In Proceedings of the IEEE Asilomar Conference on Signals, Systems, and Computers, pages 1453-1457, Pacific Grove, California, October 2006.

MODEL-BASED MAPPING OF IMAGE REGISTRATION APPLICATIONS ONTO CONFIGURABLE HARDWARE

Yashwanth Hemaraj[1,2], *Mainak Sen*[1], *Raj Shekhar*[2], *Shuvra S. Bhattacharyya*[1]

[1]Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, 20742, USA.
[2]Department of Diagnostic Radiology, University of Maryland School of Medicine, Baltimore, Maryland, USA

ABSTRACT

This paper develops techniques for mapping rigid image registration applications onto configurable hardware. Image registration is a computationally intensive domain that places stringent requirements on performance and memory management efficiency. Building on the framework of homogeneous parameterized dataflow, which provides an effective formal model for design and analysis of hardware and software for signal processing applications, we develop novel methods for representing and exploring the hardware design space when mapping image registration algorithms into configurable hardware. Our techniques result in an efficient framework for trading off performance and configurable hardware resource usage based on the constraints of a given registration application.

1. INTRODUCTION

Computationally intensive medical imaging applications have proven to be amenable to hardware acceleration. However growing data sets and a desire for real-time capabilities are placing continued pressure on performance of such accelerators. Inhibiting further performance of these devices is accurate modeling of the applications. Faithful modeling of an application's concurrency and dependencies is critical to arriving at an efficient and reliable implementation in hardware. In addition to exposing parallelism effectively, dataflow concepts can provide valuable formal properties such as efficient verification of bounded memory requirements and deadlock-free operation.

Modeling of DSP applications based on coarse-grain dataflow graphs is widespread in the DSP design community, and a large and growing set of DSP design tools support such dataflow semantics [1]. DSP applications differ in their requirements and complexity. The dataflow interchange format (DIF) [5] provides us with integrated support for a variety of dataflow semantics and features, including dynamic and reconfigurable dataflow behavior.

In this paper, we propose a new approach towards implementation of image registration algorithms that captures their concurrency and dependencies along with dynamic behavior. As a result, designers can efficiently exploit trade-offs between area and performance for different configurations of the circuit to customize their final implementation based on the input characteristics and the target platform. We demonstrate the value of this approach by showing the area and computational time for a set of possible implementations.

2. DATAFLOW MODELING

Digital signal processing applications lend themselves well to the semantics of dataflow. Design entry methods based on dataflow models of computation provide a natural way of describing applications and also expose high-level application structure that is useful for analysis, verification, and optimization of implementations [1]. In the dataflow model of computation, the computational DSP elements (dataflow graph vertices) are called *actors*. DSP designers may choose an arbitrary level of granularity for actors. Typically, the complexity of an actor ranges from a single operation to computation that could be expressed by up to a hundred lines of C code. Actors are connected with directed edges to indicate data (or token) movement. An edge acts as a first-in, first-out (FIFO) queue that holds tokens that are produced by the edge's source actor until consumed by the edge's sink actor. An actor may be executed (or *fired*) whenever it has enough data on all its input edges. When an actor fires, it consumes a set of input data values and produces a set of output data values. By examining data transfer patterns between actors, a *schedule* can be created that provides a way to coordinate the execution of all of the actors in the dataflow graph.

The firing semantics of actors varies between the different forms of dataflow. The synchronous dataflow (SDF) model [6] enforces that a given actor must consume (and produce) a predetermined number of tokens at each firing. This restriction allows for strong compile time predictability properties. SDF is a relatively mature form of dataflow and SDF-based hardware synthesis has been explored in [9][13]. These restrictions imposed by SDF, however, are too stringent for some applications, especially applications with dynamic production and consumption rates. For example, many computer vision applications have data-dependent rates of data transfer between actors [10], which cannot be captured in pure SDF.

