybrid-parallel (HP) scheme which merges PP and IP, and group-parallel (GP) scheme [13, 24–27] which are implemented targeting large, expensive supercomputers, cluster system or grid system that are impractical to be deployed onboard. In this paper, we propose a block wavelet-based automated global image registration (BWAGIR) architecture based on a block resampling scheme. The architecture with 1 processing unit outperforms the CL cluster system with 1 node by at least 7.4X, and the MPM massively parallel machine with 1 node by at least 3.4X. And the BWAGIR with 5 units achieves a speedup of about 3X against the CL with 16 nodes and a comparable speed with the MPM with 30 nodes. More importantly, our work is targeting onboard computing.

The remainder of this paper is organized as follows. In Section 2, the traditional WAGIR algorithm is reviewed and analyzed based on the hierarchy architecture. The proposed block resampling scheme is detailed in Section 3. And the architecture of BWAGIR is presented in Section 4. Section 5 gives out the proof-of-concept implementation and the experimental results with comparison to several related works. Finally, this paper is concluded in Section 6.

2. Wavelet-Based Automatic Global Image Registration Algorithm

Image registration is the process that determines the most accurate match between two images of the same scene or object. In the global registration process, one image is registered according to another known standard image. We refer to the former as *input image*, the latter as *reference image*, the best matching image as *registered image*, and the image after each resampling process as *resampled image*.

2.1. Review of WAGIR Algorithm. WAGIR can be described as the pseudocode in Algorithm 1. Here assume that the LL subbands form the feature space; 2D rotations and translations are considered as search space; the search strategy follows the multiresolution approach provided by wavelet decomposition; the cross correlation coefficient is adopted as similarity metric. Firstly, an N-level wavelet decomposes the *input image* and the *reference image* with size of $M \times M$ into *nLLi* and *nLLr* sequences where n represents corresponding decomposition level. Then $NLLi$ and $NLLr$ with the lowest resolution are registered with accuracy of $\delta 2^N$. A local best combination of rotations and translations $(best\theta, bestX, bestY)$ is obtained and used as the search center of registering the next level subbands, $(N-1)LLi$ and $(N-1)LLr$. And another combination with accuracy of $\delta 2^{N-1}$ is gained. This process iterates until the overall best result with expected accuracy δ, is retrieved after registering the original *input image* $(0LLi)$ and *reference images* $(0LLr)$. Finally, a resampling process is carried out to get the *registered image*.

At each level, the algorithm shown in Algorithm 2 is employed to register *nLLi* and *nLLr*. The result of previous level $(\theta C, XC, YC)$ is used as the search center. For each combination of rotations and translations, the algorithm shown in Algorithm 3 is performed to get a *resampled image* of *nLLr*. Then a correlation coefficient is calculated to measure the similarity between the resampled *nLLr* and the *nLLi*. The combination corresponding to the maximal correlation coefficient is the best result of current level. The resampling algorithm is performed by sequentially selecting one *registered image* location once, calculating the corresponding coordinate of the selected location in the *reference image*, accessing the neighboring 4×4 pixel window in the *reference image*, calculating the corresponding interpolation weights according to the computed coordinate, and finally calculating the pixel value of the selected location by the cubic convolution interpolation method. The correlation coefficient is calculated with (1):

$$C(A,B) = \frac{\sum_{i=0}^{M-1}\sum_{j=0}^{M-1}\left(A_{ij}\times B_{ij}\right) - (1/M^2)\left(\sum_{i=0}^{M-1}\sum_{j=0}^{M-1}A_{ij}\right)\times\left(\sum_{i=0}^{M-1}\sum_{j=0}^{M-1}B_{ij}\right)}{\sqrt{\left(\sum_{i=0}^{M-1}\sum_{j=0}^{M-1}A_{ij}^2 - (1/M^2)\left(\sum_{i=0}^{M-1}\sum_{j=0}^{M-1}A_{ij}\right)^2\right)\times\left(\sum_{i=0}^{M-1}\sum_{j=0}^{M-1}B_{ij}^2 - (1/M^2)\left(\sum_{i=0}^{M-1}\sum_{j=0}^{M-1}B_{ij}\right)^2\right)}}. \tag{1}$$

