RECONFIGURABLE ACCELERATION OF 3D IMAGE REGISTRATION

Kuen Hung Tsoi, Daniel Rueckert, Chun Hok Ho and Wayne Luk

Department of Computing
Imperial College London, UK
{khtsoi,dr,cho,wl}@doc.ic.ac.uk

ABSTRACT

This paper proposes techniques for accelerating a software based image registration algorithm for 3D medical images targeting a reconfigurable hardware platform. Various methods, including dedicated fixed point arithmetic, error model based bit width analysis, architecture exploration and application-specific memory modules, are applied to address issues from the software algorithm and to maximize the performance of FPGA technology. Based on the reconfigurability of FPGA devices, the system can be extended to swap modules optimized for different parameters, and to adopt more advanced registration algorithms. We show that a single core on 412MHz XC5VLX330T FPGA can evaluate a rigid transformation of a 3D image with 16 million voxels in 35ms. With 30 cores on an FPGA, it is over 108 times faster than a multi-threaded implementation running on a 2.5GHz Intel Quad-Core Xeon platform.

1. Introduction

Reconfigurable platforms have been widely adopted in accelerating different computational intensive digital signal processing algorithms. One example of suitable applications in reconfigurable computing is medical image analysis. Medical images can be obtained from various sources including X-rays, radionuclide scans such as Single Photon Emission Computed Tomography (SPECT) and Positron Emission Tomography (PET) scans, CT scans, ultrasound and magnetic resonance image (MRI). In practice, images acquired by sampling the subject in different time may have different coordinate systems due to axis rotation, subject position change or equipment variation. The process of transforming these images into an unified coordinate system is called image registration (IR). Comparison or integration of different images of the subject can be performed after the IR process.

The main objective of the IR process is not extracting the features from images directly but to optimize the set of parameters used in transforming target images to the coordinate system of source images. During the optimization process, parameters of translation, rotation and scaling are tuned iteratively and the cost function is based on the similarity of the transformed images.

The Image Registration Toolkit (ITK) [1] is an image registration framework which provides a board range of registration solutions for both 2D and 3D MRIs in object oriented C++ design. Despite the flexibility in software implementations, the improving resolution in imaging equipments, the increasing amount of images to be processed and the need of real time diagnosing ability make dedicated hardware platforms increasingly attractive in practical IR applications as higher computing power is required.

Studies have been conducted in the IR implementations on reconfigurable platforms. In 2003, a field programmable gate array (FPGA) based IR implementation using a B-spline free-form deformation algorithm is presented [2]. The design is implemented using the Handel-C language on a Xilinx Virtex II device (XC2V6000). Clocked at 67MHz, it achieves $3.2\times$ speedup over software version on a 2.6GHz Xeon CPU. In 2006, Altera releases a video and image processing suite for common image functions on FPGAs [3]. With a streaming interface and coupling with high level descriptions in MATLAB, this developing environment enables fast and optimized implementations on medical image processing. An FPGA based mutual information evaluation system for IR is proposed in [4]. The system utilizes a data flow model to improve hardware parallelism by sub-volume division. In 2007, reconfigurable computing platform is used for classifying brain tissues [5]. With the help of four Xilinx Virtex-4 LX200 FPGAs running at 100MHz, the software/hardware co-designed system achieves $3.5\times$ speedup over a pure software implementation on SGI Altix 350 system. In 2008, a multi-objective optimization framework for precision and resource trade off is proposed [6]. The system uses multiple copies of image in external memory for simultaneous voxels access. However, these studies do not provide a detailed error analysis on the accuracy. They also do not consider the reconfigurability of FPGA devices which can facilitate optimization for specific and changing parameters.

This paper describes a novel approach for optimizing a reconfigurable accelerator for an IR kernel derived from the ITK package, that takes into account data and platform dependent parameters. The contributions are:

- Transforming a software floating point 3D IR kernel to a fixed point multiple bit-width reconfigurable sys-

tem. Optimization on operator bit width is carried out based on an analytic error model. The new design is more cost effective in terms of resource utilization and can be achieved without sacrificing precision and accuracy.

- Classes of application specific cache systems to address the large memory bandwidth requirement. Based on the transformation parameters, different levels of reduction in external memory references can be achieved with different on-chip memory capacities.
- A modularized framework for accelerating 3D IR on reconfigurable platform. Utilizing the reconfigurability of FPGA devices, users can select and load designs optimized for different system environments during the IR process.

