J Real-Time Image Proc
DOI 10.1007/s11554-008-0075-z

SPECIAL ISSUE

Model-based mapping of reconfigurable image registration on FPGA platforms

Mainak Sen · Yashwanth Hemaraj · William Plishker · Raj Shekhar · Shuvra S. Bhattacharyya

Received: 22 July 2007 / Accepted: 12 February 2008

Abstract Image registration is a computationally intensive application in the medical imaging domain that places stringent requirements on performance and memory management efficiency. This paper develops techniques for mapping rigid image registration applications onto configurable hardware under real-time performance constraints. Building on the framework of homogeneous parameterized dataflow, which provides an effective formal model of 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 onto 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 application. Based on trends that we have observed when applying these techniques, we also present a novel architecture that enables dynamically-reconfigurable image registration. This proposed architecture has the ability to tune its parallel processing structure adaptively based on relevant characteristics of the input images.

Keywords Dataflow · HPDF · Image registration · Reconfigurable architectures

M. Sen (✉) · Y. Hemaraj · W. Plishker · R. Shekhar · S. S. Bhattacharyya
Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, USA
e-mail: mainak.sen@gmail.com

M. Sen · W. Plishker · S. S. Bhattacharyya
Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA
e-mail: plishker@umd.edu;

Y. Hemaraj · W. Plishker · R. Shekhar
Department of Diagnostic Radiology, University of Maryland, Baltimore, MD 21201, USA
e-mail: yash@umd.edu

M. Sen
Cisco Systems, San Jose, CA, USA

Y. Hemaraj
Texas Instruments, Germantown, MD, USA

R. Shekhar
e-mail: rshekhar@umm.edu

S. S. Bhattacharyya
e-mail: ssb@umd.edu

1 Introduction

Image registration is a fundamental requirement in medical imaging and an essential first step for meaningful multimodality image fusion and accurate serial image comparison. It is also a prerequisite for creating population-specific atlases and atlas-based segmentation. Despite the existence of powerful algorithms and clear evidence of clinical benefits they can bring, the clinical use of image registration remains limited. The slow speed (i.e., long execution time) of fully automatic image registration algorithms especially for 3D images has much do with this lack of clinical integration and routine use.

This paper focuses on image registration algorithms that must be executed under real-time performance constraints. Performance requirements will vary by scenario as different procedures will require different levels of accuracy and speed and use different image sizes. For example, correction during prostate radiotherapy treatment must be very accurate, but it is based on relatively small images and can tolerate a few seconds of registration time. Conversely, using preoperative images to guide needle insertion in the abdomen potentially requires large images be registered in sub-second time to correct for motion due to breathing, while still demanding high accuracy.

Among many approaches that have been developed to tackle this problem, a hardware implementation is one way to speed-up applications over existing software implementations. Such a hardware implementation is presented in [24]. Designing hardware can take significantly longer time compared to software, requiring programmers to work at a level of abstraction lower than the natural level for the application. As implementation decisions are made, designers move further away from the natural application representation. While this is standard practice to achieve high performance implementations on hardware platforms such as FPGAs and ASICs, systematic methods based on precise application modeling abstractions and associated hardware mapping techniques offer designers a faster, more effective path to implementation. Such methods make the design process more structured, while at the same time exposing opportunities for system-level performance optimization.

In this paper, we develop such a structured design methodology in the context of image registration. Our approach starts with capturing the high level algorithm structure through a carefully-designed, coarse-grain dataflow model of computation. It is essential for the dataflow model while being high-level to be also detailed and represent the behavior of the system to the finest granularity so that from this single high level representation, designers can reach various design points in the hardware design space. These design points represent relevant area-performance trade-offs associated with different configurations, as we show later for different input characteristics in image registration. Knowledge about this range of configurations can be used to customize the final implementation based on certain input characteristics. We elaborate on these input characteristics later in the paper.

In some previous work, models of computation such as dataflow graph have been used for hardware code generation. For example, [10, 18, 23], use a restricted form of dataflow graph called synchronous dataflow (explained in Sect. 2.1) to generate RTL. In [22], the authors present a system design approach using a different model, called the Kahn process network model, which can be used to generate RTL code for applications written in MATLAB to be automatically mapped onto FPGA- and microprocessor-based platforms. In [14], the authors propose a heterogeneous system modeling approach based on dataflow graphs for synthesis on FPGAs. This approach focuses mainly on core generation for reusability. Recently in [7], a System C-based design flow for digital signal processing systems was presented where a mixed specification of the system in System C and formal methods could be used to implement the system targeting FPGAs. Our approach differs from the previous efforts in that we develop methods to analyze the coarse-grain dataflow representation (which is dynamic) to systematically provide a hardware implementation that can dynamically optimize its processing structure in response to the particular image registration scenario in which it operates.

Several clinical applications to benefit from the proposed work include whole-body PET/CT registration [20], virtual colonoscopy [3] and image registration tasks involving pre- and intra-operative images in the context of image-guided surgeries [5]. The overall benefits can potentially extend to numerous other applications being developed by researchers worldwide.

In this paper, we combine and enhance our previous work to present an efficient class of dynamic architectures and an associated design flow for image registration algorithms. Our design flow is developed through efficient dataflow-based modeling techniques. We present area and power calculations that characterize our proposed architectures. In terms of modeling methodology, we present useful refinements of our previous work on developing and applying novel data flow-based models and analysis methods of image registration applications [8]. These methods provide a framework for mapping and optimizing these applications onto embedded architectures. Using this framework, we extend our previous work on evaluation of trade-offs between different design points [19] and propose a dynamically reconfigurable architecture for image registration that optimizes its processing structure adaptively based on relevant characteristics of its input. We complete our study by proposing an algorithm for switching between the two proposed architectures, and presenting overall system memory requirements (including external memory) and power estimations for the different configurations.

Our proposed methodology in this paper is relatively generic, and can be applied to other kinds of image processing tasks; further low-level fine-tuning for specific applications can be performed on top of the implementation derived through the more general dataflow-based modeling and analysis approach. Extraction/exploitation of parallelism starting from dataflow languages is a well-studied topic. Going to implementation on an FPGA from a dataflow language is also well-understood in various contexts, as the layout of dataflow applications often maps well to the layout of an FPGA's fabric. What is especially innovative about the approach that we develop in this paper over previous publications is the use of dynamic modeling techniques to target a dynamically reconfigurable system. In particular, we have found a unique application to support the exploration of this topic, and developed an integrated environment of modeling, architecture, and FPGA implementation for this application. While the kernels and dataflow techniques are previously published, the new work that we present in this paper models and analyzes the targeted application to explore development of a dynamically reconfigurable architecture along with a comprehensive view of going from a dataflow description to a working implementation in hardware in the context of an image registration algorithm.