Input: *input image* and *reference image*
Output: *registered image*
1 *Initialize registration process* (*wavelet level* $-N$, *search scope* $-$ *rotation angle:* ($\theta scope\_L, \theta scope\_R$), *horizontal offset:* ($Xscope\_L, Xscope\_R$), *and vertical offset:* (*Yscope_L, Yscope_L*));
2 *Perform wavelet decomposition of the input image and reference image;*
3 *best* $\theta = 0$; *best* $X = 0$; *best* $Y = 0$;
4 *step* $\theta = \delta 2^N$; *step* $X = \delta 2^N$; *step* $Y = \delta 2^N$;
5 for ($n = N; n \geq 0;\ n-$) **do**
    (*width, height*) = (*image_width*/2^n, *image_height*/2^n);
    //registering at current wavelet level based on the results of previous level.
    Perform **Register** (*nLLi, nLLr, best* θ, *bestX, bestY, step* θ, *stepX, stepY*);
    ($\theta scope\_L, \theta scope\_R$) = ($-$*step* θ, *step* θ);
    (*Xscope_L, Xscope_R*) = ($-$*stepX, stepX*);
    (*Yscope_L, Yscope_R*) = ($-$*stepY; stepY*);
    step θ /=2; *stepX* /=2; *stepY* /= 2;
    bestX = 2; bestY* = 2; //size of next wavelet subband is twice of current*
6 Resample (*input image, best* θ, *bestX, bestY, registered image*);
    // last resampe to obtain the result image.
7 Over.

Algorithm 1: Main WAGIR algorithm.

Register (the registering algorithm)
Input: *nLLi, nLLr,* θ*center, Xcenter, Ycenter,* $step\theta$*, stepX, stepY*
Output: local *best* θ, *bestX, and bestY*
1 (*angle, x, y*) = ($\theta scope\_L, Xscope\_L, Yscope\_L$); *//control variables*
2 max_*co* = -1; *// record the maximum correlation*
  // the registration processing
3 while *(angle <= θscope_R)* **do**
  while *(x <= Xscope_R)* **do**
    while *(y <= Yscope_R)* **do**
      (θC, XC, YC) = (θcenter + angle, Xcenter + x, Ycenter + y);
      // resample nLLr with θC, XC, YC to get registered image_out
      Perform **Resample** *(nLLr, image_out, θC, XC, YC);*
      // compute the correlation betwwen image_out and nLLi
      corre = **Correlation** *(image_out, nLLi);*
      if *(corre* > max_*co)* **then**
        max_*co = corre;*
        (bestθ, bestX, bestY) = (θC, XC, yC);
      y = y + stepY;
    x = x + stepX;
    y = Yscope_L;
  angle = angle + stepθ;
  x = Xscope_L;
4 Over.

Algorithm 2: The registering algorithm.

2.2. Analysis of WAGIR Algorithm. All analysis are based on a common assumption of the hierarchy architecture shown in Figure 1. The off-chip external memory is used to store the tremendously growing image data. The on-chip memory serves as an buffer to bridge the speed gap between external memory and accelerator.

In WAGIR, for each possible combination of rotations and translations, a resampling process is performed, and a correlation coefficient is calculated to decide which one is the best transformation between the *input image* and *reference image*. Runtime profiles from a software implementation of WAGIR listed in Table 1 show that the resampling process (without the time to compute the correlation coefficient) is the most time-consuming, and the calculation of correlation coefficient is essential because each resampling process corresponds one calculation of correlation coefficient though

```
Resample (the resample algorithm)
Input: nLLr; transθ, transX, transY
Output: image_out
1 (w, h) = (width/2^n, height/2^n); //width and height of the input nLLr
   //compute the cubic convolution weights for a xLen × tab × yLen × tab template
2 cubicTable (4, 4, tab);
3 for (ty = 0; ty < h; ty++) do
      for (tx = 0; tx < w; tx++) do
         //The inverse mapping function
         x = cos(transθ) ∗ (tx − transX − w/2) + sin(transθ) ∗ (ty − transY − h/2) + w/2;
         y = − sin(transθ) ∗ (tx − transX − w/2) + cos(transθ) ∗ (ty − transY − h/2) + h/2;
         (x_int, y_int) = (int x, int y);
         (x_fra, y_fra) = (x − x_int, y − y_int);
         (f_x, f_y) = ((int)(x_fra∗tab), (int)(y_fra∗tab));
         //read the corresponding xLen ∗ yLen weights from cubicTable to ct
         ct = cubicTable + (f_y ∗ tab + f_x) ∗ mem_size;
         if ((0 < x_int < (w − 2)) && (0 < y_int < (h − 2))) then
            // read the corresponding xLen × yLen coefficients from nLLr to st
            Read (nLLr, x_ int − 1, y_ int − 1, xLen, yLen, st);
            pixel_d = 0;
            for (ch = 0; ch < 4; ch++) do
               for (cw = 0; cw < 4; cw++) do
                  pixel_d += (∗ct)∗(∗st);
                  ct++; st++;
            if (pixel_d > 255) then pixel_ d = 255;
            if (pixel_d < 0) then pixel_ d = 0;
            pixel = (char)pixel_d;
            ∗(image_out + ty ∗ w + tx) = pixel;
4 Over
```

Algorithm 3: The resampling algorithm.

