

FPGA-based acceleration of mutual information calculation for real-time 3D image registration

Carlos R. Castro-Pareja^{a,b}, Jogikal M. Jagadeesh^a, Raj Shekhar^{*b}

^aDept. of Electrical Eng., The Ohio State Univ., 2015 Neil Ave., Columbus, OH, USA 43210-1272;

^bDept. of Biomedical Eng., Lerner Research Institute, The Cleveland Clinic Foundation, 9500 Euclid Ave, Cleveland, OH, USA 44195

ABSTRACT

Real-time image registration is potentially an enabling technology for the effective and efficient use of many image-guided diagnostic and treatment procedures relying on multimodality image fusion or serial image comparison. Mutual information is currently the best known image similarity measure for multimodality image registration. Mutual information calculation is a memory-intensive task that does not benefit from cache-based memory architecture in standard software implementations (i.e., the calculation incurs a large number of cache misses). Previous attempts to perform image registration in real time focused on parallel supercomputer implementations, which achieved real-time performance using large, expensive supercomputers that are simply impractical for deployment in a hospital. We have developed a custom hardware architecture that, in a single-module PC-based implementation, achieves registration speeds comparable to those of a 64-processor parallel supercomputer. The single-module speedup results from using parallel memory access and parallel calculation pipelines. The total speedup can be increased by using several modules in parallel. The architecture is designed for linear, mutual information-based registration and can be extended to elastic (nonlinear) registration.

Keywords: Image registration, mutual information, real-time systems

1. INTRODUCTION

Image registration is the process of aligning two images that represent the same anatomy from different points of view,

at different times or using a different imaging modality. A recent survey on image registration was presented by Zitova and Flusser¹. Image registration is an important tool in medical imaging, where it is used to merge or compare images obtained from a variety of modalities, such as magnetic resonance imaging (MRI), computed tomography (CT), positron emission tomography (PET), single photon emission computed tomography (SPECT) and ultrasound. In the medical field, real-time image registration is an enabling technology for the effective and efficient use of many diagnostic and image-guided treatment procedures relying on multimodality image fusion or serial image comparison².

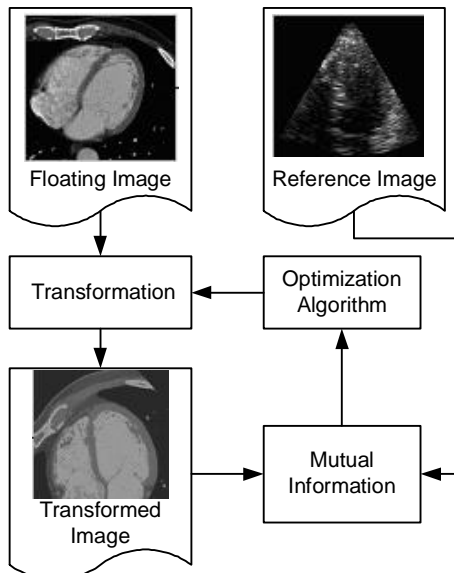


Fig. 1. Mutual-information-based registration flowchart

Mutual information is currently the best-known image similarity measure for multimodality image registration³. Its use for image registration was first introduced by Wells *et al.*⁴. Pluim presented a comprehensive survey of the literature regarding mutual-information-based registration⁵. The goal of mutual-information-based registration is to find the transformation that best aligns two images by maximizing the mutual information between them. Figure 1 describes the mutual-information-based registration process. The optimization algorithm is used to find the set of transformation parameters that