The remainder of the paper is organized as follows. In Section 2, the algorithm of 3D IR and the associated challenges of system design are introduced. In Section 3, the computational requirements in the IR algorithm is analyzed and the proposed arithmetic scheme is discussed. In Section 4, the problem of external memory reference is analyzed and a set of caching scheme for different transformation parameters are introduced. In Section 5, the modularize framework for image registration is presented. Implementation and results are shown in Section 6 and followed by the conclusion drawn in Section 7.

2. 3D Image Registration

Inputs to IR process are specially formated 3D medical images which have $(X \times Y \times Z)$ pixels. Common CT and MRI formats are 256^3 or 512^3 pixels in size [7]. A 16-bit integer value is assigned to each pixel to represent its gray scale intensity.

The registration process is to find the optimal transform for aligning two 3D images. In our example, the Rigid Registration kernel (RReg) [1] aligns 3D images through rotations and transitions. After alignment, the similarity of images is measured by the averaged Sum of Square of Difference (SSD) of all interested pixels. Using the SSD value, a monitoring program iterates the RReg kernel for a new set of transformation parameters generated by optimization algorithm. Operations in the RReg kernel include transforming pixel coordinates, fetching reference pixel values, interpolating the expected values of the transformed pixels. Fig. 1 abstracts the process of the RReg kernel.

A homogeneous transformation function, T, is applied to the coordinate in the target image to find the projected coordinate, $(x, y, z) = T(i, j, k)$, in the source image. In the RReg kernel, the T function is simplified to three vectors: $\overrightarrow{xd}$, $\overrightarrow{yd}$ and $\overrightarrow{zd}$. These vectors are added to the current projected coordinate along the corresponding traversed directions to generate the next projected coordinate. Since the transformation parameters are floating point values, the projected coordinate is also represented in floating point format. Thus tri-linear interpolation is used to estimate the gray scale value at the projected position using eight neighboring pixel values, $P1$ to $P8$ and two weight vectors, w_1 and w_2, as shown in Fig. 1. The SSD value is updated by comparing this result to the pixel value in target image.

For medical purposes, only certain areas of the images are need for diagnosis. Thus not all pixels in the target image are involved in the similarity measurement. To avoid unnecessary computation and memory access, a skipping scheme is encoded in the image. For all pixels with a negative value, s, the kernel will skip the following $|s|$ pixels along the X direction in the target image.

The original RReg kernel is implemented in objective oriented C++ code using IEEE754 single precision floating point arithmetics. While this software implementation provides a convenience mean for researchers to test comprehensive registration algorithms, it suffers from long processing time which is common in most software based IR implementations. In general, it takes the ITK software tens of minutes to align two images on PC platform. We focus on improving this kernel since it is in the main loop which iterated several hundred times, e.g. over 300 times in the example.

Analysis of the software kernel indicates that 92% of computation are floating point add and multiply operations which are expensive in reconfigurable hardware in terms of resource consumption and critical path length. We observed the fact that half of the inputs to the tri-linear interpolation tree are 16-bit non-negative integer (the pixel values) and the other half are values within the range of $[0, 1]$ (the weight vectors). This suggests that more efficient dedicated operators can be used instead of the original floating point version. It is also important to maintain correct outputs after applying the dedicated operators. Due to the read-only pixel memory access and independent transformation and interpolation of each pixel, it is possible to distribute workloads to parallel kernels in hardware and deeply pipeline each kernel for maximum throughput rate. The limiting factor in the hardware kernel is the memory bandwidth.

Since the fast on-chip memory in modern FPGA devices are insufficient to store the 32M to 256M bytes 3D images, it is unavoidable to store the pixel data in off-chip memory which is relatively slow and requires multi-cycle access time. The solution is to reduce off-chip memory access using on-chip memory blocks as caches. The memory access pattern in IR process is regular, noncontinuous and highly dependent on the transformation parameters. This suggests that customized caching systems are needed to achieve better performance over traditional CPU caches.