Table 1: Runtime profiles from a software implementation of WAGIR (three-level wavelet decomposition, grayscale image, profiling platform is Intel Celeron(R) 1.7 GHz CPU, 256M DDR266 SDRAM ×2, Microsoft VC 6.0 and WindowsXP Prof.).

Image size	Wavelet dec.	Resampling process	Correlation cal.
512×512	0.4	95.6	0.0001
$1K \times 1K$	0.6	94.2	0.0001
$2K \times 2K$	0.6	93.8	0.0000
$3K \times 3K$	0.7	93.7	0.0000

it consumes a little execution time. For example, to register an *input image* and a *reference image* with size of $M \times M$ in search space $[\theta L, \theta R] \times [xL, xR] \times [yL, yR]$, $(\theta R - \theta L) \times (xR - xL) \times (yR - yL)$ resampling processes and $(\theta R - \theta L) \times (xR - xL) \times (yR - yL)$ calculations of coefficient are needed. For each resampling process, $M \times M$ resampling operations are needed. Though wavelet decomposition can relieve this situation, the computation requirement remains significant. Therefore, to accelerate WAGIR is just to accelerate the resampling process and the calculation of correlation coefficient.

In addition to the great computation requirement, WAGIR also has great memory requirement. In each resampling process, for each location of the *resampled image*, a neighboring 4×4 pixel window in the *reference image* is needed. That means that $16M^2$ memory accesses are required for each resampling process. Meanwhile, a total of $2M^2$ accesses are also required for each calculation of correlation coefficient. Considering the great amount of resampling processes and calculations of correlation coefficient, the total access amount comes out of a massive number. Even worse, the whole *reference image* may be needed maximally to compute one row pixels of the *resampled image* when the rotation angle is $\pm\pi/4$ as shown in Figure 2. This demands that the on-chip memory should be capable to hold the whole image. If not, the amount of memory access grows significantly. And also, each *resampled image* should be buffered on-chip because it is needed in calculation of correlation coefficient. It is infeasible to assign so great on-chip memory for hardware implementation because of the mass size of remote sensing images and scarcity of on-chip memory resources. Therefore, a good memory scheduling strategy is imperative.

3. Block Resampling Scheme

To accommodate the great computation and memory requirements of WAGIR, a block resampling scheme is employed. The foundation is to produce the *resampled image* block by block because the computations of different locations are absolutely independent and irrelevant. The

```
Block Resample (the block resample algorithm)
Input: nLLr; transθ, transX, transY
Output: image_out
1  (w,h) = (width/2^n, height/2^n); // width and height of the input nLLr
   //compute the cubic convolution weights for a xLen × tab × yLen × tab template
2  cubicTable (4, 4, tab);
3  for (s = 0; s <h/S; s++) do
     for (r = 0; r <w/S; r++) do
       for (ty = 0; ty <S;ty++) do
         for (tx = 0; tx <S; tx++) do
           //The inverse mapping function
           x =  cos(transθ) ∗ (tx − transX − w/2) + sin(transθ) ∗ (ty − transY − h/2) + w/2;
           y = − sin(transθ) ∗ (tx − transX − w/2) + cos(transθ) ∗ (ty − transY − h/2) + h/2;
           (x_int, y_int) = (int x, int y);
           (x_fra, y_fra) = (x − x_int, y − y_int);
           (f_x, f_y) = ((int)(x_fra∗tab), (int)(y_fra∗tab));
           //read the corresponding xLen ∗ yLen weights from cubicTable to ct
           ct = cubicTable + (f_y ∗ tab + f_x) ∗ mem_size;
           if ((0 < x_int < ( w − 2)) && (0 < y_int < ( h − 2))) then
             // read the corresponding xLen × yLen coefficients from nLLr to st
             Read(nLLr, x_int-1, y_int-1, xLen, yLen, st);
             pixel_d = 0;
             for (ch = 0; ch < 4;ch++) do
               for (cw = 0; cw  < 4; cw++) do
                 pixel_d += (∗ct)∗(∗st);
                 ct++; st++;
             if (pixel_ d > 255) then pixel_ d = 255;
             if (pixel_ d  < 0) then pixel_ d = 0;
             pixel = (char)pixel_d;
             ∗(image_out + ty ∗ w + tx) = pixel;
4  Over
```

Algorithm 4: Block resampling algorithm.

