FPGA-based computation of Free-Form Deformations in Medical Image Registration

Jun Jiang, Wayne Luk and Daniel Rueckert
Department of Computing
Imperial College
180 Queen's Gate
London SW7 2BZ, England

Abstract

This paper describes techniques for producing FPGA-based designs that support free-form deformation in medical image processing. The free-form deformation method is based on a B-spline algorithm for modelling three-dimensional deformable objects. Our design includes four optimisations. First, we transform a nested loop to eliminate conditional statements. Second, we adopt a customised number representation format in our implementation. Third, we store the values of a third-order B-spline model in lookup tables. Fourth, we pipeline the design to increase its throughput, and we also deploy multiple pipelines such that each covers a different subimage. Our design description, captured in the Handel-C language, is parameterisable at compile time to support a range of image resolutions and computational precisions. An implementation on a Xilinx XC2V6000 device at 67 MHz can run 3.2 times faster than an Intel Xeon-based PC at 2666 MHz.

1 Introduction

This paper describes techniques for producing FPGA-based designs that support free-form deformations (FFDs) in medical image processing. Free-form deformations, which are based on B-splines, are a powerful tool for modelling three-dimensional deformable objects.

Image registration algorithm has been applied in areas such as remote sensing and three-dimensional computer vision. In medical applications, such as the detection of cancerous lesions in contrast-enhanced breast Magnetic Resonance Imaging (MRI), the free-form deformation model is adopted as an important part of non-rigid registration, a method for analyzing deformable objects [13]. However, there is one disadvantage of this image registration implementation which adopts free-form deformation as the local motion model: the processing time of a three-dimensional image with a resolution of 256 by 256 by 64 voxels takes between 15-30 minutes of processor time on a Sun Ultra 10 workstation.

In this paper we present the use of reconfigurable hardware based on FPGAs to compute free-form deformation. Our design approach has four innovations: (1) A data transformation is developed for a nested loop to eliminate the conditional statements. (2) A customised data format is adopted in our implementation. (3) The values of the third-order basis function of the B-spline are precalculated and stored in lookup tables. (4) A pipelined design has been developed and multiple pipelines have also been deployed which can perform free-form deformations in real time for a two-dimensional image with a resolution up to 256 by 256 pixels. This design is parameterisable at compile time for different image resolutions.

Other researchers have explored the use of hardware in medical computations [5], [18]. Previously, we have presented a method for eliminating conditional statements in a nested loop for FFD computation by narrowing the range of the input [9]. Another method has been presented to achieve the same effect by transforming input data [10]. This method has the advantage that all data can be processed by the hardware compared to our previous method. In this paper, we investigate the possibility of using fixed-point number representation in our design and comparing it with floating-point format in terms of clock speed and area cost.

2 B-spline based FFD

Cubic B-splines are widely used in interpolation applications [6] and graphics, such as attenuation map reconstruction [2], pre-surgical planning in plastic surgery [7], filtered back-projection [8] and high-quality image rotation [3].

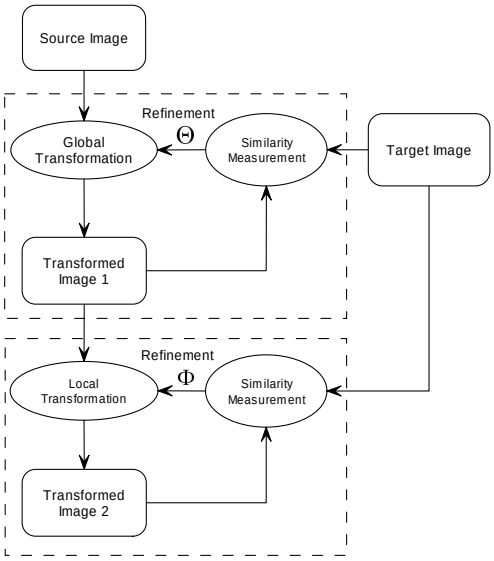

Figure 1: An approach to image registration.