*shekhar@bme.ri.ccf.org; phone 1 216 445 3246; fax 1 216 444 9198

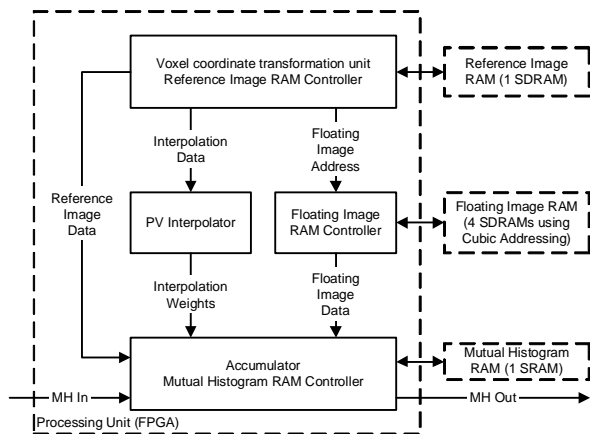


Fig. 2. Block diagram of the FAIR architecture

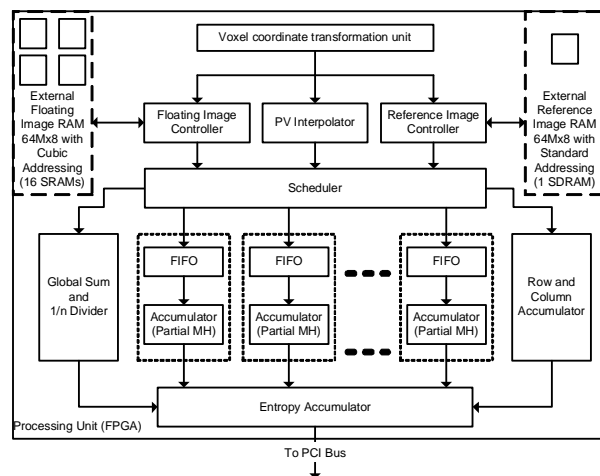


Fig. 3. Block diagram of the FAIR2 architecture

maximizes the mutual information between the reference image and the transformed floating image. Depending on the particular application, the transformation can be rigid, affine or elastic. The work presented in this paper was focused on accelerating mutual-information-based registration using affine (linear) image transformations, and is currently being enhanced to support elastic transformations.

Mutual information is defined as:

$$MI(RI, FI) = H(RI) + H(FI) - H(RI, FI) \quad (1.1)$$

where RI is the reference image, FI is the floating image (the one being transformed), $H(RI)$ and $H(FI)$ are the individual image entropies, and $H(RI, FI)$ is their joint entropy. The entropies are computed as follows:

$$H(RI) = -\sum_a p_{RI}(a) \ln p_{RI}(a) \quad (1.2)$$

$$H(FI) = -\sum_b p_{FI}(b) \ln p_{FI}(b) \quad (1.3)$$

$$H(RI, FI) = -\sum_{a,b} p_{RI,FI}(a,b) \ln p_{RI,FI}(a,b) \quad (1.4)$$

The joint voxel intensity probability $p_{RI,FI}(a,b)$, i.e., the probability of a voxel in the reference image having an intensity a given that the corresponding voxel in the floating image has an intensity b , can be obtained from the joint or mutual histogram of the two images. The mutual histogram represents the joint intensity probability distribution. In the process of mutual information-based registration, the dispersion of values within the mutual histogram is minimized, which in turn minimizes the joint entropy and maximizes the mutual information. The process of mutual information calculation involves first obtaining the individual and mutual histograms of the two images and then calculating their individual and joint entropies using the histograms' values. Mutual histogram calculation is a memory-intensive task that does not benefit from cache-based memory architecture (i.e., it incurs a large number of cache misses) in standard software implementations⁶.

Previous attempts to perform image registration in real time focused on parallel supercomputer implementations⁷, which achieved real-time performance using large, expensive supercomputers that are simply impractical for deployment in a hospital. In this paper we present a second-generation architecture for acceleration of mutual information calculation, based on our Fast Automatic Image Registration (FAIR) architecture⁶. Our latest architecture, called FAIR2, achieves registration speeds comparable to those of a 64-processor parallel supercomputer in a single-module PC-based implementation. The single-module speedup results from using parallel memory access and parallel calculation pipelines. The total speedup can be increased by using several modules in parallel. The architecture is designed for linear, mutual information-based registration and can be extended to support elastic (nonlinear) registration.