To accommodate more applications, other forms of dataflow have more flexible firing rules (i.e., rules for determining when actors can execute). A cyclo-static dataflow (CSDF) [2] graph can accommodate multi-phase actors that exhibit different consumption and production rates during different phases provided they adhere to statically-known periodic patterns. However, this model still does not permit data-dependent production or consumption patterns. Homogeneous parameterized dataflow (HPDF) [11] permits actor behavior to adapt in a structured way through dynamically-adjusted parameter values. HPDF actors may change their production and consumption rates at run-time in between successive iterations of the graph, but at any given point in time, any HPDF edge will have the same data production and consumption rates for its respective sink and source actors. The restricted form of data-dependent behavior supported by HPDF permits useful modeling flexibility, and also provides for efficient scheduling and resource allocation for actors, as well as verification of bounded memory requirements and deadlock avoidance. Furthermore, since HPDF is a meta-modeling technique, hierarchical actors in an HPDF model can be refined using any dataflow modeling semantics that provides a well-defined notion of subsystem iteration. For example, a hierarchical HPDF actor can have SDF, CSDF, or HPDF actors as its constituent modules.

3. IMAGE REGISTRATION

Image registration is the process of aligning two images that represent the same feature. Image registration can be thought of as a mapping function $F:I \rightarrow R$ that accepts an image to be mapped (also called the floating image I) and returns the image transformed such that it can map directly onto another image (also called the reference image R). Medical image registration concentrates on aligning two or more images that represent the same anat-

omy from different angles, scales, or offsets. The images may be obtained at different times or using different imaging modalities such as magnetic resonance imaging (MRI), computed tomo-graphy (CT), positron emission tomography (PET), single photon emission computed tomography (SPECT) and ultrasound. Image registration is an important tool in merging and comparing images obtained from this diversity of sources. Real-time image registration is essential in the medical field for enabling image guided treatment procedures and pre-operative treatment planning.

The various algorithms proposed for image registration can be broadly classified under two categories: *rigid* and *elastic*. A rigid registration can be approximated by a superposition of rotation, translation, scaling and shear. An elastic transformation on the other hand is based on nonlinear continuous transformations, and is implemented by finding separate transformation parameters for a set of control points.

3-dimensional image registration is of particular interest to the medical community. The ability to register a 3D image allows features to transform in multiple axes. A variety of techniques have been employed to solve this registration problem. Voxel similarity-based algorithms compute a similarity metric for each voxel in an image and this approach performs better than feature-based approaches [4]. The most commonly used similarity-based technique is mutual information based image registration. Mutual information (MI) based image registration can be very robust and can work with multi-modal images effectively. Figure 1 describes the process of image registration based on MI computation.

3.1 Mutual information based image registration

Mutual information-based image registration relies on the maximization of the mutual information between two images. Mutual information is a function of two 3-D images and a transformation between them. The transformation matrix contains the information about the rotation, scaling shear, and translations that need to be applied to one of the images in order to map it completely to the other image. A cost function based on the mutual information is calculated from the individual and joint histograms. The transformation that maximizes the cost function is the optimum transformation. The goal of the image registration application is to find the transformation T such that

$$T = \arg max_T MI(RI(x, y, z), FI(T(x, y, z))),$$

where RI is the reference image and FI is the floating image.

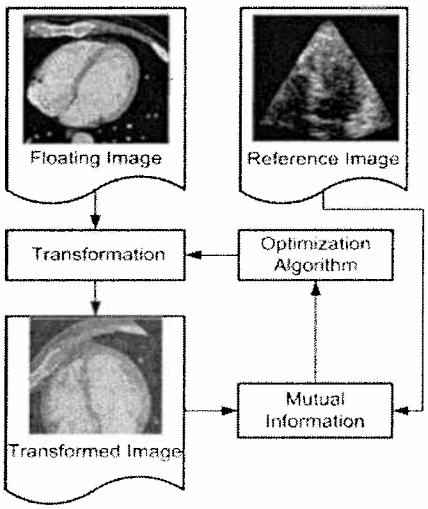

Fig 1. Image registration based on mutual information.

3.2 Computation of mutual information

Mutual information is calculated from individual and joint entropies using the following equations:

$$MI(RI, FI) = H(RI) + H(FI) - H(RI, FI),$$
$$H(RI) = -\Sigma p_{RI}(a)\log p_{RI}(a), H(FI) = -\Sigma p_{FI}(a)\log p_{FI}(a),$$
$$\text{and } H(RI, FI) = -\Sigma p_{RI, FI}(a, b)\log p_{RI, FI}(a, b), \quad (1)$$

where $H(RI)$, $H(FI)$, $H(RI, FI)$ and $MI(RI, FI)$ are the reference image entropy, floating image entropy, joint entropy and mutual information between the two images for a given transformation respectively.