In medical image processing, B-spline based free-form deformations (FFDs) are frequently used in non-rigid registration to aid the detection of cancer in 3D contrast-enhanced MRI. The goal of image registration in contrast-enhanced breast MRI is to relate any point in the post-contrast enhanced sequence to the pre-contrast enhanced reference image. The motion of the breast is non-rigid so that rigid or affine transformations alone is not sufficient for the motion correction of breast MRI. Therefore a combined transformation *T*, which consists of a global transformation and a local transformation, is defined as follows [14]:

$$T(x,y,z) = T_{global}(x,y,z) + T_{local}(x,y,z)$$

The basic idea of the FFD is to deform an object by manipulating an underlying mesh of control points.

To define a B-spline based FFD, the domain of the image volume is defined as $\Omega = \{(x,y,z)|0 \le x < X, 0 \le y < Y, 0 \le z < Z\}$. Let Φ denote a $n_x \times n_y \times n_z$ mesh of control points $\phi_{i,j,k}$ with uniform spacing. The FFD can be written as the 3D tensor product of the familiar 1D cubic B-splines:

$$\begin{aligned} &T_{local}(x,y,z) \\ &= \sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} B_i(u)B_j(v)B_k(w)\phi_{i+l,j+m,k+n} \end{aligned} \quad (1)$$

where

$$l = \lfloor \frac{x}{n_x} \rfloor - 1, \quad m = \lfloor \frac{y}{n_y} \rfloor - 1, \quad n = \lfloor \frac{z}{n_z} \rfloor - 1,$$

$$u = \frac{x}{n_x} - \lfloor \frac{x}{n_x} \rfloor,\ v = \frac{y}{n_y} - \lfloor \frac{y}{n_y} \rfloor,\ w = \frac{z}{n_z} - \lfloor \frac{z}{n_z} \rfloor$$

and B_i represents the *i*-th basis function of the B-spline.

$$\begin{aligned} B_0(u) &= (1-u)^3/6 \\ B_1(u) &= (3u^3 - 6u^2 + 4)/6 \\ B_2(u) &= (-3u^3 + 3u^2 + 3u + 1)/6 \\ B_3(u) &= u^3/6 \end{aligned} \quad (2)$$

where $u \in [0,1]$.

To relate a post-contrast enhanced image to the pre-contrast enhanced reference image, we must define a similarity criterion which measures the degree of alignment between two images. Given that the image intensity might change after the injection of the contrast agent, one cannot use a direct comparison of image intensities, such as sum of squared differences (SSD) or correlation, as a similarity measure. Alternatively, mutual information (MI) has been chosen as a voxel-based similarity measure. To avoid any dependency on the amount of image overlap, the use of normalized mutual information (NMI) was suggested as a measure of image alignment:

$$C_{similarity}(A,B) = \frac{H(A) + H(B)}{H(A,B)} \quad (3)$$

$$\begin{aligned} C_{smooth} &= \frac{1}{V}\int_0^X\int_0^Y\int_0^Z \Big[\Big(\frac{\partial^2 T}{\partial x^2}\Big)^2 + \Big(\frac{\partial^2 T}{\partial y^2}\Big)^2 \\ &\quad + \Big(\frac{\partial^2 T}{\partial z^2}\Big)^2 + 2\Big(\frac{\partial^2 T}{\partial xy}\Big)^2 + 2\Big(\frac{\partial^2 T}{\partial xz}\Big)^2 \\ &\quad + 2\Big(\frac{\partial^2 T}{\partial yz}\Big)^2\Big] dxdydz \end{aligned} \quad (4)$$

To find the optimal transformation, we minimize a cost function associated with the global transformation parameters Θ, as well as the local transformation parameters Φ. The cost function comprised two competing goals. The first term represents the cost associated with the image similarity $C_{similarity}$ in Equation 3, while the second term corresponds to the cost associated with the smoothness of the transformation C_{smooth} in Equation 4. Here, λ is the weighing parameter which defines the trade-off between the alignment of the two image volumes and the smoothness of the transformation.

$$C(\Theta,\Phi) = -C_{similarity}(I(t_0), \mathbf{T}(I(t))) + \lambda C_{smooth}(\mathbf{T}) \quad (5)$$