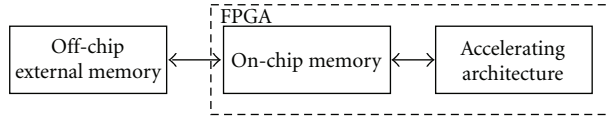

Figure 1: Assumption of hierarchy architecture.

pseudocode in Algorithm 4 describes the block resampling scheme in which the *resampled image* is computed sequentially in consecutive $S \times S$ subblocks.

The reason what causes the great memory requirement is that the *resampled image* is generated row by row in traditional resampling algorithm. This way of computation results in great scope of preloading the *reference image*. According to the mapping function (2), the scope of required *reference image* pixels to compute one row pixels of the *resampled image* is $[0, M] \times [0, M]$ maximally, that is, the whole *reference image*. But the scope to compute an $S \times S$ subblock is just $[((1 - \sqrt{2})/2)S, ((1 + \sqrt{2})/2)S] \times [((1 - \sqrt{2})/2)S, ((1 + \sqrt{2})/2)S]$. Because $S \ll M$, the preloading scope is decreased greatly. Accordingly, the size of required on-chip memory is reduced significantly:

$$\begin{aligned} x &= \cos(\text{trans}\,\theta) * \left(tx - \text{trans}\,X - \frac{M}{2}\right) \\ &\quad + \sin(\text{trans}\,\theta) * \left(ty - \text{trans}\,Y - \frac{M}{2}\right) + \frac{M}{2}, \\ y &= -\sin(\text{trans}\,\theta) * \left(tx - \text{trans}\,X - \frac{M}{2}\right) \\ &\quad + \cos(\text{trans}\,\theta) * \left(ty - \text{trans}\,Y - \frac{M}{2}\right) + \frac{M}{2}. \end{aligned} \tag{2}$$

Another benefit gained from the block resampling scheme is that the calculations of all pixels within one block

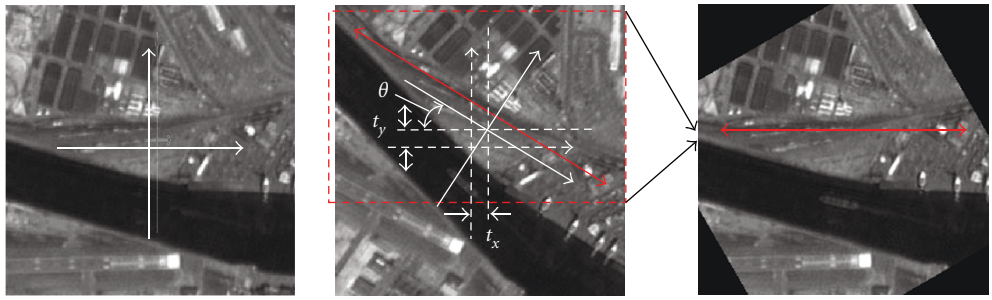

Figure 2: Memory requirement of WAGIR.

only need once preloading, and the amount of memory access is decreased. In traditional resampling algorithm, the pixels of *reference image* must be loaded from the external memory again and again if there is no enough on-chip memory to store the whole image. But in the block resampling scheme, the block size is decided by the available on-chip memory.

4. The BWAIR Architecture

As mentioned above, the resampling process and the calculation of correlation coefficient account for major of the execution time. Therefore the BWAIR architecture aims to accelerate WAGIR by accelerating the resampling algorithms and the calculation of corresponding correlation coefficient.

The BWAGIR architecture is detailed in Figure 3. The coordinate calculation module computes the coordinate of the pixel in *reference image* corresponding to each location in the *resampled image*. The interpolation weights calculation module is responsible to compute the 16 weights for the 4×4 interpolation window. The *reference image* RAM controller loads the neighboring 4×4 window. The resampled pixel calculation module is in charge of computing the values of resampled pixels. The *input image* RAM controller loads the *input image* pixels for calculation of correlation coefficient. The correlation calculation module computes the correlation coefficient. And the FIFOs are set to bridge the speed gap among the modules mentioned above.

Proposed BWAGIR architecture optimizes the resampling process and the calculation of correlation coefficient by means of parallelizing the resample process and corresponding calculation of correlation coefficient, pipelining all calculation modules, and parallel memory access.

4.1. Parallelizing Resampling and Calculation of Correlation. In a standard software implementation, the resampling process and the calculation of correlation coefficient are performed sequentially. The calculation of correlation coefficient starts after all the pixels of the *resampled image* are produced. This means that the *resampled image* must be written back into the external memory or stored in extra on-chip memory after the resampling and then read back when calculating the correlation coefficient. Extra memory volume and memory access are evident.

In BWAGIR architecture, we partition the correlation calculation into two steps.