2 Background

2.1 Dataflow modeling

In the dataflow model of computation, an application is represented as a directed graph in which vertices (*actors*) correspond to computational modules, and edges correspond to first-in, first-out buffers that queue data (*tokens*) as it passes between actors. The granularity of dataflow representations can range from fine-grained, where actors represent individual operations such as addition or multiplication, to coarse-grained, where actors typically represent sub-graphs or code segments on the order of 10–100 lines. Dataflow is widely used in the design of signal processing applications because it is an intuitive mode for algorithm designers to work with, and it also exposes high-level application structure that is useful for analysis, verification, and optimization of implementations [1].

The synchronous dataflow (SDF) model [11] has strong compile time predictability properties, and is the most mature form of dataflow for signal processing system design. In SDF, the production and consumption rates of actors—i.e., the numbers of tokens produced and consumed when actors execute—are fixed and known a priori. Therefore, the model can provide guarantees on buffer sizes and produce provable safe schedules. However, the SDF model is highly restrictive for many application areas such as computer vision because the model cannot handle data-dependent rates of data transfer between actors [16].

Various extensions and alternatives to SDF have been developed to provide for more flexible application modeling. For example, a cyclo-static dataflow (CSDF) [2] graph can accommodate multi-phase actors that exhibit different consumption and production rates during different phases, as long as the variations across phases form statically-known, periodic patterns. This provides for more flexibility, but still does not permit data-dependent production or consumption patterns.

More recently, a meta-modeling technique called homogeneous parameterized dataflow (HPDF) [17] was proposed in which actor behavior can be adapted in a structured way through dynamically-adjusted parameter values. While HPDF allows significant flexibility in dynamically changing actor behavior, the restrictions imposed in the model ensure that HPDF subsystems are homogeneous—in terms of the rate at which their constituent actors execute—across any particular level in modeling hierarchy. This permits efficient scheduling and resource allocation for actors, as well as verification of bounded memory requirements and deadlock-free operation, which are useful safety properties to guarantee in embedded hardware and software systems.

HPDF is especially useful because it 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 actors. When HPDF is applied with CSDF modeling for its constituent actors, we refer to the resulting model of computation as HPDF/CSDF. HPDF/CSDF allows for a dynamic number of phases for an actor, along with dynamic production and consumption rates on each phase. However, to satisfy HPDF constraints, the total number of tokens produced by an actor on a given edge ‘*e*’ in a given invocation must equal the total number of tokens consumed by the corresponding invocation of the sink actor of ‘*e*’. A more detailed and formal description of HPDF/CSDF is given in [6]. We use HPDF/CSDF in Sect. 4 to model a mutual information based image registration algorithm.

2.2 FPGA technology

Image registration algorithms have the potential to be mapped onto a variety of parallel platforms [25] for efficient execution. Clusters accelerate highly coarse forms of parallelism [26] and single chip multiprocessors can exploit coarse parallelism and often vector level parallelism. The Cell processor, a single chip multiprocessor designed for high performance computing, has been used to accelerate rigid registration based on mutual information [28]. With such platforms, speed is achieved through utilizing multiprocessing, SIMD techniques, and specialized memory schemes along with processing a subset of the voxels.

GPUs have been increasing their raw computational horsepower at a rate faster than CPUs. With many processing elements, high memory bandwidth, and programmability, they have become an inexpensive target of choice for many computationally intensive applications. GPU architectures are able to achieve such a dense computational power through simple processing elements, structured memory resources, and restricted communication channels. Indeed, graphics processors have proved effective for gradient flow based registration [27], for 3D non-rigid registration using sum-of-squared differences [29], and even examined for heterogeneous platforms tailored to image registration [30]. But to properly exploit the

GPU architecture, application parallelism must match the concurrency in the GPU's pixel processing datapaths while accommodating computation, memory, and IO restrictions. Major obstacles are encountered while mapping parallel applications that does not map well to GPUs compared to FPGAs which are more flexible as targets.

While work on these other platforms explore important acceleration facets of image registration, the speed necessary for real-time multimodal registration is still not available. Consequently, we have explored in this work dataflow level parallelization opportunities that are best suited to hardware platforms such as FPGAs. FPGAs are more flexible than GPUs and have increased in computational power just as quickly which makes it an ideal alternate platform to explore. They present a homogeneous computational "fabric" to the programmer, giving them the freedom to express complex application interaction while still being able to exploit parallelism present in the application.

Fast automatic image registration (FAIR) [4] is such an architecture proposed by Castro-Pareja et al. for accelerated hardware implementation of rigid image registration. FAIR is optimized and fine tuned for partial-volume-interpolation-based image registration by means of pipelining, parallel memory access, and distributed processing. The FAIR project was demonstrated by a proof-of-concept implementation that achieved greater than an order of magnitude speedup for registration of multimodality images (MR, CT and PET) of the human head, PET and CT images of the thorax and abdomen, and 3D ultrasound and SPECT images of the heart [21]. As a demonstration of single modality image registration, FAIR used the accelerated implementation also for registration of pre- and post-exercise 3D ultrasound images of the heart [21].

In this work, we target our hardware optimization frame work to an FPGA device, the Altera StratixII EP2S15F484C5. The major advantages of modern FPGA technology are its flexibility to capture many styles of parallelism and the potential for dynamic reconfiguration of the underlying processing structure. In the context of FPGA implementations, dataflow is especially useful because it effectively exposes application concurrency, and facilitates configuration of and mapping onto parallel resources. With hardware description languages (HDLs) such as Verilog and VHDL, application designers must layout their application to explicitly expose parallelism for it to be utilized. Furthermore the model of execution of HDLs is tied directly to the hardware, which often does not match the native application model. This static description of the application can therefore be cumbersome when exploring different points in the design space.

Formal modeling techniques such as HPDF provide designers with a natural way of describing applications, while implicitly exposing parallelism that may be exploited in a variety of ways. This opens up design space exploration opportunities for meeting different user constraints, and achieving different implementation trade-offs. However, streamlining the use of dataflow technology is challenging because it requires careful mapping of application characteristics into the graphical and actor-based modeling abstractions of dataflow, and because the associated optimization issues, while exposed more effectively for signal processing applications compared to other modeling abstractions, are usually NP-complete to solve exactly [1].

A dataflow representation is suitable for behavioral modeling, structural modeling and mixed behavior-structure modeling. Transformations can be applied to all three types of representations to assess different solutions in the overall design space and focus subsequent steps of the design flow on more favorable solutions.

This paper addresses these challenges for the image registration domain.

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 anatomy from different angles, obtained at different times, and/or using different imaging techniques. Image registration is a key feature for a variety of imaging techniques, and there are two main algorithmic approaches—*linear* and *elastic*. A linear transformation can be approximated by a combination of rotation, translation, and scaling coefficients, while an elastic approach is based on nonlinear continuous transformations, and is implemented by finding correlations among meshes of control points. Our study concentrates on the linear approach. As mentioned earlier, real-time image registration is essential in the medical field for enabling image-guided treatment procedures, and pre-operative treatment planning.