For computational efficiency, the optimization proceeds in several stages. Initially, the affine transformation parameters Θ (upper dashed box in Figure 1) are optimized at increasing levels of image resolution by maximizing the

```
For k=0 to k=M
Begin
  K = k + n - 1
  if (0 <= K <= N)
    out = out + const*Phi[K]
End
```

where $n \in [0, N-1],\ K \in [-1, N+M-2]$

Figure 2: Conditional for-loop, where n, M and N are integers, and also $N \geq M$.

```
For k=0 to k=M
Begin
  K   = k + n
  out = out + const*Phi[K]
End
```

where $n \in [0, N-1],\ K \in [0, N+M-1]$

Figure 3: Transformed loop with limited range of input, where n, M and N are integers, and also $N \geq M$.

normalized mutual information (NMI) (Equation 3). During the subsequent refinement, the non-rigid transformation parameters Φ (lower dashed box in Figure 1) are optimized at increasing levels of resolution of the control point mesh. The approach to our image registration is shown in Figure 1.

3 Conditional Loop Transform

In the three nested for-loop of the B-spline based FFD local deformation, there are three conditions which determine whether the loop body is executed or not. All of these conditions not only depend on the for-loop variables, but also the input values of the deformation.

Assuming that we have a simplified conditional for-loop as shown in Figure 2. We can transform the conditional for-loop into one without the *if* statement by including two more elements in the $Phi[K]$ data array (Figure 3) with one in the first and the other in the last of the array. Hence, $Phi[K]$ has $(N+M)$ instead of $(N+M-2)$ elements, and the value of $Phi[0]$ and $Phi[N+M-1]$ are assigned to zero so that the final value of out would not be affected. This method can be used to transform the FFD computation as explained below.

Figure 4 shows the pseudo code of the nested for-loop for the FFD computation used in 3D images. The inner loop body would be executed only when K, J and I are all satisfied. The variables n, m and l are the integer part of the input fixed-point numbers, of which u, v and w are the fraction part. Although one can implement a sequential

```
For k=0 to k=3
Begin
  K = k + n - 1
  if (0 <= K < CP_Z)
    For j=0 to j=3
    Begin
      J = j + m - 1
      if (0 <= J < CP_Y)
        For i=0 to i=3
        Begin
          I = i + l - 1
          if (0 <= I < CP_X)
            x = x + Bi(u) * Bj(v) * Bk(w) * Phi_X[K][J][I]
            y = y + Bi(u) * Bj(v) * Bk(w) * Phi_Y[K][J][I]
            z = z + Bi(u) * Bj(v) * Bk(w) * Phi_Z[K][J][I]
        End
    End
End
```

Figure 4: Pseudo code of B-spline based free-form deformation, where $l = \lfloor x/n_x \rfloor, m = \lfloor y/n_y \rfloor, n = \lfloor z/n_z \rfloor$.

```
For k=0 to k=3
Begin
  K = k + n
  For j=0 to j=3
  Begin
    J = j + m
    For i=0 to i=3
    Begin
      I = i + l
      x = x + Bi(u) * Bj(v) * Bk(w) * Phi_X[K][J][I]
      y = y + Bi(u) * Bj(v) * Bk(w) * Phi_Y[K][J][I]
      z = z + Bi(u) * Bj(v) * Bk(w) * Phi_Z[K][J][I]
    End
  End
End
```

Figure 5: Pseudo code of transformed loop, where $l = \lfloor x/n_x \rfloor, m = \lfloor y/n_y \rfloor, n = \lfloor z/n_z \rfloor$.

hardware implementation using Handel-C [4], the performance is predictably low. In order to get high throughput in hardware, pipelining is a useful technique. However, the conditions within the nested loop make it difficult to be pipelined.

Following the method explained earlier, we use a transformed loop structure (Figure 5) that eliminates the conditional statements by assigning the first and the last elements of the transformed data array to zero. A control lattice of a 2D FFD computation after the transformation is shown in Figure 6, in which the gray control points outside the image are the appended points after the transformation. With this method, a pipelined implementation of B-spline based FFD algorithm for processing 2D images can been successfully fitted into an FPGA.