(1) Calculate the sum of pixels of *input image* ($\sum_{i=0}^{M-1}\sum_{j=0}^{M-1} A_{ij}$), the sum of pixels of *resampled image* ($\sum_{i=0}^{M-1}\sum_{j=0}^{M-1} B_{ij}$), the sum of square of pixels of *input image* ($\sum_{i=0}^{M-1}\sum_{j=0}^{M-1} A_{ij}^2$), the sum of square of pixels of *resampled image* ($\sum_{i=0}^{M-1}\sum_{j=0}^{M-1} B_{ij}^2$), and the sum of the production of pixels of *input image* and corresponding pixels of *resampled image* ($\sum_{i=0}^{M-1}\sum_{j=0}^{M-1}(A_{ij} \times B_{ij})$).

(2) Calculate the final correlation coefficient according to (1).

This partition can avoid the extra memory volume and memory access. Once a pixel in the *resampled image* is produced, it is computed in step 1 and then discarded. Once step 1 finishes the calculations of all pixels, the five sums are sent to step 2 to finalize the calculation of correlation coefficient. Therefore, the resampling process parallel with the calculation of correlation coefficient.

4.2. Pipelining. All the calculation modules are pipelined to improve the system throughput and operating frequency. As shown in Figure 3, the BWAGIR is divided into four macrostages according to the processing flow. The first stage calculates the coordinate of the pixel in the *reference image* corresponding to each location of *resampled image* and writes the integral and fractional components into according FIFOs. At the second stage, the Reference Image RAM Controller reads the neighboring 4×4 *reference image* pixel window, and at the same time, 16 interpolation weights are produced by the Interpolation Weights Calculation Module. Stage 3 calculates the value of each location in the *resampled image* by multiplying the pixels with its corresponding weights and adding these product together. Finally, the Correlation Calculation Module computes the correlation coefficient by means described in Section 4.1. With pipelining, the total time of performing the resampling process and corresponding correlation calculation once becomes equal to the product of the worst pipeline stage time and the number of pixels in the *resampled image*.

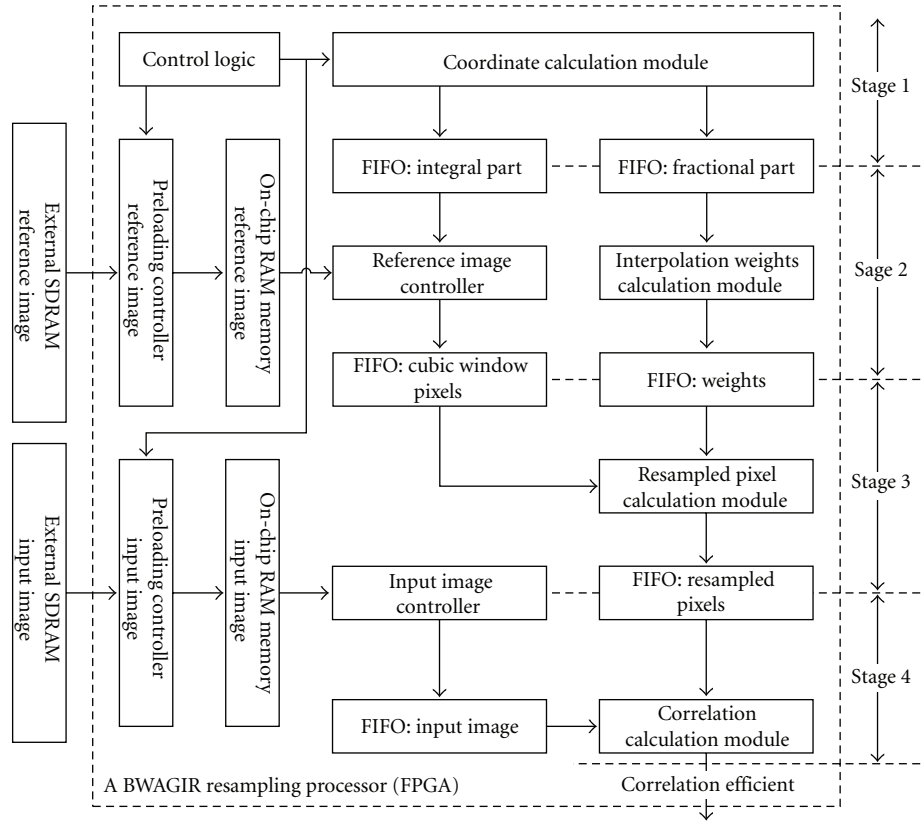

Figure 3: The BWAGIR architecture.

Table 2: Comparison of the registration time (milliseconds) with the CL cluster system.