Characteristics	FAIR ⁶	FAIR2
Mutual histogram calculation	Yes	Yes
Individual histogram calculation	No	Yes
Entropy calculation	No	Yes
Reference Image RAM	SDRAM using burst accesses	SDRAM using burst accesses
Floating Image RAM	SDRAM using cubic addressing	SRAM using cubic addressing
Mutual histogram RAM characteristics	Single-port external SRAM	Dual-port internal SRAM
Mutual histogram RAM partitioning	No	Yes

Table 1: Comparison between FAIR and FAIR2 features

2. ARCHITECTURE

Mutual information calculation can be divided into two steps: in the first step the mutual and individual histograms are formed. In the second step, the entropies are calculated and added, using the values of the mutual histogram to obtain the individual and joint voxel intensity probabilities. The time required to perform the first step is proportional to the image size, while the time required to perform the second step is proportional to the mutual histogram size. Since typical 3D image sizes are between 2^{20} and 2^{27} voxels, and typical mutual histogram sizes range between 2^{10} and 2^{16} bins, mutual histogram calculation speed will normally be the most important factor in mutual information calculation time. It has been shown that, for large images, mutual histogram calculation can take 99.9% of the total mutual information calculation time⁶. The FAIR architecture accelerated mutual information-based image registration by accelerating mutual histogram calculation⁶. Its block diagram is shown in Figure 2. It supported parallel, independent external busses for the reference image memory, the floating image memory and the mutual histogram memory. The floating image was accessed using cubic addressing to allow retrieval of a full $2 \times 2 \times 2$ voxel neighborhood in one access time. Our prototype implementation of the FAIR architecture achieved speedups of approximately 8 per module against a 1-GHz Pentium III computer when performing affine registration. The implementation was realized using FPGAs with an 80-MHz clock rate. Higher speedups were possible by using several modules in parallel, but the theoretical maximum speedup was limited by the mutual histogram transmission time. The second-generation architecture we present in this paper, FAIR2, overcomes that limitation by performing full mutual information calculation on-chip and achieves a higher single-module speedup by providing a higher memory bandwidth. Table 1 shows a comparison between the features of the two architectures.

A block diagram of the FAIR2 architecture is shown in Figure 3. As mentioned before, mutual information calculation is performed in two steps: mutual and individual histogram calculation, and entropy accumulation. These steps are described in detail in the following subsections.

2.1 Mutual histogram computation

The first step in mutual information calculation involves applying the current estimate of the inverse transformation to each voxel coordinate of the reference image, to find the corresponding voxel coordinates in the floating image, and updating the mutual histogram at the location dictated by the intensities of the voxel pair. The corresponding coordinates do not normally coincide with a grid point in the floating image, thus necessitating interpolation. Partial volume (PV) interpolation, as defined by Maes *et al.*⁸ was chosen because it provides smooth changes in the histogram values as a result of small changes in the transformation⁵. The algorithm for mutual histogram computation is shown in Figure 4. Step (a) of the algorithm is performed by the voxel coordinate transformation unit. Steps (b) and (c) are performed in parallel: The reference image coordinate values are passed to the reference image controller, the integer components of the floating image coordinates are passed to the floating image controller, and its floating components to the PV interpolator.

For each voxel in the reference image,

- a. Calculate corresponding floating image coordinates (apply affine transformation)*
- b. Load corresponding floating image 2x2x2 voxel neighborhood (8 intensity values are loaded in parallel)*
- c. Calculate the 8 partial volume interpolation weights (similar to trilinear interpolation, but weights are not added at the end)*
- d. Accumulate interpolation weights into corresponding mutual histogram bins.*

Fig. 4. Mutual histogram computation algorithm

In steps (b) and (c), the floating image controller accesses the corresponding 2x2x2 voxel neighborhood in parallel through a memory bus implemented using cubic addressing⁹. At the same time, the reference image controller loads the corresponding reference image voxel intensity value. The reference image RAM is accessed sequentially, using burst accesses. Meanwhile, the PV interpolator unit calculates the interpolation weights. The FAIR architecture used SDRAMs to store both the images. The main bottleneck in image memory access is the floating image, which is not accessed sequentially and thus does not benefit from burst accesses. In order to accelerate image access, FAIR2 uses SRAMs to store the floating image, thus allowing one 2x2x2 neighborhood access per external clock cycle.