There are two domains to be optimized: (a) parameterized kernels and memory systems with different throughputs, and (b) platform dependent constraints including logic resources and memory bandwidth. The challenges here are to analysis the relationship between the two domains and pro-

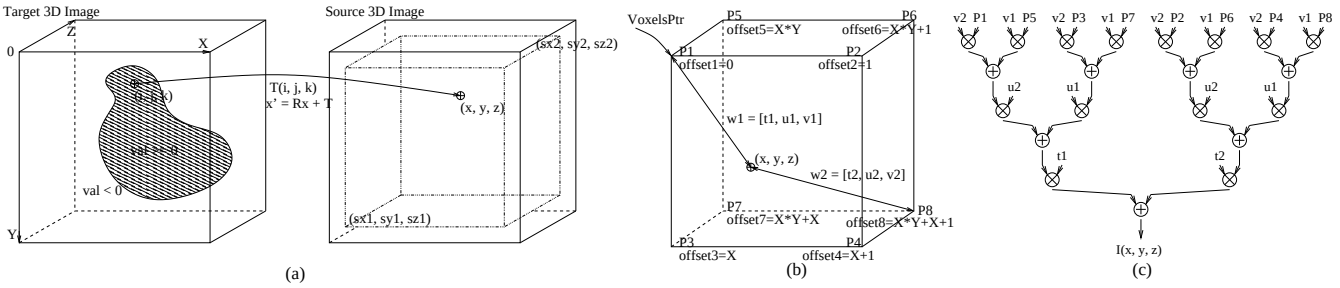

Fig. 1. RReg kernel operations. (a): Coordinate transformation from target image to source image; (b): Finding the weight vectors and values of neighboring pixels; (c): Tri-linear interpolation.

vide a framework to utilize suitable designs under given conditions. So the mapping from various algorithms and problem sizes to various target reconfigurable platforms will have predictable performance and can be automated.

3. Kernel Architecture

One critical component of the hardware IR kernel is the tree structure for interpolation as shown on the right of Fig. 1. It is built by recursively applying the basic sum-of-product structure as shown in Fig. 2. Here, the c_1 and c_2 inputs are components from the weight vectors. The p_1 and p_2 inputs are from pixel values or previous sum-of-product structures with range $[p_{min}, p_{max}]$. h and k are the outputs of the multipliers and r is the output of the adder. The main objective of the accelerator is to improve the performance of these multiply and add operators.

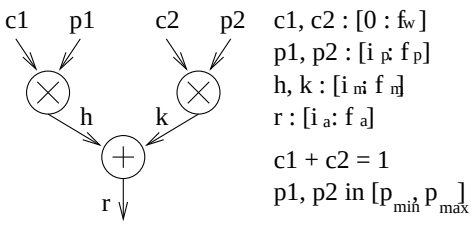

Fig. 2. Basic structure of interpolation tree.

Although fixed point arithmetic has clear advantage on area utilization over the floating point counterpart in reconfigurable logics, it has the weakness of lacking dynamic range. In order to employ fixed point arithmetic in the IR kernel without creating errors, it is necessary to characterize the inputs, the internal tasks performed and the output requirements. In this paper, we use the notation $[i : f]$ for fixed point number format, where i is the number of bits for the integral part and f is the number of bits for fractional part.

The format of pixel values is $[16 : 0]$. The projected coordinate is the form of $([log_2(X) : f], [log_2(Y) : f], [log_2(Z) : f])$. Coordinates outside the X, Y and Z boundaries of source image are discarded. Using this representation, the coordinate of the base pixel $P1$ and the first weight vector $w1$ are simply the integral and fractional parts of the projected coordinate. The second weight vector can be computed using an f-bit fixed point subtraction as $w2 = 1 - w1$.

The value of f affects size and speed of the operators in the interpolation tree. In this design, it is determined by applying precision analysis of Affine Arithmetic (AA) [8] recursively on the tree structure. Let