4 Pipelined Designs for FFD

Our pipelined designs for FFD mainly involves three steps. The first step is to precalculate the four basis functions of a third-order B-spline, as shown in Equation 2, and store the values in four lookup tables accordingly. The second step is to design a pipelined FFD core. The third step

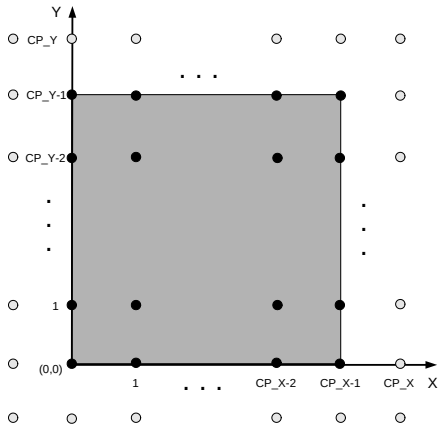

Figure 6: The arrangement of control lattice on a 2D image.

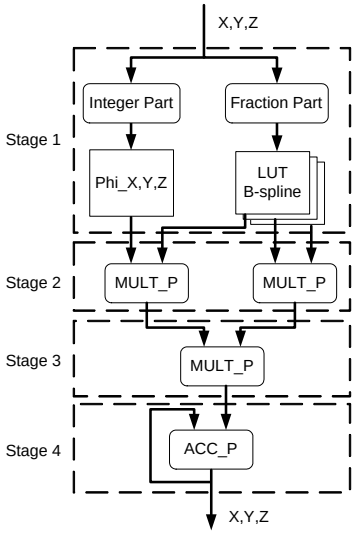

Figure 7: Single-channel pipelined hardware for free-form deformation computation. MULT_P denotes a pipelined floating-point or fixed-point multiplier. ACC_P denotes a pipelined floating-point or fixed-point adder.

is to deploy multiple pipelines.

We precalculate the values of each basis function of the third-order B-spline and store it in a lookup table with 1024 entries. The reason we choose 1024 entries for each basis function of the B-spline is that the highest resolution of a 2D medical image that we use is 2048 by 2048, and the lowest possible mesh of control points is 3 by 3. Therefore, the smallest possible u (Equation 2) is $(3-1)/(2048-1)$, which means a lookup table with 1024 entries for approximating each basis function is accurate enough.

Thus, for four basis functions of the third-order B-spline, we have four lookup tables, each with 1024 entries. With a customised data format of a 12 bit-width version, each lookup table requires 1024$\times$12 bits.

Before the pipelined multiplier and adder, there is a Stage 1 circuit which pre-processes input data in fixed-point format (Figure 7). This circuit can be obtained by applying the slowdown transformation used in deriving systolic array designs [12]. The input data x, y and z to this stage are interleaved and processed so that the integer and fraction part of the lookup table containing the data can be stored in separate temporal registers. Thereafter the integer part is used as the index to the input control point lattice, which is stored in external memories (LUT_CP X, Y, Z block in Figure 7), while the fraction part is used as the index to the B-spline lookup tables (LUT B-spline block in Figure 7). All the results that come from the lookup tables and control point lattice are fed into the next stage of the pipeline on every cycle.

In Stage 2, 3 and 4, one pipelined adder and three pipelined multipliers are used to produce the final result. All these four arithmetic operators are in customisable fashion. We can either choose floating-point arithmetic format [1], which is based on Xilinx LogiCORE [16], or alternatively, fixed-point format. Stage 2 and 3 each takes six cycles, while Stage 4 takes three cycles.

Currently, we only use one input channel; therefore, the total number of execution cycles is around

$$N^d \times (i+1)^d \times 3 \tag{6}$$

where N denotes the pixels in each axis of an image, i denotes the order of B-spline, and finally d represents the dimension of an image.

The input data have to be interleaved so that the whole pipeline could be fully used. In the pseudo code (Figure 3), we can see that the value of $Bi(u)$, $Bj(v)$ and $Bk(w)$ need to be accessed simultaneously in order to make a more efficient pipeline. Therefore, we replicate the lookup tables twice to meet the requirement of accessing three independent lookup tables concurrently (Equation 1).

For implementations targeting Celoxica's RC2000 board with a Xilinx XC2V6000 chip, the total cost for a 12 bit-width mantissa with 8 bit-width exponent representation is only 25 percent of the block RAM resources.