There are many approaches to 3D image registration [13]. But for hardware implementation, a robust, accurate, flexible algorithm that does not require manual feedback is preferred. Algorithms based on voxel (a pixel in 3-d space) similarity fulfill the above criteria better than feature-based approaches [9]. Of them, the most commonly used technique is image registration based on *mutual information* [15]. Mutual information (MI) methods have been shown to be robust and effective for multi-modal images.

3.1 MI-based image registration

Figure 1 represents the algorithmic flow of MI-based image registration. MI-based image registration relies on maximizing 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 so that a one-to-one correspondence is established between the coordinates of the images where they overlap. A cost function based on the mutual information is calculated from the individual and joint histograms. The transformation that maximizes the cost function is viewed as the optimum transformation. The goal of MI-based image registration is then to find this optimal transformation T:

$$T = \arg\max_{T} \mathrm{MI}(\mathrm{RI}(x,y,z), \mathrm{FI}(T(x,y,z)))$$

Here, RI is the reference image, and FI is the floating image (the image that is being registered).

3.2 Computation of mutual information

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

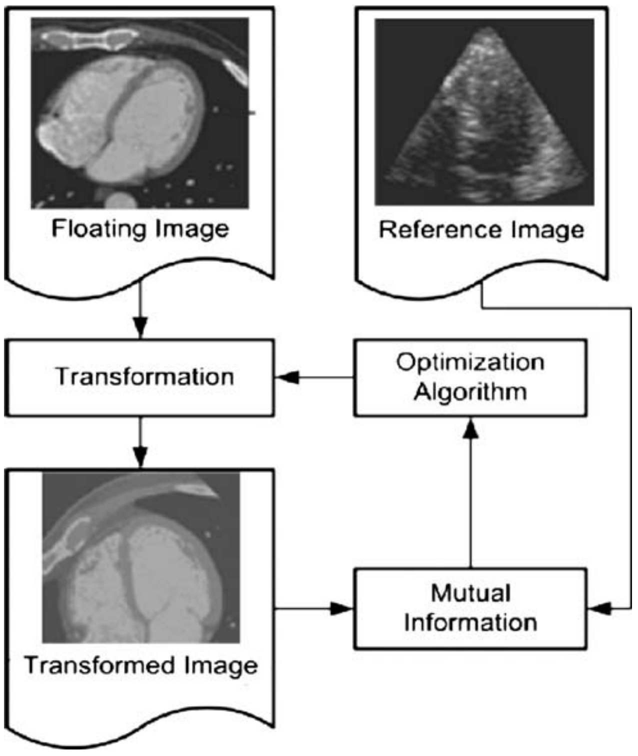


Fig. 1 Mutual-information-based image registration

$$\begin{aligned} MI(\mathrm{RI},\mathrm{FI}) &= H(\mathrm{RI}) + H(\mathrm{FI}) - H(\mathrm{RI},\mathrm{FI}),\\ H(\mathrm{FI}) &= -\sum p_{\mathrm{FI}}(a)\log p_{\mathrm{FI}}(a),\\ H(\mathrm{RI}) &= -\sum p_{\mathrm{RI}}(a)\log p_{\mathrm{RI}}(a), \text{ and}\\ H(\mathrm{RI},\mathrm{FI}) &= -\sum p_{\mathrm{RI,FI}}(a,b)\log p_{\mathrm{RI,FI}}(a,b) \end{aligned} \quad (1)$$

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.

The mutual histogram represents the joint intensity distribu tion. The joint voxel intensity probability, $p_{RI,\,\mathrm{FI}}(a, b)$ 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 obtained from 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 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 (we refer to this as the 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.

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 introduces a new value, which makes the MH sparse and hence ineffective in MI calculation. In [12], it was shown 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 8 interpolation weights directly into the mutual histogram instead of calculating a resultant intensity level and increments 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.

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. Since the size of the mutual histogram is less than the size of the image, it is the first part that is the performance bottleneck.

It has been shown that the size of the mutual histogram can be selected as 64×64 for 8-bit images (i.e., images in which each pixel is represented by 8 bits). By doing so, we can obtain a very good density of MH values, while at the same time pre serving 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. 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 in the adder and 8 writes to write back the updated value after adding the weights). Accesses to the reference image are sequential and standard caching techniques can be effectively 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 do not benefit from memory-burst accesses or memory-caching schemes. The access pattern problem is hard and a case study is interesting in its own merit. Though this problem can be alleviated to some extent using SRAM or RLDRAM (or other such memories with low latency) and hence reduce the latency of the whole system, we have not studied any algorithmic solutions to the problem.

In our study, we instead concentrate on the throughput of the system. Speedup of registration is achieved by identifying throughput bottle neck areas and optimizing them in order to decrease the processing time. Speedup of the algorithm can be obtained by using pipelined architectures and also by using parallel architectures [4]. 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 graph models to describe the inherent concurrency in MI-based image registration, analyze the bottleneck areas from these models, and use high level dataflow graph transformations to exploit potential areas that can be parallelized.

3.3 Optimization

Our targeted image registration algorithm calculates the transformation matrix for which the mutual information between the images is maximum. There are a variety of methods for exploring this optimization space, which vary in speed, convergence, and suitability to certain problems. For example, techniques such as the downhill simplex method provide faster convergence than the others. In the simplex method, in order to optimize a transformation with m parameters, the optimizer needs to store $(m + 1)$ previous values. Despite their variation, they uniformly require some similarity calculation to evaluate potential solutions. Often the results of the similarity calculation serve as feedback to guide the optimizer. Therefore improvements in efficiency and performance of similarity calculations will benefit all optimization approaches. A more comprehensive explanation of such optimizations and references can be found in [15].

4 Application modeling

In this section, we construct a hierarchical dataflow representation of MI-based image registration, and we use the HPDF meta-modeling approach integrated with CSDF (HPDF/CSDF) for modeling lower-level, multi-phase interactions between actors. For all positive n, the number of tokens produced by the nth complete invocation of a source actor must equal the number of tokens consumed by the nth complete invocation of the associated sink actor. Fig. 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

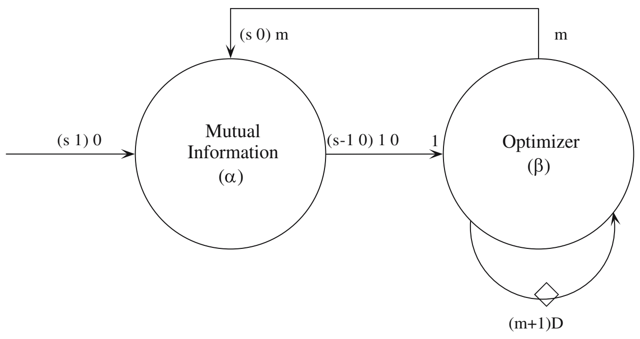


Fig. 2 Top level model of image registration application

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 (an order of execution that can be used to correctly execute the system) for this HPDF graph would be

$$(s\alpha)\beta\alpha \tag{2}$$

In this paper, we describe our schedules as *looped schedules* which is a compact form of representing the execution order of actors (as in Eq. 2). A looped schedule is generally of the form $(nT_1T_2...T_m)$, and such a schedule 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).