$$\begin{aligned} E(\widetilde{c_1}) &= 2^{-f_w-1}\varepsilon_1, \qquad E(\widetilde{c_2}) = 2^{-f_w-1}\varepsilon_2, \\ E(\widetilde{p_1}) &= 2^{-f_p-1}\varepsilon_3, \qquad E(\widetilde{p_2}) = 2^{-f_p-1}\varepsilon_4, \\ E(\tilde{h}) &= w_1 E(\widetilde{p_1}) + p_1 E(\widetilde{c_1}) + E(\widetilde{p_1})E(\widetilde{c_1}) \\ &\quad +2^{-f_m-1}\varepsilon_5, \\ E(\tilde{k}) &= w_2 E(\widetilde{p_2}) + p_2 E(\widetilde{c_2}) + E(\widetilde{p_2})E(\widetilde{c_2}) \\ &\quad +2^{-f_m-1}\varepsilon_6, \\ E(\tilde{r}) &= E(\tilde{h}) + E(\tilde{k}) + 2^{-f_a-1}\varepsilon_7 \end{aligned} \tag{1}$$

be the error model of each variable (edges in the tree). Here f_w, f_p, f_m and f_a are the fraction size in bits for the input w, input p, the multipliers and the adder. h and k use the same fraction size due to the symmetric tree structure. $\varepsilon_1..\varepsilon_7 \in [-1, 1]$ are the error source from the round to the nearest process. To preserve accuracy, the required output error, $\hat{E}(\tilde{r})$, should be less than or equal to 1 ulp. That is,

$$\hat{E}(\tilde{r}) \leq 2^{-f_r}. \tag{2}$$

Given the symmetric structure and the fact that $w_1 + w_2 = 1$, the relation between fraction size in every edge of tree structure can be found by substitution of Equation 1, Equation 2 and taking the maximum value of ε to 1 and the maximum value of p to P. Thus

$$\begin{aligned} 2^{-f_a-1} \quad &\geq \quad 2^{-f_p-1} + P2^{-f_w} \\ &\quad +2^{-f_p-f_w-2} + 2^{-f_m}. \end{aligned} \tag{3}$$

This method is applied recursively to the interpolation tree to find the relation between fraction size of every variables in the tree and the final rounding precision. More detail error analysis can be found in [8]. In this application, several attributes of the structure can help to simplify the total error analysis. By partitioning the interpolation tree to three levels, $L1$, $L2$ and $L3$, of sub-trees with basic architecture as shown in Fig. 2, we have:

- *All values from the weight vectors have the same format and* $f_w > max(log_2(X), log_2(Y), log_2(Z))$.
- *All* p *inputs in the first (leaf) level are the exact integer values from the 3D image array. So* $E(p_{L1}) = 0$.
- *All operators on the same level of the interpolation tree have the same input and output format.*
- *In all sub-tree structure,* $w1 + w2 = 1$ *and the two* p *inputs have same range. Thus the output* r *has the same range limit of the* p *inputs.*
- *Final output of interpolation will be rounded to 16-bit integer for the SSD computation. i.e.* $f_{a3} = 0$

Thus the error model in each level can be constructed as

$$\hat{E}(L1) = P2^{-f_w} + 2^{-f_{m1}} + 2^{-f_{a1}-1}, \quad (4)$$

$$\hat{E}(L2) = \hat{E}(L1) + P2^{-f_w} + \hat{E}(L1)2^{-f_w-1} + 2^{-f_{m2}} + 2^{-f_{a2}-1}, \quad (5)$$

$$\hat{E}(L3) = \hat{E}(L2) + P2^{-f_w} + \hat{E}(L2)2^{-f_w-1} + 2^{-f_{m3}} + 2^{-1} \; and \quad (6)$$

$$\hat{E}(L3) \leq 1. \quad (7)$$

Substitution and expansion of the above inequalities show the relationship between system parameters and the precision of the interpolated output. The interpolation tree can be optimized, for more efficient area utilization, by reducing the size of fa, fm and fw respectively. The following characteristics of reconfigurable platforms are considered when evaluating the performance.

- *Multiplier is more expensive than adder when implemented using LUT primitives.*
- *Dedicated multiplier blocks in modern FPGA devices have fixed bit width. Thus the cost of multiplier increases discretely with increasing bit width.*
- *The impact on area improvement by reducing bit width of nodes near the leaves is more significant than that of the nodes near the root of the tree structure.*

Assuming $X = Y = Z = 2^8$, one set of parameters optimized for area and fulfilling the precision requirements in Equation 7 is: $f_w = 22; f_{a1} = 6; f_{m1} = 2; f_{a2} = 7; f_{m2} = 3; f_{m4} = 4$.