A faster system can be achieved as follows. For the purpose of illustration, only 2D images are considered. A 2D version of the algorithm in Figure 4 can be transformed by

Table 1: Estimation of hardware resources in two different implementations.

Implementations	One channel	Three channels
Multipliers	3	5
Adders	1	3

```
For j=0 to j=3
Begin
  J = j + m0..m15    //within a grid, the value
                     //of m0..m15 is the same
  For i=0 to i=3
  Begin
    I = i + l0..l15  //within a grid, the value
                     //of l0..l15 is the same
    x0 = x0 + Bi(u0) * Bj(v0) * Phi_X[J][I]
    x1 = x1 + Bi(u1) * Bj(v1) * Phi_X[J][I]
    ...
    x15 = x15 + Bi(u15) * Bj(v15) * Phi_X[J][I]

    y0 = y0 + Bi(u0) * Bj(v0) * Phi_Y[J][I]
    y1 = y1 + Bi(u1) * Bj(v1) * Phi_Y[J][I]
    ...
    y15 = y15 + Bi(u15) * Bj(v15) * Phi_Y[J][I]
  End
End
```

Figure 8: Pseudo code of transformed loop, where l0..l15 are the integer part of x0..x15, m0..m15 are the integer part of y0..y15, u0..u15 are the fraction part of x0..x15, v0..v15 are the fraction part of y0..y15.

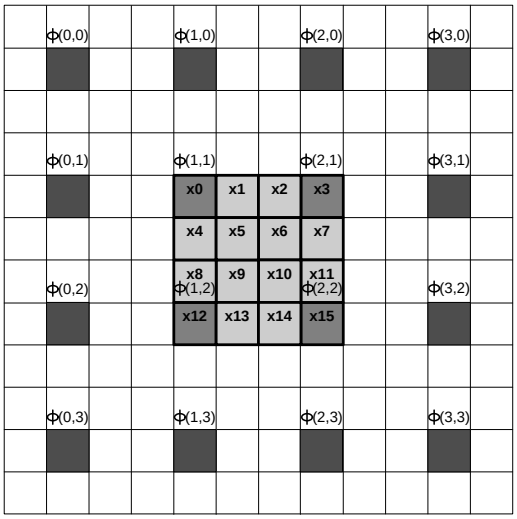

Figure 9: The effect of control lattice on the pixels in the grid.

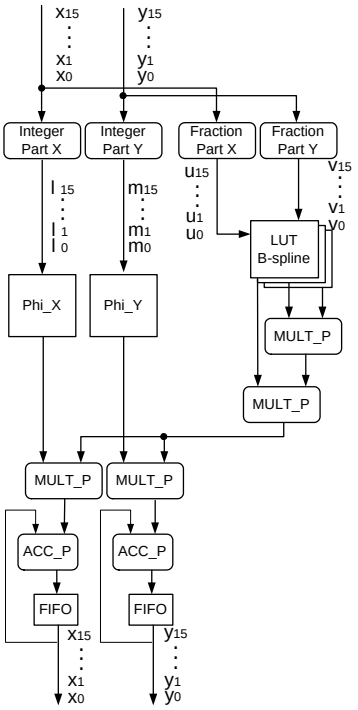

Figure 10: The new 2-channel architecture for processing 2D images.

inlining into the one shown in Figure 8. For all the pixels within a grid of control lattice, the surrounding sixteen control points have similar effect on them. As we can see from Figure 9, the 16 central gray pixels are manipulated by the control points from $\Phi(0,0)$ to $\Phi(3,3)$. Assuming that we have two input channels for storing the values of the position of pixels in x and y directions, and two more channels for the values of control point Phi_X and Phi_Y, we can feed in $x0..x15$ sequentially in the x channel while calculating the effect of upper-left $\Phi(0,0)$ on these pixels, then calculating the effect of $\Phi(1,0)$ on them after 16 clock cycles. Finally all the sixteen control points would have been applied on these pixels (Figure 10). The total memory banks that we need for processing 2D images are seven, two for channels of input data x and y, two for channels of output data, two for control points and one for the precalculated B-spline data. In the case of computing 3D images, ten memory banks are needed.