The internal representation of the hierarchical MI actor is shown in Fig. 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 ($r \leq s$) 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 "Reference Image" only if a valid voxel results from the transformation on input coordinates. For every input token in D and E, 8 output tokens are produced on both the outgoing edges. The corresponding 8 intensity locations in the "MH Memory" are updated based on the tokens produced by D.

After all coordinates are processed, which occurs during the first *8r phases* of the MH Memory actor or equivalently after s phases of the "Coordinate Transform" actor, one token of size $q \times q$ is sent to the "Decomposer", 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 the equations given in (1). We added the "Decomposer" mainly for ease of representation of the application in dataflow; this module was subsumed by "MH Memory" in the final hardware implementation. A valid schedule for the Mutual Information subsystem based on Fig. 3 would be $(sABC)(rDE(8FG))(q^2ZH)$.

Looking more closely at "Coordinate Transform", we see that it has an additional input edge that takes in the initial m tokens from the "Optimizer" (β) but produces no output. Figure 3 only represents the steady-state behavior of the Mutual Information subsystem for simplicity. Figure 4 represents the initialization and the steady-state

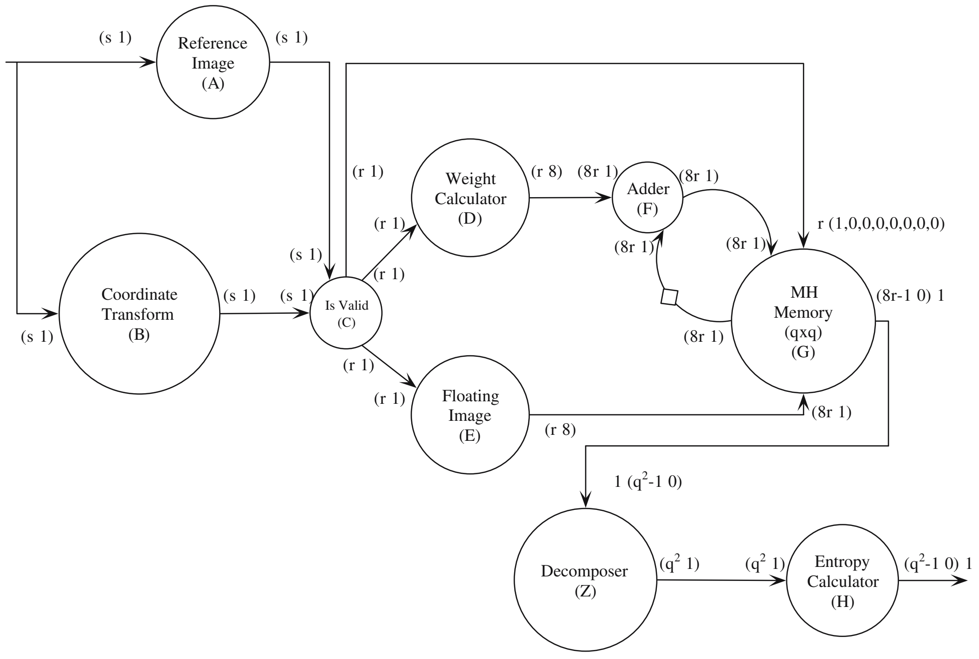


Fig. 3 Dataflow model of mutual information subsystem

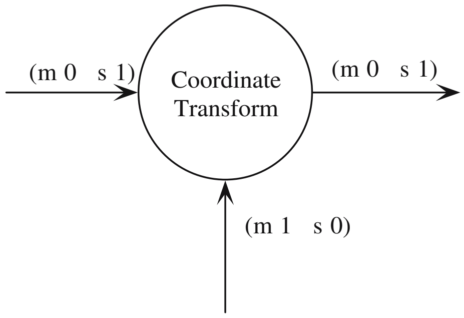


Fig. 4 Initial and steady-state modeling of coordinate transform

behavior of Coordinate Transform, where the initial *m* tokens are used to calculate the new transformation matrix, and hence the values inside the actor are updated without producing any data. A valid schedule of the whole "Mutual Information" subsystem considering the initial and steady-state behavior of "Coordinate Transform" is:

$$(mB)(sABC)(rDE(8FG))(q^2ZH) \tag{3}$$

Figure 5 shows the dataflow model of the "Entropy Calculator". "Row Sum" (*I*) executes once every time it obtains 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 Fig. 5; here we show how one such valid schedule can be derived. Since a valid schedule for "Entropy Calculator" is quite complex, we derive it step-by-step—the graph has three distinct paths, the upper path (involving *Z*, *I*, *J*, *K*) can be executed with the valid schedule $(q(qZ)IJ)K$; the middle path (involving *Z*, *J*, *N*, *O*) can be executed with the valid schedule $(q(q-1)ZL)(qZLN)O$; and the lower part of the graph (involving *Z*, *T*, *U*) can have the valid schedule $(q^2ZT)U$. Combining these, a valid schedule for the "Entropy Calculator" subsystem can be derived as:

$$(q-1(qZLT)IJ)(qZTLN)IJKOUV \tag{4}$$

Combining (3) and (4), the valid schedule for the "Mutual Information" subsystem can be derived as:

$$(mB)(sABC)(rDE(8FG)) \\ \times (q-1(qZLT)IJ)(qZTLN)IJKOUV \tag{5}$$

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

Looped schedules are useful for software code generation from a dataflow graph as every actor appearance in a looped schedule can be replaced by a function call (or inline code), and parenthesized terms can be replaced by software loops to generate complete executable code for an application [1]. Looped schedules are also useful in generating test benches with which different hardware alternatives can be functionally validated.

In addition to leading to looped schedule representations, the model of Fig. 6 shows potential for parallel hardware mapping at various levels of abstraction. For example, extensive parallelism within the processing structure for a single pixel (which we henceforth refer to as "intra-pixel" parallelism) is possible for the MH memory and adder. From Fig. 3, we can see a data-rate mismatch between "Weight Calculator" and "Adder". Similar data-rate mismatch exists between "Floating Image" and "MH Memory". These naturally suggest a parallel architecture, as shown in Fig. 6, where multiple copies (8 in the illustration as the data-rates are mismatched by a factor of 8) of an actor can be created for an intra-pixel parallel implementation. We also note that the resultant graph in Fig. 6 becomes HPDF as all the parameterized actors now have the same production and