Step (d) is performed by the scheduler and the FIFO-accumulator pipelines. Step (d) consists of accumulating the 8 interpolation weights calculated by the PV interpolator into their corresponding mutual histogram bins, dictated by their corresponding reference and floating image voxel intensity values. For a given reference image voxel, there are 8 voxel intensity pairs, each with its own interpolation weight. Since the mutual histogram memory needs to be read and written once for each intensity pair, 16 accesses to the mutual histogram memory are required per reference image voxel. To provide 16 accesses in the time of a single image memory access, we have applied three different techniques. First, we implemented the mutual histogram using internal FPGA memory, running at four times the image memory access frequency. Second, we used dual-ported memories to be able to perform both a read and a write access simultaneously. Third, the mutual histogram memory was partitioned to allow a higher number of accesses to occur at the same time. The mutual histogram memory is arranged such that the address of a mutual histogram bin is obtained by concatenating the corresponding reference image and floating image voxel intensity values. Therefore, the least significant bits (LSBs) of the mutual histogram memory addresses correspond to the LSBs of the floating image voxel intensity values. FAIR2 supports partitioning the mutual histogram memory into 2^n blocks, according to the n LSBs of the memory addresses. This partitioning scheme maximizes the likelihood that different floating image voxel intensity values in the same neighborhood will be assigned to different mutual histogram blocks, thus allowing a higher degree of parallelism in mutual histogram RAM accesses. However, using this partitioning scheme alone will fail in the presence of large image areas of uniform intensity. To prevent this problem, the scheduler groups equivalent voxel intensity pairs and adds their corresponding interpolation weights together before sending them to the accumulators. This operation also prevents read-after-write hazards in the accumulation pipeline. FIFOs are used as buffers to handle traffic spikes to individual mutual histogram blocks.

While the mutual histogram is computed, the row and column accumulator unit computes the individual histograms. The architecture of the row and column accumulator is similar to the mutual histogram accumulation pipeline. The global sum module keeps track of the number of voxels accumulated into the histograms and calculates its inverse (i.e. it applies the $1/n$ function to it). The resulting value is used in the second step to calculate the voxel intensity probabilities from the individual and joint histogram values.

2.2 Entropy accumulation

In the second step of mutual information calculation, the accumulator modules send the partial mutual histogram values to the entropy accumulator. In the entropy accumulator, the individual and joint voxel intensity probabilities are calculated from the mutual histogram values, and the $p \cdot \ln(p)$ function is applied to them. The results are then accumulated, thus obtaining the mutual information value.

To calculate mutual information, it is necessary to evaluate the function $f(p) = p \cdot \ln(p)$ for all the individual and joint intensity probabilities. Since the probabilities range from 0 to 1, the $\ln(p)$ function will have a range of $[-\infty, 0]$, with the problem that it is undefined for $p = 0$. Fortunately, the $p \cdot \ln(p)$ function has a much more limited range of $[-e^{-1}, 0]$ and

is defined in the full range of $0 \leq p \leq 1$. Hardware-based logarithm calculation is usually performed using series approximations or a combination of series approximations and look-up tables (LUTs)¹⁰⁻¹⁴. Using series approximation requires implementing a series of arithmetic units that consume significant hardware resources, and take several clock cycles to finish calculation. It is important to minimize resource usage in an FPGA implementation to optimize physical space and to improve speed. Existing LUT-based approaches for logarithm calculation have a limited accuracy (up to 24 fixed-point bits) that makes them unsuitable for the current application^{13,14}. In this subsection we present an algorithm for the calculation of $p \cdot \ln(p)$ that allows simple, LUT-based implementation in FPGAs and has a small enough error range acceptable for its application in mutual information calculation.