4. Memory System

Speeding up memory access is essential in this design since the interpolation process will start only after all the eight pixel values from the source image are ready. Let ρ be the percentage of pixels interested for registration in an image with N pixels. The time for an evaluation process in a fully pipelined architecture is

$$T_{eva} = N \times \rho \times (M_{target} + M_{source})/f, \quad (8)$$

where M_{target} and M_{source} are the number of clock cycles for memory access reading data from target and source images. f is the working frequency of the RReg kernel. The memory access required for skipping uninterested region is ignored here as it contributes less than 1% in T_{eva}. In our analysis, we assume 2^{24} pixels in each image with 95% of them are interested for registration. The memory bottleneck is the result of asymmetric input and output data rate. For single interpolated output value which updates the final SSD, 1 pixel from target image and 8 pixels from source image are required. To prevent the hardware from being idle, the memory bandwidth in random access mode must be at least 9 times of the processing speed which is not realistic in most environments.

In a system with SDRAM, the CAS latency, T_{CL}, dominates the evaluation time. For example, when using single 32-bit channel DDR2 SDRAM for both target and source image, $M_{target} = T_{CL}$ and $M_{source} = T_{CL} \times 4$. Following Equation 8, a DDR2 based design at 200MHz and with $T_{CL} = 3$ can complete an evaluation in 598ms.

Multiple memory channels can be used to improve performance of reconfigurable platform. For example, four independent DDR2 SDRAM banks are associated with a Virtex 5 FPGA in the Alpha Data ADM-XRC-5T2 platform [9]. To utilize this bandwidth, the source image is split into four groups according to the even and odd position in the Y and Z directions. Thus the four edges, $P1P2$, $P3P4$, $P5P6$ and $P7P8$, in the neighboring pixel cube in Fig. 1(c) will reside in the four groups evenly. Mapping the groups to the memory banks in the 5T2 system can reduce the source image access time, i.e. $M_{source} = T_{CL}$. The pixels of target image can be streamed from the host system to the FPGA internal memory as an extra memory channel. A fully pipelined design in 5T2 at 200MHz can complete an evaluation process in 120ms.

The fast internal memory of FPGA can be used as a cache layer to further improve the overall performance. The small cache size in the FPGA is a limiting factor of this improvement. Under the limited cache size, we can improve cache efficiency by capturing application specific information, such as custom cache design for predictable access pattern. In this work, a optimized cache system is constructed to reduce external memory references with increasing on-chip memory requirements.

This cache system is to provide faster access for the neighboring pixel values in the interpolation process. The efficiency of the cache depends on the number of pixels that can be retrieved from it in each interpolation. Since the RReg kernel traverses along the X direction by accumulating the $\overrightarrow{xd}$ vector for over 99.5% of the transformation, we will analyze the cache performance along the X direction as an example. In the following discussion, we present different cache systems which are suitable for transformation vector values in different ranges as shown in Fig. 3.

For $|\overrightarrow{xd}| \leq 0.5$, $1/|\overrightarrow{xd}|$ accumulation steps are required to advance the projected coordinated to the next integral pixel

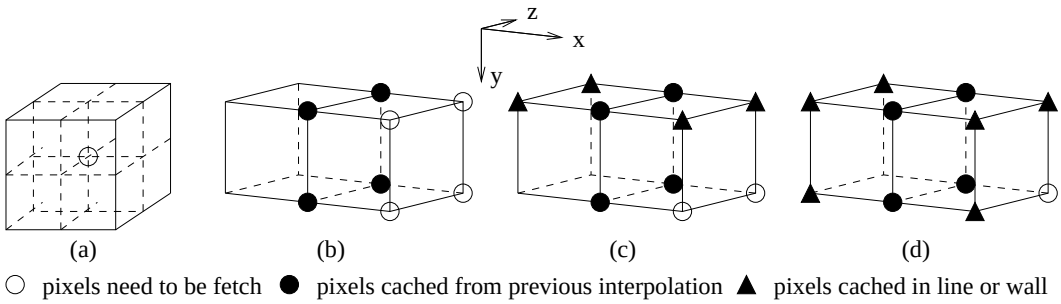

Fig. 3. Dedicated Cache Systems for RReg Kernel: (a) Uncached System; (b) Previous Half Cache; (c) Bottom Line Cache; (d) Side Wall Cache.

cube. Thus a cache system keeping the current eight pixel values will have a miss rate $|\overrightarrow{xd}|$, which is always less than 50%. That means the M_{source} term in Equation 8 is reduced at least by half.