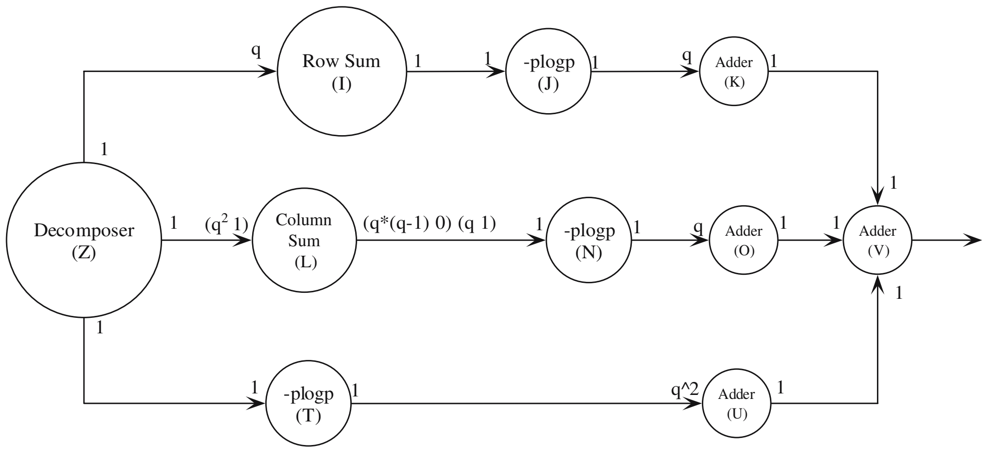
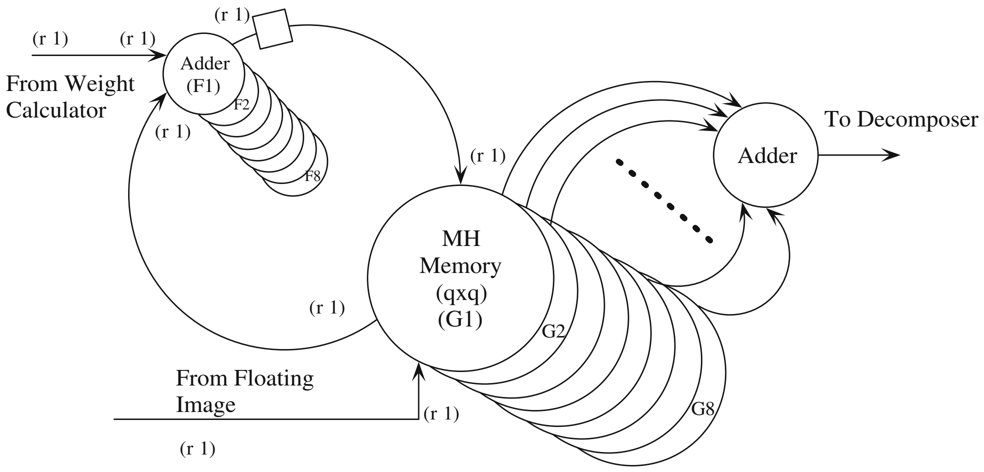


Fig. 5 "Entropy Calculator" where *q* depends on the number of bits used to represent each pixel in the input image. The value of *q* is 64 for 8-bit images

Fig. 6 Parallel architecture for MH update showing intra-pixel parallelism



consumption rates and hence fire at the same rate. The data flow model in Fig. 3 also exposes parallelism among the processing structures of any two pixels, as input actors *A* and *B* have *s* distinct phases without any interdependency amongst them where *s* is the number of pixels in the image. This (which we henceforth refer to as "inter-pixel" parallelism) leads to another set of useful parallel implementation considerations.

Based on these insights, we develop in Sect. 8 an architecture that applies both intra- and inter-pixel parallelism, and balances these forms of parallelism adaptively in response to relevant input characteristics.

5 Actor implementation

The lowest level (non-hierarchical) actors in our dataflow-based design are implemented in Verilog. As an illustration of a Verilog-based actor in our design, Fig. 7 shows the code corresponding to the Adder actor (*F* in Fig. 3). An interesting point to note in this code example is that by analyzing the dataflow behavior, we can ensure that the interface code between the adder and the weight calculator places the correct weight at every clock cycle in the input buffer labeled 'weight'. This illustrates how using dataflow as a high-level modeling abstraction helps to structure the hardware implementation process, and makes the hardware description language (HDL) code modular and more reliable.

In our overall implementation, we used a one-to-one mapping in hardware from each dataflow graph actor except for the "Decomposer" module, which, as described in Sect. 4, was subsumed inside "MH Memory" for efficient implementation. We have verified the correctness of our design and hardware description language implementation through functional simulation.

```
/* global definitions in top.v */
reg [imsize+fracwidth-1:0] mh [0:4096];
reg [imsize+fracwidth-1:0] edgeweights [0:numweights-1];

/*one example module */
module mhupdate
#(parameter imsize = 8,
parameter fracwidth = 8,
parameter numweights = 8,
parameter lognumweights = 3)
(input [imsize-1:0] rival,fival,
input [imsize+fracwidth-1:0] weight,
input clk);
reg [11:0] curval;
reg [lognumweights:0]counter;

always @(posedge resetall)
   counter <= 0;

always @(posedge clk)
   begin
      if(counter < numweights) begin
         mh[curval] <= mh[curval] + weight;
         curval[5:0] <= rival[imsize-1:imsize-6];
         curval[11:6] <= fival[imsize-1:imsize-6];
         counter <= counter + 1;
      end
      else
         counter <= 0;
   end
endmodule
```

Fig. 7 Example verilog code (partial) of the adder from Fig. 3

6 Experimental setup

We explored in detail the effect of having a parallel architecture on the targeted application, as suggested by the application-level dataflow model. In our experimental setup, we varied the degree of parallelism, and studied the relationship between the performance and area of the implementation. We also found 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.

6.1 Application parallelism

When the "Floating Image" 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 mutual histogram memory), it takes 8 firings of this set of actors (corresponding to the values of the 8 neighborhoods) 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) neighbor hood(s) each. This would mean that the number of firings of each set of actors becomes 4, 2, (or 1), respectively. These sets of actors can fully execute in parallel without any dependency. The dependency in the mutual histogram memory is removed by allowing eight copies of it to be updated independently, and the result for each location in the memory is obtained by adding each of the corresponding locations in the eight copies of the table at the end. 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. Memory requirements also increase with increasing image size. In addition, there is a cost of interfacing these external memories that needs to be addressed. Each memory component comes with a latency that adds to the processing time.

6.2 Relationship between PVV and performance

When the transformed coordinate falls in the valid region, there are eight firings of the actor set ("Adder", "MH Memory" in Fig. 3). However, when "Is valid" 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 the corresponding input coordinates, and the next token is processed by the coordinate transform actor, indicating a new iteration. In our implementation, when an invalid voxel coordinate is generated for the first time, there is a two-cycle penalty for filling the pipeline (as we have to propagate the invalid signal through "Weight Calculator" and "Adder"); however, the penalty is only one clock cycle for every subsequent invalid signal (as now, we already have the two relevant actors filled with the invalid signal).

We explore performance-area trade-offs for different PVV values in Sect. 8.