2.2.1 LUT-based entropy calculation

The function $f(p)$ is approximated in the range $[0, 1]$ by using the piecewise-polynomial function $\hat{f}_{N,m}(p)$ with m segments, defined in (2.8) below.

$$\hat{f}_{N,m}(p) = \begin{cases} P_0(p) & \text{for } 0 \leq p < \Delta p \\ P_1(p \bmod \Delta p) & \text{for } \Delta p \leq p < 2\Delta p \\ \vdots & \vdots \\ P_i(p \bmod \Delta p) & \text{for } i \cdot \Delta p \leq p < (i+1) \cdot \Delta p \\ \vdots & \vdots \\ P_{m-1}(p \bmod \Delta p) & \text{for } (m-1) \cdot \Delta p \leq p < 1 \end{cases} \quad (2.8)$$

where $\Delta p = 1/m$ is the segment size of $\hat{f}_{N,m}(p)$ and P_i is a polynomial of order $N-1$. To minimize the maximum approximation error, each P_i is obtained by calculating the Chebyshev approximation for $f(p)$, defined in (2.9) for $i \cdot \Delta p \leq p < (i+1) \cdot \Delta p$ ¹⁵. The Chebyshev approximation is simple to calculate for continuous functions and has the advantage that it is very close to the minimax approximation, the most accurate polynomial approximation¹⁶. Equation (2.10) is used to calculate the coefficients.

$$P_i(p \bmod \Delta p) \approx \left[\sum_{k=0}^{N-1} c_{i,k} T_k \left(\frac{2(p \bmod \Delta p) - 1}{\Delta p} \right) \right] - \frac{1}{2} c_{i,0} \quad (2.9)$$

$$c_{i,k} = \frac{2}{N} \sum_{l=1}^N f \left[\frac{\Delta p}{2} \cos \left(\frac{\pi(l-1/2)}{N} \right) + \Delta p \cdot (i+1/2) \right] \cos \left(\frac{\pi k(l-1/2)}{N} \right) \quad (2.10)$$

The Chebyshev polynomials are defined by $T_n(x) = \cos(n \cdot \arccos(x))$, for $-1 \leq x \leq 1$. The Chebyshev polynomials of order 0 to 3 are shown in (2.11). Since each polynomial is used to approximate $f(p)$ in a specific $[i \cdot \Delta p, (i+1) \cdot \Delta p]$ range, whereas the Chebyshev polynomials are defined in $[-1, 1]$, the variable conversion shown in (2.12) was applied to the equations.

$$T_0(x) = 1, T_1(x) = x, T_2(x) = 2x^2 - 1, T_3(x) = 4x^3 - 3x \quad (2.11)$$

$$x = (2(p \bmod \Delta p) - \Delta p) / \Delta p \quad (2.12)$$

To keep our arithmetic pipeline simple, we considered only 1st, 2nd and 3rd-order approximations for our architecture. Equation (2.13) defines the i th polynomial component of $\hat{f}_N(p)$:

$$P_i(p_d) = k_{i,3} \cdot p_d^3 + k_{i,2} \cdot p_d^2 + k_{i,1} \cdot p_d + k_{i,0}, \quad (2.13)$$

where $p_d = p \bmod \Delta p$. The polynomial coefficients $k_{i,j}$ are stored in the i th entry of the LUT. They are calculated from the Chebyshev coefficients as shown in equations (2.14)-(2.17), which are derived from (2.9), (2.11) and (2.12):

$$k_{i,0} = 0.5 \cdot c_{i,0} - c_{i,1} + c_{i,2} - c_{i,3} \quad (2.14)$$

$$k_{i,1} = (c_{i,1} - 4 \cdot c_{i,2} + 9 \cdot c_{i,3}) \cdot (2/\Delta p) \quad (2.15)$$

$$k_{i,2} = (2 \cdot c_{i,2} - 12 \cdot c_{i,3}) \cdot (2/\Delta p)^2 \quad (2.16)$$