If we use this method to process 3D images of resolution N by N by N, the total number of cycles for execution is around $N \times N \times N \times 64$. In this case, there are 16×3 values coming out every 64×16 clock cycles. Compared with the previous architecture, the throughput is three times higher, however, it needs more hardware resources (Table 1).

Table 2: Hardware results of floating-point representation in different mantissa bit-width for a Xilinx XC2V6000 FPGA for images of 256 by 256 resolution.

Format	Exponent (bits)	Mantissa (bits)	Clock Speed (MHz)	Area (slices)
Floating-point	8	23	67.4	3423
	8	16	74.3	3340
	8	12	76.3	3189

Table 3: Hardware results of fixed-point representation in different fraction bit-width for a Xilinx XC2V6000 FPGA for images of 256 by 256 resolution.

Format	Integer (bits)	Fraction (bits)	Clock Speed (MHz)	Area (slices)
Fixed-point	8	23	72.8	4031
	8	16	83.3	3661
	8	12	85.1	3490

5 Custom Number Representation

Our investigation shows that normally a fixed-point multiplier requires more hardware resources than a floating-point multiplier with the same dynamic range. In this case, we have three multipliers and only one adder. Thus, we decide to implement the pipeline using floating-point arithmetic rather than to implement it using fixed-point arithmetic. A customisable floating-point library which utilizes the Xilinx LogiCORE has been used to construct our pipeline in hardware. The speed and area characteristics of our custom floating-point number representation are investigated (Table 2). For comparison, we have also implemented our design using fixed-point format (Table 3). The area cost of our design using fixed-point format is higher than the floating-point format, while the clock speed for the fixed-point format is slightly faster.

The width of the floating-point number in our custom floating-point library can be adjusted: we can change the mantissa and exponent parameters at compile time to achieve a variety of trade-offs in performance and hardware resources.

The accuracy of the entire calculation has been estimated by testing 2D images with a resolution of 256 by 256, and a 4 by 4 control point lattice. When comparing with the standard IEEE double-precision format with 52-bit mantissa, our design with 12-bit mantissa only has an average error of 0.0009 (Figure 11) and a maximum error of 0.0021 (Figure 12).

Based on the speed and area cost of this implementation

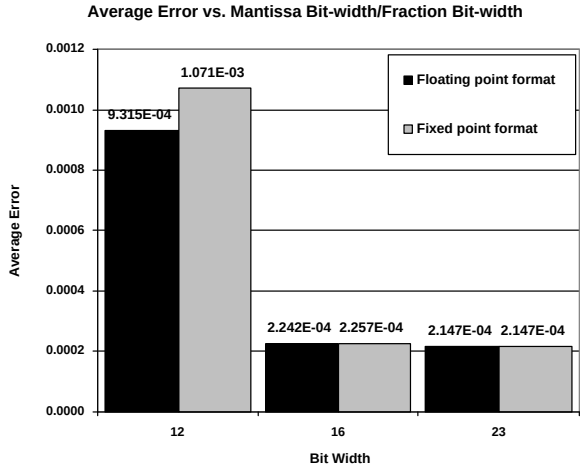

Figure 11: Average error against mantissa(floating-point) and fraction(fixed-point) size ranging from 12 bits to 23 bits.

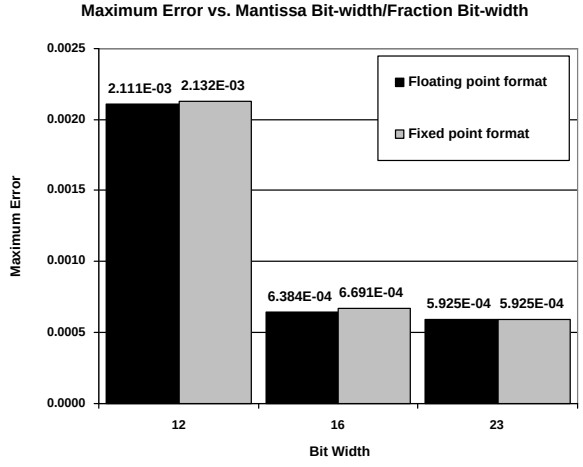

Figure 12: Maximum error against mantissa(floating-point) and fraction(fixed-point) size ranging from 12 bits to 23 bits.