7 Results and explanation

To validate our claims about the benefits of this design approach, we collected hardware synthesis results for various proposed configurations of our targeted image registration application. Ideally we would compare these results to a variety of acceleration platforms including GPUs. However, comparisons to existing GPU approaches are not appropriate yet because they use algorithms different than those used here. GPUs in particular are yet to be utilized efficiently for mutual information based registration. Our results are obtained using the Quar tusII synthesis tool from Altera for the StratixII family of FPGAs (StratixII EP2S15F484C5). Table 1 presents the synthesis results we obtained for various configurations. Here, the columns represent different numbers of parallel datapaths for the MH Update actor, and the rows represent (from top to bot tom) the amount of external memory required for the system, the amount of logic circuitry used in the FPGA for the MI actor, the number of DSP elements (specialized, coarse-grain hardware modules on the FPGA for signal processing) used by the circuit for the MI actor, the total number of ALUTs (adaptive lookup tables) used in the FPGA for the MI actor, and the maximum frequency of operation of the circuit representing MI. The amount of required external memory increases with increasing numbers of parallel data-paths due to multiple corresponding copies of the "MH Memory" module.

The adaptive logic module (ALM) is the basic building block of the Altera StratixII FPGA. Each ALM contains a variety of LUT (look-up table)-based resources, two full adders, carry-chain segments, and two flip-flops. Each ALM can be adaptively divided into two adaptive LUTs (ALUTs).

The "LC Registers" (logic cell registers) row in Table 1 represents the total number of registers used, and row labeled "Total FPGA Area" gives an idea of the available

Table 1 Synthesis results for the overall system for different configurations of the MH update actor

Number of parallel datapaths	1	2	4	8
External memory	256 Kb	512 Kb	1 Mb	2 Mb
LC registers in FPGA	427	576	871	1463
DSP elements	30	30	30	30
Total FPGA area (number of ALUTs)	598	878	1,439	2,588
Max freq of operation (MHz)	74	72.2	74	70.1

resources in the FPGA that is used. From Table 1 we can see that both of these figures of merit increase as the number of data-paths increase. However, the number of DSP elements used remain almost constant as the DSP elements are required in the Coordinate Transform actor ('B' in Fig. 3), which is not replicated with replication of datapaths. Table 2 is independent of the PVV as the PVV only affects the runtime of the circuit.

Next, we simulated the performance of the various configurations of the circuit with four different PVV values—100%, 90%, 50% and 10%. This simulation was carried out in terms of the required number of clock cycles. We observed that when the PVV value is low, invalid signals are dense, and conversely, that invalid signals are sparse when the PVV value is high. This has a bearing on the performance, as mentioned in Sect. 6.2.

Figure 8 shows the area (measured by the number of adaptive logic units in the circuit without considering the external memory) and performance (measured by the number of execution cycles) as we vary (1) the number of parallel datapaths in the MH update actor and (2) the PVV. In Fig. 8, we see that the number of clock cycles (and hence execution time) decreases with increasing number of parallel datapaths for any particular PVV and the relationship is almost linear as the datapaths can execute independently of each other. On the other hand, the number of clock cycles decreases when the PVV decreases without any change in the architecture. Hence, the throughput of the system is a function of both the number of parallel datapaths and the PVV. However, keeping in mind that the PVV is input-dependent and architecture-independent, we further note that the relative change in the number of clock cycles for increasing data-path counts is dependent on the PVV.

Building on this key observation, we propose a PVV-based dynamically-reconfigurable FPGA implementation in Sect. 8. For a more complete overview of the different configurations explored, we present in Fig. 9 a full-system area estimation with consideration included for external memory. Even though we do not foresee image registration being used in embedded platforms in the near future; power has become a concern even in FPGAs [31]. Also, power, in addition to area, can be considered as a cost metric in the cost performance analysis of the various configurations of our system. Hence we measured the dynamic power

Table 2 Comparison of power consumption of circuit under different datapath configurations

Number of parallel datapaths	1	2	4	8
Power for logic (mW)	4	15	25	35
Dynamic power for FPGA (mW)	92	115	147	159

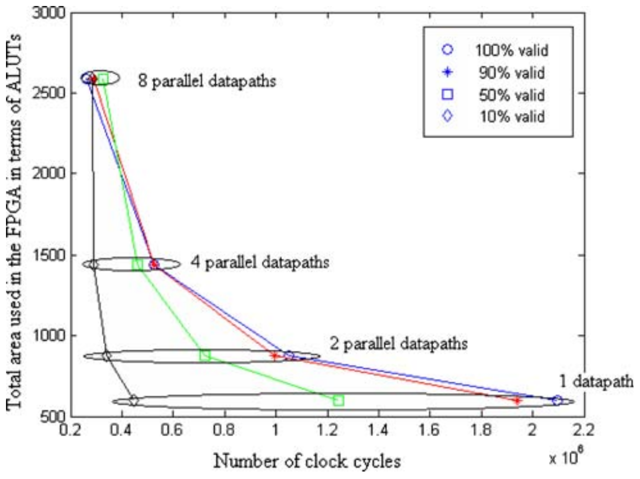


Fig. 8 Area vs. clock cycles for different PVV values and different numbers of datapaths

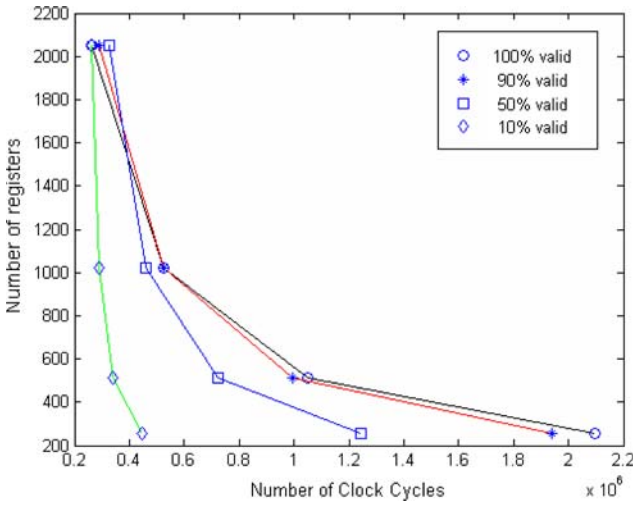


Fig. 9 Overall system memory requirements in bytes, including external memory

dissipation of the FPGA for both the logic part and the full circuit, including RAM, I/O power, DSP elements, and clocks (without considering external memory, as that would depend on the physical board on which the application is finally implemented). These measurements are summarized in Table 2. As expected, we see an increase in the power consumption as the number of parallel datapaths increase.

8 Dynamic reconfiguration

Based on the discussion in the last section, we see that there are two parameters to determine the execution time of our system–namely the PVV and number of parallel datapaths. However, they are not independent of each other, as the relative change in execution time with changing datapaths is dominated by the PVV. For example, we see that for 100% PVV, the gain in execution time with eight datapaths (compared to a single datapath) is around 0.05×10^6 clock cycles, whereas the gain is almost 2×10^6 clock cycles when the PVV drops to 10%. So intuitively, for high PVV, the overhead of dedicating hardware to expose parallel datapaths and decrease the processing time for one output from the Coordinate Transform in Fig. 3 (which we have been referring to as intra-pixel parallelism) may be better utilized by distributing the hardware overhead to process multiple pixels in parallel (which we have been referring to as inter-pixel parallelism).