$$k_{i,3} = 4 \cdot c_{i,3} \cdot (2/\Delta p)^3 \quad (2.17)$$

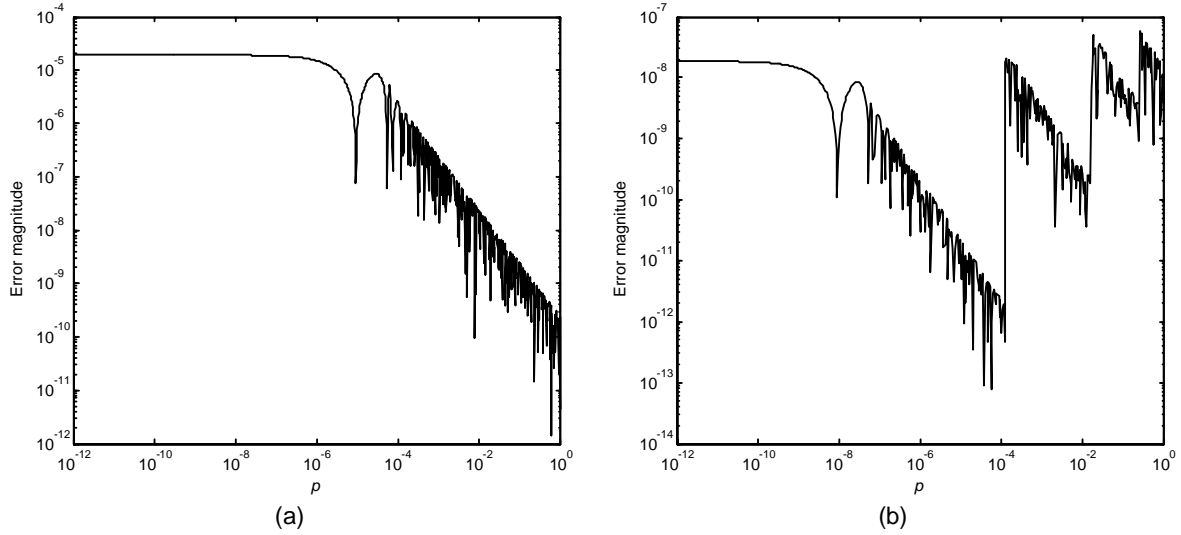


Fig. 5. (a) Error magnitude plot for a single LUT with 2^{14} entries. (b) Error magnitude plot for 4 LUTs with 2^{11} entries each.

2.2.2 Error characteristics

The approximation error in the i th segment of $\hat{f}_{N,m}(p)$ is given by:

$$\varepsilon_i(p) = \hat{f}_{N,m}(p) - f(p) = P_i(p \bmod \Delta p) - f(p) \quad (2.18)$$

The position p_{maxi} of the maximum absolute error e_i in this segment is obtained by evaluating p when the derivative of the error is equal to zero:

$$d\varepsilon_i/dp = \dot{P}_i(p_{maxi} \bmod \Delta p) - \dot{f}(p_{maxi}) = 0 \quad (2.19)$$

$$\Rightarrow 3k_{i,3} \cdot (p_{maxi} \bmod \Delta p)^2 + 2k_{i,2} \cdot (p_{maxi} \bmod \Delta p) + k_{i,1} = 1 + \ln(p_{maxi}) \quad (2.20)$$

$$e_i = \max_{i \cdot \Delta p \leq p < (i+1) \cdot \Delta p} \varepsilon_i(p) = \varepsilon_i(p_{maxi}) \quad (2.21)$$

Since using an N th-order piecewise-polynomial approximation assumes that the N th derivative of $f(p)$ is constant in each segment of $\hat{f}_{N,m}(p)$, the segment size Δp must be determined by the rate of change of the N th derivative and the desired maximum error. The first and second derivatives of $f(p)$ are $\dot{f}(p) = 1 + \ln(p)$ and $\ddot{f}(p) = 1/p$. The rate of change of all derivatives approaches infinity as p nears zero and decreases steadily as p increases. Therefore, the approximation error is higher for smaller values of p , which further implies that this error is largest in the first segment of $\hat{f}_{N,m}(p)$, as shown in (2.22).

$$e_{max} = e_0 = \varepsilon_0(p_{max0}) \quad (2.22)$$

For a given Δp , using a higher-order polynomial approximation yields a lower maximum error. Therefore, for a given number of LUT entries, e_{max} will decrease as the polynomial order increases. To be able to store the LUT using internal memory, a reasonable maximum number of entries is 16K. For a LUT with 16K entries, the maximum error is in the order of 10^{-5} to 10^{-6} for 1st to 3rd-order approximations.