For $0.5 < |\overrightarrow{xd}| < 1$, the probability of advancing to the next integral pixel cube along the X direction after each accumulation is $|\overrightarrow{xd}|$. A cache system is proposed to cache the four pixel values near the next integral pixel cube. As the solid circles shown in Fig. 3(b), $p2$, $p4$, $p6$ and $p8$ in current interpolation are stored and may be used as $p1$, $p3$, $p5$ and $p7$ in the next interpolation process. In cache hit situation, the external memory reference to source image can be reduced by half. Thus the source image memory cycles is now $(1 - |\overrightarrow{xd}|/2) \times M_{source}$. This analysis is also applied when $1 < |\overrightarrow{xd}| < 1.5$ while the cache hit rate is now $2-|\overrightarrow{xd}|$.

The above analysis is based on the assumption that the Y and Z components in $\overrightarrow{xd}$ are close to zero. The actual cache performance will decrease when the Y and Z components increase. Larger caches can further reduce the M_{source} term by caching immediate lines and planes along the Y and Z directions as shown in Fig. 3(c) and Fig. 3(d). These systems are applied when $0.5 < |\overrightarrow{yd}| < 1.5$ and/or $0.5 < |\overrightarrow{zd}| < 1.5$. The cache hit situations in line and wall caches reduce the external memory references in source image to 25% and 12.5%.

For $1.5 < |\overrightarrow{xd}|$, the probability of advancing to next integral pixel cube after each accumulation is $2 - |\overrightarrow{xd}|$. As it is impossible to stay in the previously projected pixel cube, the cache system for $|\overrightarrow{xd}| < 0.5$ is not applicable in this case. The cache systems in Fig. 3 will have less than 50% hit rate and this reduces as $|\overrightarrow{xd}|$ increases. When $2 < |\overrightarrow{xd}|$, no previously fetched pixel values can be reused along the X direction and thus no cache can help reducing the external memory reference.

5. Reconfigurable Framework

Custom ASIC designs usually provide better die size utilization and higher working frequency over FPGA designs. While reconfigurable devices have the ability of dynamically adapting changes in system parameters. In IR applications, the software optimizer constantly adjusts the transformation vectors and sends them to the hardware RReg kernel after each similarity evaluation. From the analysis in Section 4, different types of cache systems should be used according to the values of these vectors.

In the proposed framework, the optimizer program has the ability to detect the changes in system parameters and load the corresponding cache modules into hardware during the optimization process. The transformation vectors are registered and updated by the optimizer after each similarity check and then a suitable caching scheme is selected based on the analysis in Section 4. Only the cache system is reconfigured and the other parts of the RReg kernel stay unchanged. To facilitate this adaptive feature, the RReg kernel is designed in a modular structure with fixed interface between modules. The regions of partial reconfigurable modules (RMs) and the locations of interface bus macros are defined by the Xilinx PlanAhead tool. We also implement all the cache systems as RMs within the defined region. Finally, the iMPACT programming tool, which can identify a partial bitstream, is called as an external program to reconfigure the FPGA as needed [10].

Time for hardware reconfigurations occurred between evaluation processes is considered as overhead. The Virtex 5 device has lower reconfiguration latency than earlier generations of FPGA devices and less constrains on the shape and location of the RMs [11]. Reconfiguring less than 5% of the XC5VLX330T device, the overhead time, T_{cfg}, is less than 10ms.

This overhead can be justified by the reduction of external memory references in successive evaluation processes as shown below.

$$T_{cfg} \leq E_{successive} \times M_{reduced}/f, \tag{9}$$

where $E_{successive}$ is the number of successive evaluation before next reconfiguration, $M_{reduced}$ is the averaged number of external memory references reduced in each evaluation and f the working frequency of RReg kernel. For example, a 200MHz kernel reconfigured to have 50% external memory reduction will save over 36ms in a single evaluation iteration.

Table 1. Results of different cores.

	floating point	fixed point
Bit width	32 (IEEE754)	variable
LUTs	11711 (5.65%)	4146 (2.00%)
FFs	13484 (6.50 %)	5634 (2.72%)
Max. Frequency (MHz)	398	412

Thus the T_{cfg} overhead is negligible in this case.