The mutual histogram represents the joint intensity distribution. The joint voxel intensity probability, $p_{RI, FI}(a, b)$ is the probability of a voxel in the reference image having an intensity a and the corresponding voxel for a particular transformation T in the floating image having an intensity b, can be obtained from the mutual histogram of the two images.

The individual voxel intensity probabilities are the histograms of the reference and floating images in the region of overlap of the two images for the applied transformation. The individual histograms can be computed by taking the row sum and the column sum of the joint histogram.

The calculation of mutual information starts with the accumulation of the mutual histogram values to the mutual histogram memory while every coordinate is being transformed (MH update stage). This is followed by the MI calculation stage where the values stored in the mutual histogram memory are used to find the individual and joint entropies described above. Thus the overall calculation of mutual information is a memory intensive task. Since the access of the MH memory is dependent on the intensity values of the floating and the reference images, standard cache based memory architectures do not accelerate the calculation.

In the MH update stage, voxel coordinates are multiplied by the transformation matrix and the resultant coordinates obtained are used to update the joint histogram. Since the new coordinates do not always coincide with the location of a voxel in the reference image, interpolation schemes need to be employed. In the trilinear interpolation scheme, the new value of the floating image $FI(x', y', z')$ is calculated based on the amount of offset the new coordinates (x', y', z') have from the nearest voxel position. However this scheme makes the MH sparse and hence ineffective in MI calculation. Maes et al. [7] showed that the partial volume interpolation scheme does not cause such unpredictable variations in the MH values as the transformation matrix changes. This method accumulates the eight interpolation weights directly into the mutual histogram instead of calculating a resultant intensity level and increment that intensity level's MH count by one, as in trilinear interpolation. Thus the partial volume interpolation scheme ensures a smooth transition in the MH memory and hence causes smooth MI changes for various transformations applied.

Constructing the mutual histogram, the first step in mutual information calculation, involves performing partial volume interpolation n times, where n is less than or equal to the number of voxels in the reference image. The number of operations in the second step, the calculation of mutual information, is a function of the size of the mutual histogram, which is less than the size of the image making the first step the performance bottleneck.

It has been shown that the size of the mutual histogram can be selected as 64×64 for 8-bit images. By doing so, we can obtain a very good density of MH values while at the same time preserving the variation along the different entries.

At current microprocessor speeds, the time of mutual histogram calculation for 3-D images is dominated almost exclusively

by the memory access time [3]. Around 25 memory accesses are needed to perform partial volume interpolation per voxel of the reference image: 1 to access the reference image voxel, 8 to access the 8-voxel neighborhood in the floating image and 16 accesses to the mutual histogram memory (8 reads to get the old value and 8 writes to write back the updated value). Accesses to the reference image are sequential and standard caching techniques can be used. The mutual histogram memory has a small size and thus accesses to it also have high locality of reference. However, the floating image is accessed in a direction across the image that depends on the transformation being applied. Unless there is no rotation component, this direction is not parallel to the direction in which voxels are stored, hence accesses have poor locality and are not suited to memory-burst accesses or memory-caching schemes.

Speedup of the algorithm can be obtained by using pipelined architectures and also by using parallel architectures [3]. Since the majority of the registration execution time is spent on calculating the mutual histogram, accelerating mutual histogram calculation has been the focus of our work. The aim of this paper is to use dataflow graphs to describe the inherent concurrency in applications, analyze the bottleneck areas, and use dataflow graph transformations to exploit potential areas that can be parallelized.

3.3 Optimization

The image registration algorithm calculates the transformation matrix for which the mutual information between the images is maximum. Initially, a small number of test transformations are applied. The values of these transformations and the MI values are stored in an *optimizer*. The optimizer outputs the values of the new transformation depending on the values of the mutual histogram in the previous iterations. Optimization of the transformation parameters depends on the nature of the images and the amount of misalignment between the two images. In the downhill simplex optimization method, which provides fast convergence, in order to optimize a transformation with m parameters, the optimizer needs to store $(m+1)$ previous values. There is a trade-off between the convergence time and the complexity of the optimizer.