In this section, we compare a *multiple one-voxel/one-memory* architecture against a *single one-voxel/eight-memory* architecture; we develop a more flexible, *dynamic reconfigurable image registration architecture*; and present results from our design of this new architecture. In this development, we compare and adaptively apply inter-pixel parallelism and intra-pixel parallelism in the forms in which these were exposed by our dataflow-based design (Sect. 4).

Based on our results in Sect. 7, we see that the PVV is input-dependent. Furthermore, we see that as the PVV increases, the run-time increases and memory access becomes more and more of a bottleneck. Gradually, it becomes more performance-effective to trade-off inter-pixel parallelism in the architecture for intra-pixel parallelism in the form of multiple (parallel) memories that alleviate the memory bottleneck. This trend is demonstrated by the data in Table 3, which compares the performance, for different PVV values, of a 1-voxel/8-memory architecture (intra-pixel parallelism) to a 7-voxel architecture with 1 memory module per voxel (inter-pixel parallelism). The value of 7 is selected here because for the targeted FPGA device, the area of a 1-voxel/8-memory architecture is approximately 7 times that of a 1-voxel/1-memory architecture. The units of performance in Table 3 are nano seconds per voxel per co-ordinate transform, and the frequencies of operation of the different memory architectures vary between 70 and 74 MHz for the various configurations.

We note that in Table 3, considering the given area constraint, the performance of a 1-voxel/1-memory architecture is better than that of a 1-voxel/8-memory architecture. However, this trend changes as the voxel validity percentage increases. Therefore, our image registration architecture monitors the PVV metric at run-time and dynamically reconfigures the architecture from inter-pixel parallelism mode to intra-pixel parallelism mode once a transition point of around 50% PVV is observed. In order to prevent rapid change in architecture in case the PVV oscillates around 50%—which would result in thrashing behavior—we assign a threshold *T* such that the architecture gets reconfigured when a $(50 - T)\%$ PVV state is followed by a $(50 + T)\%$ PVV state or vice-versa. *T* can be set by the user depending on image characteristics such that the dynamic reconfiguration happens only if necessary in terms of performance. This dynamic architecture can be viewed as an (ideally) once-per-image, PVV-driven re-scaling of the subsystem shown in Fig. 6.

Table 3 Comparison of intra-versus inter-pixel parallelism modes for different PVV values

PVV (%)	Performance of 7 1 voxel-1 memory	Performance of 1 1voxel-8 memory
10	6.39/7 = 0.91	2.54
50	17.8/7 = 2.54	2.91
90	27.82/7 = 3.97	2.5
100	30.08/7 = 4.29	2.33

The units of performance in Table 3 are nano-seconds per voxel per co-ordinate transform

As described above, our proposed architecture monitors the PVV at run-time and dynamically reconfigures itself between the inter-pixel and intra-pixel parallel modes when the PVV crosses a certain threshold. Note that the optimal transition point is in general image-dependent, and our use of a fixed value of $(50 \pm T)\%$ as a transition point is therefore a heuris tic approach. Dynamically determining the transition point is a useful topic for further investigation.

The reconfiguration cost is another important aspect of this design. The overhead of reconfiguring the system can be minimized by incorporating a simple block of logic that switches between the two designs. The goal of such reconfiguration is to be able to keep as many actors common as possible between the two designs to reduce overhead. Actors such as coordinate transform, reference image and optimizer in Fig. 3 can be reused across different configurations. Changes with respect to the production and consumption rates are required for the floating image and weight calculator during reconfiguration.

For our current implementation, we have assumed that each set of images to be registered is independent of the next set of images, and hence we always start with a 1-voxel/1-memory architecture. An interesting direction for future work is to explore further optimization when correlations among different sets of images are known a priori.

9 Conclusion

In this paper, we have presented a structured design approach towards implementation of an image registration algorithm onto an FPGA for real-time constraints. We have captured the inherent concurrency of the application at the inter-pixel and intra-pixel levels by modeling it through the framework of homogeneous parameterized dataflow. We have also presented some dataflow-motivated parallel architectures for image registration, and presented a detailed study of area/performance trade-offs for these architectures. Based on the results obtained, we have also presented the design and FPGA mapping of an architecture for dynamically-reconfigurable image registration. We have demonstrated the ability of the architecture to strategically adapt its parallel processing configuration in response to relevant image characteristics, and for this purpose, we have formulated the PVV metric, which represents the percentage of valid voxels that results from a transformation on the given floating image.