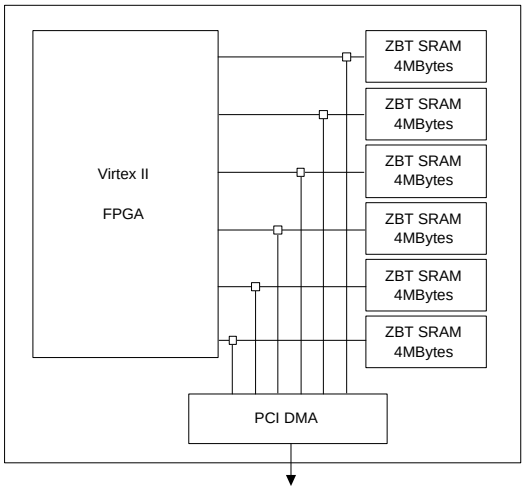

Figure 13: The RC2000 board with six memory banks.

in an FPGA, we conclude that a custom floating-point representation of 12-bit mantissa will be sufficient for the FFD calculation used in an image of resolution 256 by 256. As a result, it is possible to deploy several pipelines on a large FPGA regardless of memory boundary and available pins, such as Xilinx Virtex II devices.

6 Performance

For a 2D image of resolution 256 by 256, the clock speed after place and route of our current two-pipeline implementation on a Xilinx Virtex II XC2V6000 device (Table 4) with 12-bit mantissa and 8-bit exponent is 67 MHz, which still exceeds the clock rate between the FPGA and the six ZBT SRAMs on the RC2000 (Figure 13), which only supports up to 67 MHz. Therefore, the estimated execution time for the XC2V6000 device is

$$\frac{256 \times 256 \times 16 \times 3}{67 \times 10^6 \times 2} \simeq 0.023(second)$$

Compared with the software version which runs on an Intel Xeon 2.66 GHz PC, the execution speed of our customised system is around 2.2 times faster. If we are not limited by the number of external memory banks, then the optimised architecture that we propose in Section 4 can be chosen as our FFD processor. The execution speed would be 3.2 times faster than the software version, if we can run this design at around 67 MHz.

Looking towards the future, medical images are adopting increasingly higher resolution. For instance, if we need to process a 2D image of resolution 1024 by 1024, then it is essential to partition the image into multiple subimages, such that each subimage will be dealt with by a separate processor, which also has access to the relevant control point information. To achieve near real-time performance, we can partition each high-resolution image into 3 by 3 subimages. A system containing 9 pipelined processors running in parallel is required: each processor would process one subimage, which contains 12-bit fixed-point data, at around 85 MHz (see Table 3). For this system, the slice resources required would be around 3490 $\times$ 9 = 31410, whereas the Xilinx XC2V6000 has 33792 slices. The estimated execution time is (1024 $\times$ 1024 $\times$ 48) / (85 $\times$ 10^6 $\times$ 9) = 66 milliseconds, which gives a frame rate of around 15 frames per second.

7 Summary

We have described hardware techniques for medical image processing. The key elements of our approach include precalculating the B-spline basis function, adopting custom number representation, transforming a nested loop to avoid conditional calculation, and developing fully-pipelined circuits and multiple pipeline designs. Current and future work includes integrating our design with a hardware image warper [11], automating conditional loop transformations [17], and exploring the use of run-time reconfiguration to reduce the amount of FPGA resources required [15].

Acknowledgements

The support of Xilinx, Inc., Celoxica Limited, and the UK Engineering and Physical Sciences Research Council (Grant number GR/N 66599, GR/R 31409 and GR/R 55931) is gratefully acknowledged.

References

[1] A. Abdul Gaffar, W. Luk, P.Y.K. Cheung and N. Shirazi, "Customising floating-point designs," *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, IEEE, 2002.

[2] X.L. Battle, C. Le Rest, A. Turzo and Y. Bizais, "Free-form deformation in tomographic reconstruction. Application to attenuation map reconstruction," *IEEE Transactions on Nuclear Science*, vol.47, pp. 1065 - 1071, June 2000.