4. APPLICATION MODELING

In this section, we construct a hierarchical dataflow representation of the MI-based image registration algorithm and we use the HPDF meta-modeling approach integrated with CSDF for modeling lower-level, multi-phase interactions between actors. Figure 2 shows our top level HPDF model of the application. Here, "$(m+1)D$" represents $(m+1)$ units of *delay*; each unit of delay is analogous to the z^{-1} operator in signal processing, and is typically implemented by placing an initial data value on the corresponding dataflow edge. The MI actor consumes one data value (*token*) on every execution. This token contains co-ordinates of the reference image and the floating image. After s executions, each consuming one token (coordinate values in this case), where s denotes the size of the image, the MI actor produces the entropy between the reference and floating images. This value is then sent to the optimizer as a single token.

The optimizer, which stores the previous $(m+1)$ values to perform a simplex optimization of an m-parameter transformation vector, sends m tokens to the MI actor. Since m can vary depending on the number of parameters used to represent the desired transformation, the associated edge represents a variable-rate edge of the HPDF graph. A valid schedule for this HPDF graph is

$$(s\alpha)\beta\alpha. \qquad (2)$$

The internal representation of the hierarchical MI actor is shown in Figure 3. Here, "Reference Image" (A) consumes one token (coordinates) and produces one token (intensity values at the input coordinates), and "Coordinate Transform" (B) produces one token, which represents the transformed coordinates. If this voxel is valid (i.e., the voxel coordinate falls within the floating image coordinates boundary), it is passed on to the "Weight Calculator" (D) and "Floating Image" (E).

Now since all voxels may not be valid, r tokens $(r \le s)$ are produced from the "Is Valid" (C) actor. This actor also produces r tokens on the edge that connects it to "MH Memory" (G) — specifically, it passes a token from A only if a valid voxel results from the transformation on input coordinates. For every input token in D and E, eight output tokens are produced on both the outgoing edges. The corresponding eight intensity locations in the G are updated based on the tokens produced by D.

After all coordinates are processed, which occurs during the the first $8r$ *phases* of the MH Memory actor or equivalently after s phases of the B actor, one token of size $q \times q$ is sent to the "Decomposer"(Z), which in turn sends out $q \times q$ tokens to the "Entropy Calculator" (H) actor. H consumes all of these tokens, and produces a single token that contains the entropy value corresponding to the transformation applied based on equations given in eqn. 1. We added the Z mainly for ease of representation of the application in dataflow and it was subsumed by G in the final hardware implementation. A valid schedule (ordering of execution) for the mutual information subsystem based on Figure 3 is

$$(sABC)(rDE(8FG))(q^2ZH).$$

In this paper, we describe our schedules as *looped schedules* which is a compact form of representing the execution order of actors. A looped schedule of the form $(nT_1T_2...T_m)$ represents n successive repetitions of the execution sequence $T_1T_2...T_m$, where each T_i is either an actor or another looped schedule (to express nested looped schedules).

Looking more closely at B, we see that it has an additional input edge on which takes in the initial m tokens from the "Optimizer" (β) but produces no output. Figure 3 only represents the steady-state behavior of mutual information subsystem for simplicity. Figure 4 represents the initialization and the steady-state behavior of Coordinate Transform - where the initial m tokens are used to calculate the new transformation matrix and hence it

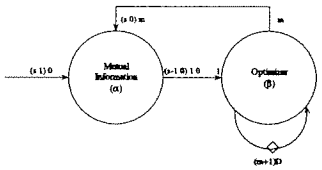

Fig 2. Top level model of image registration application.

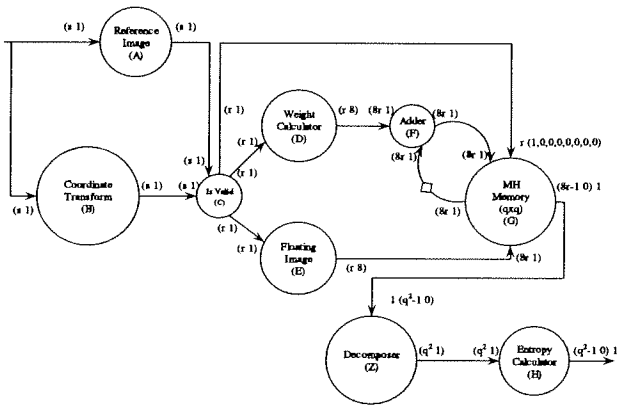

Fig 3. Dataflow model of mutual information subsystem.