2.2.3 Multi-LUT approach

From section 2.2.2 we know that the maximum error will occur in the first segment of $\hat{f}_{N,m}(p)$. Figure 5 (a) shows the error magnitude plot for an LUT with 2^{14} bins, using 3rd-order polynomials and double-precision floating-point numbers. As predicted in section 2.2.2, the error decreases as p increases.

LUT Number	Number of Entries (m)	LUT Range		
		p_{min}	p_{max}	Δp
1	2048	0	2^{-13}	2^{-24}
2	2048	2^{-13}	2^{-6}	2^{-17}
3	2048	2^{-6}	2^{-2}	2^{-13}
4	2048	2^{-2}	2^0	2^{-11}

Table 2: Characteristics of a 4- LUT implementation of the entropy calculation pipeline.

Implementation Method	Image Size (voxels)		
	2^{18}	2^{21}	2^{24}
Software (PIII, 1 GHz)	430	3500	28000
FAIR (1 Unit) ⁶	62	430	3370
FAIR (2 Units) ⁶	36	220	1700
FAIR2 (1 Unit)	5	42	335

Table 3: Comparison of mutual information calculation times in milliseconds.

To further reduce LUT size and improve accuracy, we used a multi-LUT approach, where each LUT had a different Δp value and was based on a different approximation function $\hat{f}_{N,m_i}(p)$. The general n -LUT form of this approximation is shown in (2.23), where $\Delta p_1 < \Delta p_2 < \dots < \Delta p_n$. Having larger values of Δp for higher values of p is equivalent to approximating the function at different resolutions, which allows us to reduce the LUT size while keeping the approximation error below the allowable maximum.

$$\hat{F}_N(p) = \begin{cases} \hat{f}_{N,m_1}(p) & \text{for } 0 \leq p < p_{max_1} \\ \hat{f}_{N,m_2}(p) & \text{for } p_{max_1} \leq p < p_{max_2} \\ \vdots & \vdots \\ \hat{f}_{N,m_n}(p) & \text{for } p_{max_{n-1}} \leq p < 1 \end{cases} \quad (2.23)$$

Using this approach we designed the 1st-order polynomial, 4-LUT configuration shown in Table 1. The maximum error was in the order of 10^{-8} for a total of 8K entries. Figure 5 (b) shows a plot of the error magnitude vs. p for this configuration. Using lower-order polynomials has the advantage of reducing the LUT data width and the number of arithmetic components in the pipeline.

3. IMPLEMENTATION AND RESULTS

The system was implemented as a proof of concept using an Altera Stratix EP1S40 FPGA in a PCI prototyping board manufactured by SBS Inc.¹⁷. The FPGA ran at a maximum internal frequency of 200 MHz, with 100 MHz memory busses. Mutual histogram RAM was partitioned into 8 blocks, and implemented using internal 4 Kbit memory blocks. Reference and floating image memory were implemented using standard PC100 SDRAMs and high-speed SRAMs, respectively. Entropy calculation was implemented using the 4-LUT, 1st-order polynomial configuration shown in Table 2. As shown before, for a given LUT, the error magnitude decreases as p increases. All LUTs were implemented inside one internal 512 Kbit memory block. Mutual information was calculated using 32-bit fixed point numbers.

As implemented, the system was able to process 50 million reference image voxels per second. Table 3 shows timing results and compares them against the corresponding results for the FAIR architecture⁶, and single-processor software implementations. The reason for the higher speedup with smaller images is the on-chip entropy calculation, which eliminates the need to transmit the mutual histogram back to the host computer.

