A Framework for Customizable FPGA-based Image Registration Accelerators

Davide Conficconi
davide.conficconi@polimi.it
DEIB, Politecnico di Milano, Italy

Eleonora D'Arnese
eleonora.darnese@polimi.it
DEIB, Politecnico di Milano, Italy

Emanuele Del Sozzo
emanuele.delsozzo@polimi.it
DEIB, Politecnico di Milano, Italy

Donatella Sciuto
donatella.sciuto@polimi.it
DEIB, Politecnico di Milano, Italy

Marco D. Santambrogio
marco.santambrogio@polimi.it
DEIB, Politecnico di Milano, Italy

ABSTRACT

Image Registration is a highly compute-intensive optimization procedure that determines the geometric transformation to align a *floating* image to a *reference* one. Generally, the registration targets are images taken from different time instances, acquisition angles, and/or sensor types. Several methodologies are employed in the literature to address the limiting factors of this class of algorithms, among which hardware accelerators seem the most promising solution to boost performance. However, most hardware implementations are either closed-source or tailored to a specific context, limiting their application to different fields. For these reasons, we propose an open-source hardware-software framework to generate a configurable architecture for the most compute-intensive part of registration algorithms, namely the similarity metric computation. This metric is the *Mutual Information*, a well-known calculus from the Information Theory, used in several optimization procedures. Through different design parameters configurations, we explore several design choices of our highly-customizable architecture and validate it on multiple FPGAs. We evaluated various architectures against an optimized MATLAB implementation on an Intel Xeon Gold, reaching a speedup up to 2.86×, and remarkable performance and power efficiency against other state-of-the-art approaches.

CCS CONCEPTS

• **Hardware** → **Hardware accelerators**; • **Computer systems organization** → **Reconfigurable computing**; • **Applied computing** → Health informatics.

KEYWORDS

Image Registration; Domain Specific Accelerator; Mutual Information; Reconfigurable computing; FPGAs

1 INTRODUCTION

Image processing is pervading our day-life being the basis of many application fields, such as photography, navigation, and medical practice. In the clinical domain, image analysis is significantly increasing for brain activity analysis, cancer identification, and surgery/treatment planning [1, 2, 34]. All these applications require the processing of hundreds of images, with multiple algorithms, in the shortest time frame. Although several improvements have been introduced to optimize compute-intensive standard algorithms, there is still room for improvement in the advanced ones. A clear example is *image registration.*

Image registration is the process of identifying the parameters of the geometrical transformation matrix that allows the correct overlap of two or more images acquired in different conditions or time instants [6]. It is widely employed not only in the medical field [26] but also in target recognition [37], satellite imaging [20], and remote sensing [5]. Even though this task is divided into mono- and multi-modal solutions (input images from one or different sensors), it always relies on a three-block structure: the transformation model, the optimization method, and the similarity metric [32]. The transformation model is clustered in rigid (rigid allowing translation and rotation, and affine, which adds scaling and shear factors) and non-rigid, which allows object deformation [16]. The optimizer searches the transformation space to find the optimal parameters of the geometric transformation. Among the several optimizers, Simplex and Powell's methods are widely spread approaches, thanks to their gradient-free feature that reduces the computational complexity [28]. Specifically, Simplex optimizes all the parameters at the same time, while Powell's method optimizes one parameter at a time [28]. Another commonly employed family of algorithms are the evolutionary ones, which optimize the parameters in the searching space based on a probability function [31]. Finally, the most used similarity metric in multi-modal registration, commonly incorporated in a cost function, is the Mutual Information (MI) [30, 33] due to its robustness and reliability [4].

The achievement of a correct registration is strictly correlated to multiple iterations of the three blocks, where the similarity metric has proven to be the most compute-intensive, working directly with all the data contained in the employed images [29]. Therefore, researchers proposed various approaches exploiting different hardware solutions [13, 28] to accelerate either part or the entire algorithm [8, 14]. Unfortunately, even though improvements have been done, the majority of the available solutions are closed-source and, generally, tailored to a specific scenario, highly reducing, if not completely preventing, the users from customizing them.