updates the values inside the actor without producing any data. Hence the schedule of the whole mutual information subsystem — considering the initial and steady-state behavior of B — is:

$$(mB)(sABC)(rDE(8FG))(q^2ZH). \quad (3)$$

Figure 5 shows a parameterized dataflow model of the H. "Row Sum" (I) executes once every time it gets one row (q elements) to produce one token, but the "Column Sum" (L) can only produce an output for every input after it has already received $q \times (q-1)$ elements corresponding to $(q-1)$ rows. There are many valid schedules that can be proposed for Figure 5. Since a valid schedule for H is quite complex, we derive it step-by-step — the graph has three distinct paths, the upper path (involving Z, I, J, K) would have a schedule $(q(qZ)IJ)K$, the middle path (involving Z, L, N, O) would have a schedule $(q(q-1)ZL)(qZLN)O$, and the lower part of the graph (involving Z, T, U) would have a schedule $(q^2ZT)U$. Combining these, one possible valid schedule for the H subsystem is:

$$(q-1(qZLT)IJ)(qZTLN)IJKOUV. \quad (4)$$

Modeling of H exposes very large buffering overhead.

Combining eqn. 3 and eqn. 4, the schedule for the mutual information subsystem becomes:

$$(mB)(sABC)(rDE(8FG))(q-1(qZLT)IJ)(qZTLN)IJKOUV \quad (5)$$

and taking eqn. 2 also into account, the schedule for the whole image registration algorithm can be derived by replacing α with eqn. 5.

Interestingly, the model described above shows potential for parallel hardware mapping at various levels of abstraction. For example, extensive "intra-voxel" (within the processing structure

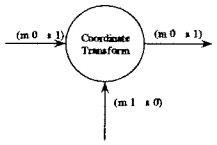

Fig 4. Initial and steady-state modeling of Coordinate Transform.

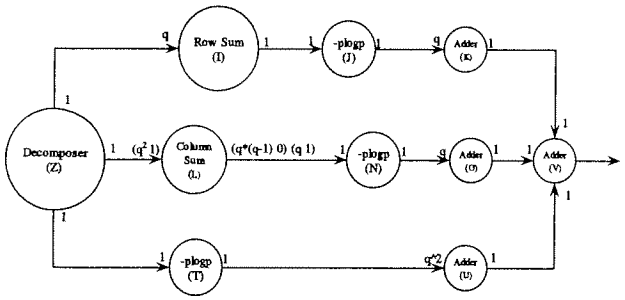

Fig 5. Parameterized "Entropy Calculator".

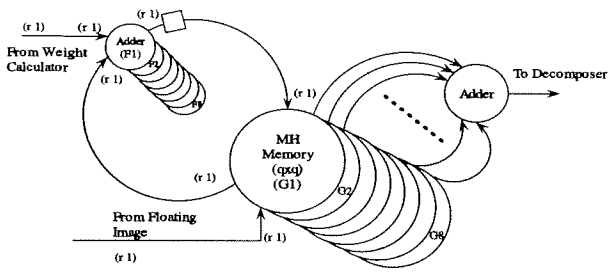

Fig 6. Parallel architecture for MH update.

for a single voxel) parallelism is possible for F and G. From Figure 3, we can see a data-rate mismatch between D, F; and E, G. This naturally suggests a parallel architecture as shown in Figure 6 where multiple copies (eight in the illustration as the data-rates are mismatched by a factor of eight) of an actor can be created for a parallel implementation. We also note that the resultant graph in Figure 6 becomes HPDF as all the parameterized actors now have the same production and consumption rates and hence fire at the same rate. The dataflow model also exposes inter-voxel parallelism, (as input actors A and B have s distinct phases where s is the number of voxels in the image), which leads to another set of useful parallel implementation considerations.

We develop an architecture in this paper that applies intra-voxel parallelism based on input characteristics.