Size		CL					BWAGIR	
		1node	2nodes	5nodes	15nodes	16nodes	1unit	5units
512×512	PP	103.11	51.80	20.82	7.64	6.86	8.7	1.8
	IP	102.98	61.00	38.54	27.50	27.44		
	HP	103.22	51.60	20.83	7.94	8.82		
	GP	103.22	51.79	20.85	7.25	6.80		
$1K \times 1K$	PP	345.93	173.77	69.86	25.62	23.02	39.4	7.9
	IP	345.94	187.43	88.72	44.66	43.31		
	HP	345.90	172.22	69.84	25.01	24.56		
	GP	345.95	172.32	69.86	24.21	22.90		
$3K \times 3K$	PP	2849.95	1445.13	575.63	213.55	192.41	385.7	75.5
	IP	2849.88	1440.52	626.68	231.41	218.20		
	HP	2849.97	1442.60	575.62	207.36	191.83		
	GP	2849.96	1439.75	576.02	204.50	191.87		

4.3. Parallel Memory Access. Parallelizing the resampling process and calculation of correlation coefficient demands parallel access to *input image* and *reference image*. But the way of accessing the *input image* differs with that of accessing the *reference image*. The *input image* is accessed sequentially, and the *reference image* is accessed by 4×4 window. Therefore, two external memories are used to store the *input image* and the *reference image,* respectively. And they are preloaded into the respective on-chip RAMs block by block.

As mentioned above, the performance of the pipeline is decided by the worst stage calculation time. The four pipeline stages differ in data source and operation. The worst case is the second stage because a 4×4 neighborhood, that is, 16 pixels, is loaded from on-chip *reference image* RAM. If these pixels are loaded normally, it takes at least 16 cycles to calculating each location in the *resampled image*. This restraints the throughput of the pipeline significantly. As a rule, multibank memory organization can settle this problem by distributing the sequential multiple access to different

Table 3: Comparison of the registration time (milliseconds) with the MPM parallel machine.

Size		MPM					BWAGIR	
		1node	5nodes	15nodes	16nodes	30nodes	1unit	5units
512×512	PP	39.052	7.917	3.116	2.635	1.714	8.7	1.8
	IP	39.051	10.414	5.350	5.022	4.189		
	HP	39.050	7.917	2.850	2.916	1.564		
	GP	39.055	7.917	2.750	2.633	1.450		
$1K \times 1K$	PP	145.625	29.317	11.318	9.567	6.184	39.4	7.9
	IP	145.629	32.188	12.841	12.142	8.683		
	HP	145.633	29.067	10.117	9.816	5.415		
	GP	145.633	29.183	10.082	9.665	5.271		
$3K \times 3K$	PP	1327.00	276.336	102.926	86.233	68.517	385.7	75.5
	IP	1326.24	292.485	105.821	100.466	75.243		
	HP	1327.25	270.950	91.967	87.015	46.801		
	GP	1328.00	267.150	88.350	87.717	45.516		

memory banks with separate ports. Because the 16 pixels are not consecutive, it is difficult to distribute them evenly into 16 banks. Therefore, we adopt a compromising strategy that the on-chip memory for the *reference image* is divided into 8 banks each of which has three ports—one for write and the other two for read. This makes convenient to write the 64-bit word which is composed of 8 consecutive 8 bits pixels into on-chip memory parallelly and load the 8 pixels (two lines in the 4×4 window) parallelly. Thereby, it takes only 2 cycles to load the 16 pixels within a window. Though this cannot match the speed of calculation modules yet (one result, one cycle), the stage 2 calculation time is decreased by 8 times.

4.4. Parallelizing Multiple BWAGIR Processing Units. The processing speed of proposed architecture can be further improved by parallelizing multiple BWAGIR processing units. There are two ways to achieve this.

(i) *Processing multiple blocks belonging to the same resampled image*. This way multiplies the preloading scope, and the on-chip memory volume. Another disadvantage is the data is not utilized sufficiently because each preloading only supports the calculation of one *resampled image*.

(ii) *Processing multiple blocks belonging to different resampled images*. This way enlarges the preloading scope a little because the blocks at the same position of different *resampled images* almost have the same preloading scope. But the on-chip memory volume is still multiplied because parallel processing demands great data memory bandwidth, that is, each processor requires an independent data memory. This parallel can decrease the memory access because each preloading supports the calculation of multiple *resampled images*.

Therefore the more economical way is to process multiple blocks which are at the same position of different *resampled images*.