Within this context, this work proposes a completely open-source hardware-software framework for multi-modal image registration. The proposed framework automates the design and synthesis of a customizable FPGA accelerator that targets the most compute-intensive part of image registration, namely the MI calculus. Thanks to the various customization parameters the framework exposes, the user can quickly explore the design space, tune the features of the MI accelerator, and tailor it to multiple case studies with different requirements. On the other hand, our solution offers high-level APIs based on the PYNQ framework [35] to easily

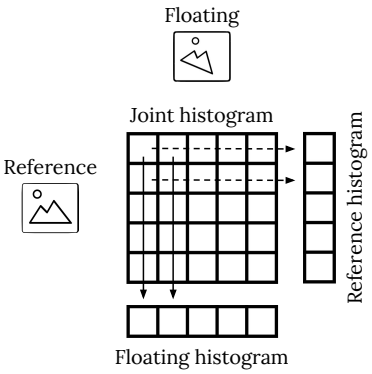

Figure 1: The relationship between joint and single histograms

integrate the accelerator within Python applications. We evaluated various versions of our accelerator on multiple FPGAs achieving a speedup up to 2.86× against an optimized MATLAB implementation, and remarkable results in terms of both performance and power efficiency against other state-of-the-art hardware and software approaches. This enables our accelerator to be suitable for both embedded and high-end FPGA-based devices.

The main contributions of this work are the following:

- A completely open-source hardware-software framework [11] featuring the automation of both design and synthesis of hardware design for image registration;
- A highly customizable FPGA-based accelerator for MI calculation along with Python APIs for transparent exploitation;
- A framework to support the exploration of different design parameters to tailor the proposed accelerator according to the case-specific requirements of the final user.

The remainder of the paper is organized as follows: Section 2 provides the theoretical background, while Section 3 proposes an overview of literature solutions. Section 4 details the proposed solution, Section 5 presents the experimental setup and the validation of the work, and, finally, Section 6 concludes the paper.

2 BACKGROUND

This Section gives a high-level description of the registration process and a description of the theoretical principles behind the proposed accelerator for mutual information computation. With this discussion, we aim at explaining all the properties and structures that will be exploited in the design of the FPGA-based accelerator.

2.1 Image Registration

Image registration is a highly employed procedure in various fields and, therefore, different implementations have been proposed during the years. This procedure processes data ranging from satellite images of the Earth to medical images both anatomical and functional [29]. Based on the image types, we can distinguish between mono- and multi-modal registration; the former works with images taken from the same device, while the latter employs images taken from different sensors [6]. An additional distinction can be done between feature-based and intensity-based approaches [33]. The first one requires the identification of relevant features from the images, giving a higher visual certainty of the reached correspondence, but it is not applicable when the landmark points identification is not trivial [6], as in multi-modal applications. Intensity-based algorithms exploit heuristic solutions that compare the intensity distribution of the images and decide whether or not the images are correctly aligned according to a similarity metric [33]. In this work, we consider a multi-modal intensity-based registration solution that can exploit either the (1+1) Evolutionary or Powell's optimization methods with the MI similarity metric to find the optimal parameters of the affine transform between medical images.

2.2 Mutual Information

A wide range of algorithms employs MI quantity. In imaging, it is an essential similarity metric for image registration [23], in the genomics field it is exploited in phylogenetic [21] and relevance networks [7], as well as in the training of Hidden Markov Models and features selection [3, 15].

MI is a concept borrowed from the Information Theory that relates to the concept of entropy, and it is a measure of the statistical dependence of two random variables X and Y [12]. In the proposed scenario, the two variables are represented by images, where we can identify a *reference* and a *floating* one. In particular, we want to align the floating image to the reference image. From a mathematical point of view, MI describes how similar the joint entropy $H(X,Y)$ is to the two single entropies $H(X)$ and $H(Y)$, as reported in eq. (1).

$$MI(X,Y) = H(X) + H(Y) - H(X,Y) \tag{1}$$

Therefore, an essential step is the definition and computation of the different entropies, as defined by the Shannon's equation eq. (2), where $P(x)$ and $P(y)$ are the marginal probabilities, and $P(x,y)$ is the joint probability.

$$\begin{aligned} H(X) &= -\sum_{x \in X} P(x) \log P(x) \\ H(X,Y) &= -\sum_{x,y \in X,Y} P(x,y) \log P(x,y) \end{aligned} \tag{2}$$

In the imaging field, the probabilities come from the histograms of the images, while the joint one from the joint histogram. Starting from the input images, we compute the joint histogram, which is a square matrix of $N \times N$, where N is the number of gray levels in the images [17]. The value of each element of the joint histogram $hist(x,y)$ is equal to the total number of voxels of X with intensity x corresponding to the voxels of Y with intensity y [12].