4. CONCLUSIONS

We have presented our second-generation architecture, called FAIR2, to accelerate mutual information calculation that achieves speedup rates an order of magnitude higher than those achieved by the first-generation FAIR architecture⁶. The increase in speedup is a result of performing full mutual information calculation on-chip, using faster FPGAs, and storing and partitioning the mutual histogram in internal memory. This architecture is currently being enhanced to support elastic registration algorithms.

REFERENCES

1. B. Zitova, J. Flusser, "Image registration methods: a survey," *Image and Vision Computing*, **21:11**, 977-1000, 2003.
2. R. Shekhar, V. Zagrodsky, C. R. Castro-Pareja, V. Walimbe, J. M. Jagadeesh, "High-speed Registration of Three- and Four-dimensional Medical Images by Using Voxel Similarity," *RadioGraphics*, **23:6**, 1673-1681, 2003.
3. M. Holden, et al., "Voxel similarity measures for 3-D serial MR brain image registration", *IEEE Trans. on Medical Imaging*, **19:2**, 94-102, 2000.
4. W. M. Wells, P. Viola, H. Atsumi, S. Nakajima, and R. Kikinis, "Multi-modal volume registration by maximization of mutual information," *Medical Image Analysis*, **1:1**, 35-51, 1996.
5. J. P. W. Pluim, J. B. A. Maintz, M. A. Viergever, "Mutual-information-based registration of medical images: a survey," *IEEE Trans. on Medical Imaging*, **22:8**, 986-1004, 2003.
6. C. R. Castro-Pareja, J. M. Jagadeesh, R. Shekhar, "FAIR: A hardware architecture for real-time 3-D image registration," *IEEE Trans. on Information Technology in Biomedicine*, **7:4**, 2003 (in print).
7. T. Rohlfing, C. R. Maurer, "Non-rigid image registration in shared-memory multiprocessor environments with application to brains, breasts, and bees," *IEEE Trans. on Information Technology in Biomedicine*, **7:1**, 16-25, 2003.
8. F. Maes *et al.*, "Multimodality image registration by maximization of mutual information," *IEEE Trans. on Medical Imaging*, **16:2**, 187-198, 1997.
9. M. Doggett and M. Meißner, "A memory addressing and access design for real time volume rendering", *Proceedings of the 1999 IEEE International Symposium on Circuits and Systems, ISCAS '99*, **4**, 344-347, 1999.
10. D. K. Kostopoulos, "An algorithm for the computation of binary logarithms," *IEEE Trans. on Computers*, **40:11**, 1267-1270, 1991.
11. D. M. Mandelbaum and S. G. Mandelbaum, "A fast, efficient parallel-acting method of generating functions defined by power series, including logarithm, exponential, and sine, cosine," *IEEE Trans. on Parallel and Distributed Systems*, **7:1**, 33-45, 1996
12. J. Hormigo, J. Villalba and M. J. Schulte, "A hardware algorithm for variable-precision logarithm," *Proc. of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, 215-224, 2000.
13. H. Hassler and N. Takagi, "Function evaluation by table look-up and addition," *Proc. of the 12th Symposium on Computer Arithmetic*, 10-16, 1995.
14. S. L. SanGregory, C. Brothers, D. Gallagher and R. Siferd, "A Fast, Low-Power Logarithm Approximation with CMOS VLSI Implementation," *42nd Midwest Symposium on Circuits and Systems*, **1**, 388-391, 1999.
15. W. H. Press et al., *Numerical Recipes in C: The Art of Scientific Computing*, ch. 5, Cambridge University Press, Cambridge, 1992.
16. M. J. D. Powell, *Approximation Theory and Methods*, ch. 7-8, Cambridge University Press, Cambridge, 1981.
17. *Tsunami PCI FPGA Processor Technical Data Sheet*, Release version 1.2, SBS Technologies, Waterloo, Ontario, Canada, 2003.