5. EXPERIMENTAL SETUP

We explored in detail the effect on the application of having a parallel architecture as suggested by the dataflow model described above. In this paper, we extensively study the exploitation of intra-voxel parallelism. Thus, we varied the degree of parallelism as shown in Figure 6 and studied the resulting relationship between performance and area. We also noted that the percentage of voxels that fall in the valid range after a transformation by the "Coordinate Transform" greatly influences the runtime of the algorithm. Hence, we studied our system performance by varying the percentage of valid voxels (PVV) for a given transformation.

5.1 Degree of parallelism

When E in Figure 3 is provided with the base address in the floating image space, the actor generates the floating image values (corresponding to the neighborhoods) and provides it to the mutual histogram memory for updating the mutual histogram with the weights generated by the weight calculator actor. When we have just one set of actors (floating image, weight calculator and the mutual histogram memory), it takes eight firings of this set of actors (corresponding to the values of the eight neighborhood) before the next input can be processed by the coordinate transform actor. However, if we have two copies of the above set of actors, then each set can process four neighborhoods each. Similarly if we have four (or eight) copies, then each set can process two (or one) neighborhood(s) each. This would mean that the number of firings of each set of actors becomes 4, 2 (or 1) respectively. As updating the mutual histogram is a crucial part of the algorithm, such parallel execution should result in significant improvement of the whole application.

However, the parallel configurations result in extra FPGA resources and extra external memory. The memory requirement also increases with increasing image size.

5.2 Relationship between PVV and performance

We implemented our system as a self-timed system, so each actor is sent a 'ready' signal by the actors preceding it.

When the transformed coordinate falls in the valid region, there are eight firings of the actor set (F and G in Figure 3). However when C does not generate a signal (indicating that for the given input coordinates, the transformation produces coordinates outside of the valid coordinate boundary), the iteration of the graph stops for those input coordinates and the next token is processed by the coordinate transform actor indicating a new iteration. For our implementation, any isolated invalid signal causes a two cycle penalty, but consecutive invalid signals cause only one cycle penalty for each invalid signals.

5.3 Implementation

We implemented each actor in Verilog. The Verilog code

was simulated for functional correctness and synthesis was performed targeting the Altera StratixII EP2S15F484C5 device. The code was synthesized for different degrees of parallelism of the floating image and weight calculator actor.

6. RESULTS

In this section, we present hardware synthesis results for various proposed configurations of the image registration application. The results are obtained using the QuartusII synthesis tool from Altera for the StratixII family of FPGA (Stratix EP2S15F484C5). Table 1 presents the synthesis results for various configurations - the columns represent the number of parallel datapaths for MH Update actor and rows represent in order — external memory required, logic circuitry in the MH update actor, DSP elements for the MI actor, logic circuitry of the MI actor, and maximum frequency of operation for the MI actor. Table 1 assumes that PVV is 100%. Figure 7 shows the area (measured by the number of logic cell registers in the circuit without considering the external memory) and performance (measured by the number of execution cycles) trade-off curve when we vary the number of parallel datapaths in the MH update actor for different validity rates in transformed voxels. The trend in all of the above mentioned cases reflect that the number of execution cycles decreases with increasing amounts of parallel data paths, although the corresponding area increases. We notice that the PVV is an important metric for performance. The relative performance gain at lower PVV values by increasing the number of parallel data-paths is less than the performance gain at higher values of PVV.

Number of parallel datapaths	1	2	4	8
External Memory	256Kb	512Kb	1Mb	2Mb
LC Registers in FPGA	427	576	871	1463
DSP Elements	30	30	30	30
Total FPGA Area (number of ALUTs)	598	878	1439	2588
Max freq of operation (MHz)	74	72.2	74	70.1

Table 1 Synthesis results for the whole system for different configurations of the MH Update actor.

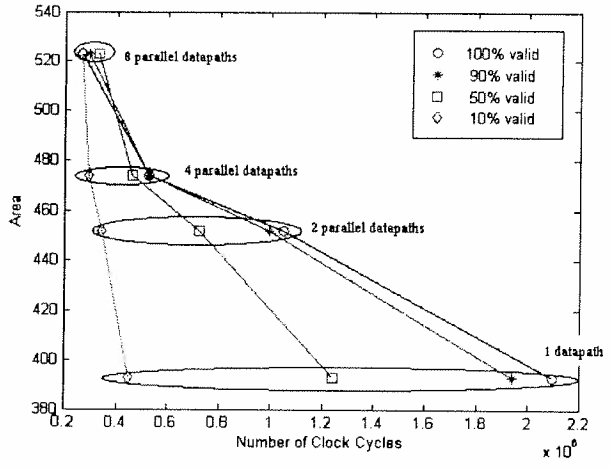

Fig 7. Area versus clock cycles for different PVV values for different numbers of datapaths.