5. Implementation and Experimental Results

As a proof of concept, the BWAGIR architecture is modeled with VerilogHDL, simulated with ModelSim SE 6.1d, synthesized with Quartus II 6.0, and implemented in an external prototype board with an Altera EP2S130F1020C3 FPGA. One BWAGIR unit occupies about 67% ALUTs (70557) and 35% memory bits (2345144) and can operate at a clock rate of 100 MHz. Two Micron MT16LSDT12864AG-1 GB PC133 SDRAMs are used as the external memories for *input image* and *reference image,* respectively. And the on-chip memories for *input image* and *reference image* are implemented with internal memory blocks. The prototype board is connected to a host computer with a USB cable. Only the registration component of WAGIR is executed on the board, and the other components are all performed on the host. Table 2 lists a comparison of the timings between the BWAGIR architecture and the CL machine which is a cluster system with 16 nodes; each node is equipped with Pentium4-1.7 G CPU and 512 MB local storage, and all nodes are connected by the 100 Mb/s Ethernet. Table 3 lists a comparison of the timings between the BWAGIR and the MPM machine which is a massively parallel computer with MIMD architecture and has 32 processors with 1 GB local storage for each processor, speed of MPM CPU is valued as 1.66 gigaflops/sec, topology of network is fat tree, and point-to-point bandwidth is 1.2 Gb/s. The images are processed with three-level wavelet decomposition and registered within the search space of $\theta \in (-16°, 16°)$, $x \in (-16, 16)$, $y \in (-16, 16)$.

It can be concluded what follows.

(i) The BWAGIR with 5 units may perform more than 5X faster than that with 1 unit because parallelizing multiple processing units cannot only improve the execution speed but also reduce the amount of memory access.

(ii) The BWAGIR with 1 unit outperforms 7.4X at least over kinds of parallel schemes on the CL with 1 node and 3.4X at least faster than the MPM with 1

node because the way of memory access cannot fully benefit from the traditional cache-based memory architectures present in most modern computers.

(iii) The BWAGIR with 5 units achieves a speedup of about 3X against the CL with 16 nodes, a speedup of greater than 1X against the MPM with 16 nodes, and a comparative speed with the MPM with 30 nodes. This is because numerous communications between nodes cut down the expected performance improvement of the parallel schemes.

It should be noted that the timings of BWAGIR with 1 unit is obtained on the prototype board actually, and the timings of the BWAGIR with 5 units are simulation times because our board cannot support parallelizing multiple units with limitation of available volume and number of FPGAs.

6. Conclusion

WAGIR algorithm for remote sensing application is extremely computation-intensive and demands execution times on the order of minutes, even hours, on modern desktop computers. Therefore a customized FPGA architecture is proposed in this paper to accommodate the great computational requirements of the algorithm and the trend of migration from ground computing to onboard computing.

To implement the algorithm in FPGA efficiently, a block resampling scheme is adopted to relieve the great computation and memory requirements. And based on the block scheme, the proposed BWAGIR architecture derives its improvement from (1) pipelining all computational logics, (2) parallelizing the resampling process and calculation of correlation coefficient, and (3) parallel memory access. A practical implementation with two standard PC133 SDRAMs, operating at 100MHz, outperforms 7.4X compared to the CL cluster system with 1 node and about 3.4X over the MPM machine with 1 node. This speedup is derived using just one BWAGIR processing unit. For further improvement, multiple units can be paralleled to implement arrays of processing units using VLSI or FPGAs to perform distributed image registration. Compared with the CL with 16 nodes, the BWAGIR architecture with 5 units can achieve about 3X speedup. And also it achieves a comparative speed with the MPM machine with 30 nodes. More importantly, our architecture can meet the requirement of onboard computing.

Acknowledgments

Our work is supported by the National Science Foundation of China under contracts no. 60633050 and no. 60621003 and the National High Technology Research and Development Program of China under contract no. 2007AA01Z106 and no. 2007AA12Z147.

References

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

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

[3] C. R. Castro-Pareja and R. Shekhar, "Hardware acceleration of mutual information-based 3D image registration," *Journal of Imaging Science and Technology*, vol. 49, no. 2, pp. 105–113, 2005.

[4] C. R. Castro-Pareja, J. M. Jagadeesh, and R. Shekhar, "FAIR: a hardware architecture for real-time 3D image registration," *IEEE Transactions on Information Technology in Biomedicine*, vol. 7, no. 4, pp. 426–434, 2003.

[5] J.-P. Djamdji, A. Bijaoui, and R. Maniere, "Geometrical registration of images: the multiresolution approach," *Photogrammetric Engineering & Remote Sensing*, vol. 59, no. 5, pp. 645–653, 1993.

[6] J. Flusser, "An adaptive method for image registration," *Pattern Recognition*, vol. 25, no. 1, pp. 45–54, 1992.