6. Implementation and Results

A fully pipelined interpolation core optimized for fixed point arithmetic has been implemented on a Xilinx XC5VLX330T-FFG1738 FPGA. The cores are described in VHDL, synthesized using Synplciity Synplify 9.0 and placed and routed using Xilinx ISE 10.1i tool chain. The timing information is extracted from the associated static timing analysis tool. The performance of implementations using single precision floating point and optimized fixed point (using the parameters at the end of Section 3) operators are shown in Table 1.

The results show that using optimized fixed point operators can speed up the tri-linear core and reduce the area by 2.83 times compared with the equivalent floating point implementation while achieving the same accuracy. As a single core occupies less than 3% of the area of an FPGA, we can easily fit 30 independent cores on a single device and each core can operate on different transformation vectors.

The available memory bandwidth limits the performance of the accelerated core. The accelerated RReg kernel is implemented on the 5T2 [9] platform with four banks of DDR2 SDRAM. Splitting and redistributing the source image into these memory banks enable the design to work at 200MHz with high throughput.

Running on a 2.5GHz Quad-Core Intel Xeon CPU, an optimized (GCC 4.0 with `-O3`) multi-threaded software version of the RReg kernel can complete a similarity evaluation in 126ms for a pair of 256^3 images with 90% interested pixels. The same result can be achieved by the 200MHz hardware RReg kernel in 72ms. The communication overhead between the software optimizer and the RReg kernel include three transformation vectors and one SSD value which is negligible. Without the external memory restrictions in the 5T2 platform, the evaluation can be completed in 35ms at 412MHz. With 30 parallel cores on chip, we can evaluate 30 different transformations concurrently, resulting in a throughput of 1.168ms per evaluation. This is 108 times faster than software running on a 2.5GHz Quad-Core Xeon processor.

7. Conclusion

This paper presents a reconfigurable framework for accelerating registration algorithms for 3D medical images. The hardware design involves different modular components such as weighted vector estimation, tri-linear interpolation and a cache system. By analytical error modeling, the tri-linear interpolation process is optimized to improve the clock rate and reduce the circuit area without making accuracy unacceptable. Loading the different modules according to system parameters can be achieved through the proposed reconfigurable framework. Our experiments show that such design can reduce the area and improve the speed compared with a straight forward floating point implementation. Future work includes applying the analysis to advanced medical IR algorithms and support of arbitrary size images.

8. References

[1] D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. Leach, and D. J. Hawkes, "Non-rigid registration using free-form deformations: Application to breast MR images," *IEEE Transactions on Medical Imaging*, vol. 18, no. 8, pp. 712–721, 1999.

[2] J. Jun, W. Luk, and D. Rueckert, "FPGA-based computation of free-form deformations in medical image registration," *Proc. International Conference on Field Programmable Technology (FPT)*, pp. 234–241, 2003.

[3] Altera Corporation, "Medical imaging implementation using FPGAs," White Paper, Altera Corporation, Apr. 2006.

[4] M. Sen, Y. Hemaraj, S. S. Bhattacharyya, and R. Shekhar, "Reconfigurable image registration on FPGA platforms," in *Proc. IEEE Biomedical Circuits and Systems (BioCAS'06)*, Nov. 2006, pp. 154–157.

[5] J. J. Koo, A. C. Evans, and W. J. Gross, "Accelerating a medical 3D brain MRI analysis algorithm using a high-performance reconfigurable computer," in *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, 2007, pp. 11–16.

[6] O. Dandekar, W. Plishker, S. Bhattacharyya, and R. Shekhar, "Multiobjective optimization of FPGA-based medical image registration," in *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2008.

[7] J. V. Hajnal, D. J. Hawkes, and D. L. G. Hill, *Medical Image Registration*. CRC Press, 2001.

[8] D. Lee *et al.*, "Accuracy-guaranteed bit-width optimization," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 10, pp. 1990–2000, Oct. 2006.

[9] *ADM-XRC-5T2 Hardware Manual*, Alpha Data Ltd., 2008.

[10] *Difference-Based Partial Reconfiguration*, Xilinx, Inc., 2007, version 2.0.

[11] Xilinx University Program, "Partial reconfiguration professor workshop," Xilinx, 2007.