A dynamically-reconfigurable FPGA implementation was proposed based on the PVV metric in [8]. We considered inter-voxel parallelism along with intra-voxel parallelism, and we developed a comparison of the associated performance gains in [8].

7. CONCLUSION

In this paper, we have proposed a model-based mapping of an image registration algorithm onto configurable hardware. Both inter- and intra-voxel parallelism is exposed by the carefully-deigned dataflow model of the application. We explored various levels of intra-voxel parallelism and presented area-performance trade-offs for different parallel configurations. Our experiments quantify how increasing the number of parallel data-paths results in increased area but decreased runtime. Also we show that the percentage of valid voxels (PVV) is an important metric in exploring the design space.

8. ACKNOWLEDGEMENTS

The authors are grateful to Dr. William Plishker for his reviews and insightful suggestions for this paper.

9. REFERENCES

[1] S. S. Bhattacharyya, R. Leupers, and P. Marwedel. Software synthesis and code generation for DSP. *IEEE Trans. on Circuits and Systems — II: Analog and Digital Signal Processing*, 47(9):849-875, September 2000.

[2] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete. Cyclo-static dataflow. *IEEE Trans. on Signal Processing*, 44(2):397-408, February 1996.

[3] C. Castro-Pareja, J M. Jagadeesh, and R. Shekhar. FAIR: A hardware architecture for real-time 3-d image registration. *IEEE Trans. on Information Technology in Biomedicine*, 7(4):426-434, 2003.

[4] M. Holden, D. Hill, E. Denton, J. Jarosz, T. Cox, T. Rohlfing, J. Goodey, and D. Hawkes. Voxel similarity measures for 3D serial MR brain image registration. *IEEE Trans. on Medical Imaging*, pages 94-102, 2000.

[5] C. Hsu, F. Keceli, M. Ko, S. Shahparnia, and S. S. Bhattacharyya. DIF: An interchange format for dataflow-based design tools. *In Proc. of the International Workshop on Systems, Architectures, Modeling, and Simulation*, pages 423-432, July 2004.

[6] E. Lee and D. Messerschmitt. Synchronous Data Flow. *Procs of the IEEE*, September 1987.

[7] F. Maes, D. Vandermeulen and P. Suetens, Medical image registration using mutual information, *Proc. IEEE* 19, 1699 (2003).

[8] M. Sen, Y. Hemaraj, S. S. Bhattacharyya, and R. Shekhar. Reconfigurable image registration on FPGA platforms. *In Proc. of IEEE Biomedical Circuits and Systems Conference*, November 2006.

[9] M. Sen and S. S. Bhattacharyya. Systematic exploitation of data parallelism in hardware synthesis of DSP applications. In *Procs of the International Conference on Acoustics, Speech, and Signal Processing*, pages V-229-V-232, May 2004.

[10] M. Sen, I. Corretjer, F. Haim, S. Saha, J. Schlessman, S. S. Bhattacharyya, and W. Wolf. Computer vision on FPGAs: Design methodology and its application to gesture recognition. In *Proc. IEEE Workshop on Embedded Computer Vision*, pages CD-ROM version, 8 pages, San Diego, California, June 2005.

[11] M. Sen, S. S. Bhattacharyya, T. Lv, and W. Wolf. Modeling image processing systems with homogeneous parameterized dataflow graphs. In *Proc. International Conference on Acoustics, Speech, and Signal Processing*, pages V-133-V-136, March 2005.

[12] R. Shekhar R, V. Zagrodsky, C. R. Castro-Pareja, V. Walimbe, and J. M. Jagadeesh. High-speed registration of three- and four-dimensional medical images by using voxel similarity. *RadioGraphics*, 23(6):1673-1681, 2003.

[13] M. Williamson. Synthesis of Parallel Hardware Implementations from Synchronous Dataflow Graph Specifications. *Ph.D. Thesis, University of California at Berkeley*, May 1998.