[3] C. Berthaud, E. Bourennane, M. Paindavoine and C. Milan, "Implementation of a real time image rotation using B-Spline interpolation On FPGA's Board," *International Conference on Image Processing, 1998. ICIP 98. Proceedings. 1998*, vol.3, pp. 995–999, 1998

Table 4: Performance comparison of FFD processors for 2D images of 256 by 256.

Processor	Data Format	Clock Speed (MHz)	Exec.Time (ms)	Throughput (pixels/second)
XC2V6000 (one pipeline, Figure 10)	fixed-point	67	16	4187500
XC2V6000 (two pipelines, Figure 7)	floating-point	67	23	2791667
XC2V6000 (one pipeline, Figure 7)	fixed-point	85.1	37	1772917
XC2V6000 (one pipeline, Figure 7)	floating-point	76.3	41	1583333
Xeon (Dual processor)	floating-point	2666	50	1310720
AMD Athlon	floating-point	1400	100	655360
Pentium 4	floating-point	1800	110	595782
Pentium III	floating-point	933	140	468114

[4] *http://www.celoxica.com*, Celoxica Limited.

[5] S. Coric, M. Leeser, E. Miller and M. Trepanier, "Parallel-beam backprojection: an FPGA implementation optimized for medical imaging," *Proc. ACM International Symposium on FPGAs*, ACM Press, 2002.

[6] L.A. Ferrari and J.H. Park, "An efficient spline basis for multi-dimensional applications: image interpolation," *Proceedings of 1997 IEEE International Symposium on ISCAS '97*, vol.1, pp. 757–760,1997.

[7] Jianlin Gao, MengChu Zhou, Haimin Wang and Chongzhe Zhang, "Three dimensional surface warping for plastic surgery planning," *2001 IEEE International Conference on Systems, Man, and Cybernetics*, vol.3, pp. 2016–2021, 2001.

[8] S. Horbelt, A. Munoz, T. Blu and M. Unser, "Spline kernels for continuous-Space image processing," *IEEE International Conference on Acoustics, Speech, and Signal Processing, 2000. ICASSP '00. Proceedings. 2000*, vol.4, pp. 2191 -2194, 2000.

[9] J. Jiang, W. Luk and D. Rueckert, "FPGA-based Computation of Free-Form Deformations," *IEEE International Conference on Field-Programmable Technology*, pp. 407–410, 2002.

[10] J. Jiang, W. Luk and D. Rueckert, "An FPGA-based Computation of Free-Form Deformations", *Proc. Field-Prog. Logic and Applications*, LNCS 2778, Springer-Verlag, 2003.

[11] J. Jiang, S. Schmidt, W. Luk and D. Rueckert, "Parameterizing designs for image warping," *Reconfigurable Technology: FPGAs and Reconfigurable Processors for Computing and Communications, Proc. SPIE*, vol. 4867, 2002.

[12] S.Y. Kung, "VLSI Array Processors," *Prentice Hall*, 1988.

[13] D. Rueckert, C. Hayes, C. Studholme, P. Summers, M. Leach and D. J. Hawkes, "Non-rigid Registration of Breast MR images using Mutual information," *In First Int. Conf. on Medical Image Computing and Computer-Assisted Intervention (MICCAI '98), Lecture Notes in Computer Science, Cambridge, MA*, pp. 1144–1152, 1998. Springer-Verlag.

[14] D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. 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.

[15] N. Shirazi, W. Luk and P.Y.K. Cheung, "Framework and tools for run-time reconfigurable designs," *IEE Proc. Comput. Digit. Tech.*, May 2000, pp. 147–152.

[16] *http://www.xilinx.com/products/logicore/lcoredes.htm*, Xilinx, Inc.

[17] M. Weinhardt and W. Luk, "Pipeline vectorization," *IEEE Trans. on Computer-Aided Design*, February 2001, pp. 234–248.

[18] T. Yokota, M. Nagafuchi, Y. Mekada, T. Yoshinaga, K. Ootsu and T. Baba, "A Scalable FPGA-based Custom Computing Machine for a Medical Image Processing," *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 307–308, 2002.