[7] B. S. Manjunath, C. Shekhar, and R. Chellappa, "A new approach to image feature detection with applications," *Pattern Recognition*, vol. 29, no. 4, pp. 627–640, 1996.

[8] A. D. Ventura, A. Rampini, and R. Schettini, "Image registration by recognition of corresponding structures," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 28, no. 3, pp. 305–314, 1990.

[9] Q. Zheng and R. Chellappa, "A computational vision approach to image registration," *IEEE Transactions on Image Processing*, vol. 2, no. 3, pp. 311–326, 1993.

[10] H. Li, B. S. Manjunath, and S. K. Mitra, "A contour-based approach to multisensor image registration," *IEEE Transactions on Image Processing*, vol. 4, no. 3, pp. 320–334, 1995.

[11] J. P. Djamdji and A. Bijaoui, "Disparity analysis: a wavelet transform approach," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 33, no. 1, pp. 67–76, 1995.

[12] H. H. Li and Y.-T. Zhou, "A wavelet-based point feature extractor for multisensor image restoration," in *SPIE Aerosense Wavelet Applications III*, vol. 2762 of *Proceedings of SPIE*, pp. 524–534, Orlando, Fla, USA, 1996.

[13] 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.

[14] J. Le Moigne, A. Cole-Rhodes, R. Eastman, et al., "Multisensor registration of earth remotely sensed imagery," in *Image and Signal Processing for Remote Sensing VII*, vol. 4541 of *Proceedings of SPIE*, pp. 1–10, Toulouse, France, September 2001.

[15] J. Le Moigne and I. Zavorin, "Use of wavelets for image registration," in *Wavelet Applications VII*, vol. 4056 of *Proceedings of SPIE*, pp. 99–104, Orlando, Fla, USA, April 2000.

[16] P. Thevenaz, U. E. Ruttimann, and M. Unser, "A pyramid approach to subpixel registration based on intensity," *IEEE Transactions on Image Processing*, vol. 7, no. 1, pp. 27–41, 1998.

[17] R. L. Allen, F. A. Kamangar, and E. M. Stokely, "Laplacian and orthogonal wavelet pyramid decompositions in coarse-to-fine registration," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3536–3541, 1993.

[18] Q.-S. Chen, M. Defrise, and F. Deconinck, "Symmetric phase-only matched filtering of Fourier-Mellin transforms for image registration and recognition," *IEEE Transactions on Pattern*

Analysis and Machine Intelligence, vol. 16, no. 12, pp. 1156–1168, 1994.

[19] J. C. Olivo, J. Deubler, and C. Boulin, "Automatic registration of images by a wavelet-based multiresolution approach," in *Wavelet Applications in Signal and Image Processing III*, vol. 2569 of *Proceedings of SPIE*, pp. 234–244, San Diego, Calif, USA, 1995.

[20] H. Shekarforoush, M. Berthod, and J. Zerubia, "Subpixel image registration by estimating the polyphase decomposition of cross power spectrum," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 532–537, 1996.

[21] R. C. Hardie, K. J. Barnard, and E. E. Armstrong, "Joint MAP registration and high-resolution image estimation using a sequence of undersampled images," *IEEE Transactions on Image Processing*, vol. 6, no. 12, pp. 1621–1633, 1997.

[22] R. J. Althof, M. G. J. Wind, and J. T. Dobbins, "A rapid and automatic image registration algorithm with subpixel accuracy," *IEEE Transactions on Medical Imaging*, vol. 16, no. 3, pp. 308–316, 1997.

[23] Y. C. Hsieh, D. M. McKeown, and F. P. Perlant, "Performance evaluation of scene registration and stereo matching for cartographic feature extraction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 214–238, 1992.

[24] A. El-Ghazaw and P. Chalermwa, "Wavelet-based image registration on parallel computers," in *Proceedings of ACM/IEEE on High Performance Networking and Computing (SuperComputing '97)*, p. 20, 1997.

[25] P. Chalermwat, *High performance automatic image registration for remote sensing*, Ph.D. Thesis, George Mason University, Fairfax, Va, USA, 1999.

[26] H. Zhou, X. Yang, H. Liu, and Y. Tang, "First evaluation of parallel methods of automatic global image registration based on wavelets," in *Proceedings of the International Conference on Parallel Processing (ICPP '05)*, pp. 129–136, 2005.

[27] H. Zhou, Y. Tang, X. Yang, and H. Liu, "Research on grid-enabled parallel strategies of automatic wavelet-based registration of remote-sensing images and its application in ChinaGrid," in *Proceedings of the 4th International Conference on Image and Graphics (ICIG '07)*, pp. 725–730, 2007.