References

1. Bhattacharyya, S.S., Leupers, R., Marwedel, P.: Software synthesis, code generation for DSP. IEEE Trans. Circuits, Syst.-II. Analog Digital Signal Process **47**(9), 849–875 (2000)
2. Bilsen, G., Engels, M., Lauwereins, R., Peperstraete, J.A.: Cyclo-static dataflow. IEEE Trans. Signal Process **44**(2), 397–408 (1996)
3. Castro-Pareja, C.R., Daly, B., Shekhar, R.: Elastic registration using 3D chainmail. In: Proceedings of the SPIE (Medical Imaging) (2006)
4. Castro-Pareja, C., Jagadeesh, J.M., Shekhar, R.: FAIR: a hardware architecture for real-time 3-d image registration. IEEE Trans. Inf. Technol. Biomed. **7**(4), 426–434 (2003)
5. Dandekar, O., Walimbe, V., Siddiqui, K., Shekhar, R.: Image registration accuracy with low-dose CT: how low can we go? In: Proceedings of the IEEE International Symposium on Biomedical Imaging, pp. 502–505 (2006)
6. Haim, F., Sen, M., Ko, D., Bhattacharyya, S.S., Wolf, W.: Mapping multimedia applications onto configurable hardware with parameterized cyclo-static dataflow graphs. In: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, pp. III-1052–III-1055, May 2006
7. Haubelt, C., Falk, J., Keinert, J., Schlichter, T., Streub, M., Deyhle, A., Hadert, A., Teich, J.: A system C-based design methodology for digital signal processing systems. EURASIP J. Embedded Syst. Article ID 47580, 22, (2007)
8. Hemaraj, Y., Sen, M., Shekhar, R., Bhattacharyya, S.S.: Model-based mapping of image registration applications onto configurable hardware. In: Proceedings of the IEEE Asilomar Conference on Signals, Systems, and Computers, October 2006
9. Holden, M., Hill, D., Denton, E., Jarosz, J., Cox, T., Rohlfing, T., Goodey, J., Hawkes, D.: Voxel similarity measures for 3D serial MR brain image registration. IEEE Trans. Med. Imaging **19**, 94–102 (2000)
10. Horstmannshoff, J., Meyr, H.: Efficient building block based RTL code generation from synchronous data flow graphs. In: Proceedings of the Design Automation Conference (2000)
11. Lee, E., Messerschmitt, D.: Synchronous data flow. In: Proceedings of the IEEE, September 1987
12. Maes, F., Vandermeulen, D., Suetens, P.: Medical image registration using mutual information. Proc. IEEE **19**, 1699 (2003)
13. Maintz, J.B., Viergever, M.: A survey of medical image registration. Med. Image Anal. **2**(1), 1–36 (1998)
14. McAllister, J., Woods, R., Walke, R., Reilly, D.: Multidimensional DSP core synthesis for FPGA. J. VLSI Signal Process Syst. Signal Image Video Technol. **43**(2–3) (2006)
15. Pluim, J.P.W., Maintz, J.B.A., Viergever, M.A.: Mutual information based registration of medical images: a survey. IEEE Trans Med Imaging **22**(8), 986–1004 (2003)
16. Sen, M., Corretjer, I., Haim, F., Saha, S., Schlessman, J., Bhattacharyya, S.S., Wolf, W.: Computer vision on FPGAs: design methodology and its application to gesture recognition. In: Proceedings of the IEEE Workshop on Embedded Computer Vision, pages CD-ROM version, San Diego, pp. 8, June 2005
17. Sen, M., Bhattacharyya, S.S., Lv, T., Wolf, W.: Modeling image processing systems with homogeneous parameterized dataflow graphs. In: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, pp. V-133–V-136, March 2005
18. Sen, M., Bhattacharyya, S.S: Systematic exploitation of data parallelism in hardware synthesis of DSP applications. In: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, pp. V-229–V-232, May 2004
19. Sen, M., Hemaraj, Y., Bhattacharyya, S.S., Shekhar, R.: Reconfigurable image registration on FPGA platforms. In: Proceedings of the IEEE Biomedical Circuits and Systems Conference, London, pp. 154–157, November 2006
20. Shekhar, R., Walimbe, V., Raja, S., Zagrodsky, V., Kanvinde, M., Wu, G., Bybel, B.: Automated three-dimensional elastic registration of whole-body PET and CT from separate or combined scanners. J. Nucl. Med. **46**(9), 1488–1496 (2005)
21. Shekhar, R., Zagrodsky, V., Castro-Pareja, C.R., Walimbe, V., Jagadeesh, J.M.: High-speed registration of three- and four-dimensional medical images by using voxel similarity. Radiographics **23**(6), 1673–1681 (2003)
22. Stefanov, T., Zissulescu, C., Turjan, A., Kienhuis, B., Deprettere, E.: System design using Kahn process networks: the Compaan/Laura approach. In: Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, February 2004
23. Williamson, M.: Synthesis of parallel hardware implementations from synchronous dataflow graph specifications. Ph.D. thesis, University of California at Berkeley, May 1998
24. Zitová, B., Flusser, J.: Image registration methods: a survey. Image Vis. Comput. **21**(11), 977–1000 (2003)
25. Plishker, W., Dandekar, O., Bhattacharyya, S.S., Shekhar, R.: A taxonomy for medical image registration acceleration techniques. In: Proceedings of the IEEE-NIH Life Science Systems and Applications Workshop, Bethesda, pp. 215–218 November 2007
26. Ino, F., Ooyama, K., Hagihara, K.: A data distributed parallel algorithm for nonrigid image registration. Parallel Comput **31**, 19–43 (2005)
27. Koehn A., Drexl H., Ritter F., Koenig M., Peitgen H.-O.: GPU Accelerated image registration in two and three dimensions. In: Informatik Aktuell. Springer, Berlin (2006)
28. Ohara, M., Yeo, H., Savino, F., Iyengar, G., Gong, L., Inoue, H., Komatsu, H., Sheinin, V., Daijavad, S., Erickson, B.: Real-time mutual-information-based linear registration on the cell broadband engine processor. In: presented at 4th IEEE International Symposium on Biomedical Imaging, Arlington, 2007
29. Köhn, A., Drexl, J., Ritter, F., König, M., Peitgen, H. O.: GPU accelerated image registration in two and three dimensions. In: Handels, H., Ehrhardt, J., Horsch, A., Meinzer, H.-P., Tolxdorff, T. (eds.) Bildverarbeitung für die Medizin. Informatik aktuell, pp 261–265. Springer, Berlin (2006)
30. Plishker, W., Dandekar, O., Bhattacharyya, S.S., Shekhar, R.: Towards a heterogeneous medical image registration acceleration platform. In: Proceedings of the IEEE Biomedical Circuits and Systems Conference, Montreal, pp. 231–234, November 2007

31. Kevin, M.: Power, suddenly we care. In: FPGA and Programmable Logic Journal, April 2005. http://www.fpgajournal.com/articles_2005/pdf/20050426_power.pdf

Author Biographies

Mainak Sen received his Ph.D at the Electrical and Computer Engineering Department at the University of Maryland, College Park in November 2006. He received his B. Engineering degree (with honors) in Computer Science and Engineering from Jadavpur University, India in 2001. His research interests are model-based hardware design, modeling of dynamic applications using dataflow graphs, hardware\software co-design. He has worked extensively on mapping of image processing algorithms onto FPGAs. He is currently working at Cisco Systems in San Jose, CA, USA.

Yashwanth Hemaraj received his Masters in Electrical and Computer Engineering from University of Maryland in June 2007 and his B.S in Electronics and Communication Engineering from National Institute of Technology Karnataka, Suratkal, India. He is currently working at Texas Instruments, Germantown, Maryland. His research interests include DSP algorithms, Image, Video and Speech processing algorithms.

William Plishker is a Postdoctoral research fellow with a dual appointment with University of Maryland, College Park and Department of Radiology, University of Maryland School of Medicine. He received his Ph.D in Electrical Engineering from University of California, Berkeley in 2006 and B.S in Computer Engineering from Georgia Institute of Technology in 2001. His research interests include design automation techniques for programmable embedded systems.

Raj Shekhar is an Assistant Professor of Diagnostic Radiology, Bioengineering, and Electrical and Computer Engineering at the University of Maryland, Baltimore and College Park. He previously served as a Staff Scientist at the Cleveland Clinic and as a Senior Engineer at Picker International (now Philips Medical Systems). Dr. Shekhar received his doctorate in Biomedical Engineering from the Ohio State University in 1997. Dr. Shekhar's research interests are medical image processing, real-time computing, 3D ultrasound, and image-guided interventions. Dr. Shekhar has authored over 50 scientific papers, including over 20 peer-reviewed articles. He also holds three US patents.

Shuvra S. Bhattacharyya is a Professor in the Department of Electrical and Computer Engineering, University of Maryland at College Park. He holds a joint appointment at the University of Maryland Institute for Advanced Computer Studies (UMIACS). Dr. Bhattacharyya is coauthor or coeditor of four books and the author or coauthor of more than 100 refereed technical articles. His research interests include VLSI signal processing, embedded software, and hardware/software co-design. He received the B.S. degree from the University of Wisconsin at Madison, and the Ph.D. degree from the University of California at Berkeley. Dr. Bhattacharyya has held industrial positions as a Researcher at the Hitachi America Semiconductor Research Laboratory (San Jose, California), and Compiler Developer at Kuck & Associates (Champaign, Il, USA).