As shown in Fig. 1, it is possible to exploit the joint histogram to efficiently obtain the single histograms of the input images. Indeed, by summing the rows and the columns of the joint histogram we obtain the reference and the floating histograms, respectively [12]. For the entropy calculation, we need to obtain the marginal and joint probabilities, which can be easily extracted by dividing each value of the single and joint histograms by the image dimensions in voxels. Referring to eq. (2), the last step is to apply the equations, hence, to multiply each probability by its logarithmic value, and, by accumulating them, we obtain the entropies. Finally, we combine all the entropy values to extract the MI, as in eq. (1).

3 RELATED WORKS

This Section contains an overview of the current literature with an in-depth focus on hardware-based solutions for multi-modal registration, being the scope of the proposed case study. In a pure software scenario we should mention SimpleITK [22, 36], OpenCV, and the MATLAB Image Processing Toolbox [24]. While the first two are open-source, the last one is a licensed closed-source product. On the other hand, MATLAB is easy to use, while OpenCV provides fewer functionalities for image registration compared to the others, and SimpleITK, even though it can be easily used with Python, provides, like all the others, little control on small details. The literature contains several works exploring FPGA- and GPU-based solutions for multi-modal image registration to overcome the limits of pure CPU implementations [28]. From an algorithmic point of view, based on [14, 29], it is possible to conclude that the most compute-intensive part is typically the calculus of the similarity metric. For this reason, in [29], the authors develop an FPGA-based accelerator to compute the similarity metric, namely the correlation, and the transformation model, an affine one, to register iris eye images through the Simplex optimizer. Besides, [14] proposes an FPGA-based approach, based on [8], to compute the MI value for multi-rigid registration, with a single computation of the MI for 7-bit 256×256 images taking around 0.26 seconds. The authors explored such an approach through an extensive design space exploration in [13]. On the other hand, [9] exploits GPUs to accelerate the sole joint histogram for brain images registration, based on MI, with a presorting strategy of the pixels. In [27], authors present a 3D MI-based image registration algorithm, using bitonic sort and count, which they tailor for GPU to achieve the best performance out of rigid transformation and Powell optimizer. Based on [27], [18] develops a CUDA-based optimization strategy for deformable registration fashion that aims at optimizing joint histogram – and then Normalized MI – and gradient computation, though exploiting some pre-computation mechanisms and dataset-specific techniques that reduce the computational requirement.

As discussed so far, hardware-based solutions are desirable in image registration, given the high-intensity workload. However, FPGA-based solutions are generally closed-source and not customizable by the final user, while GPUs are not customizable architectures at all, and are known to be power hungry devices. Based on these considerations, with this work, we propose an open-source hardware-software framework for image registration that exploits a customizable FPGA-based accelerator for the computation of MI, easily reusable in several image registration algorithms or even different fields of applications, such as phylogenetic [21] and features selection [15]. Moreover, being based on the PYNQ framework, our APIs are easily employable through Python, resulting transparent to the end-users. Finally, the overall hardware-software system is deployable on different FPGAs from embedded to high-end.

4 PROPOSED DESIGN METHODOLOGY

The image registration procedure we consider in this work is an intensity-based multi-modal algorithm, and its three main building blocks are affine transformation, Powell's and (1+1) Evolutionary optimization methods, and mutual information similarity metric. Since the similarity metric is the most compute-intensive part [14, 29] and multiple fields of application benefit from a MI accelerator (e.g., genomics computations, telecommunication field), we present a framework to produce an architecture able to perform the MI computation. Fig. 2 shows the steps of the whole workflow to register two images and which part is offloaded to the hardware accelerator (Evaluate MI).

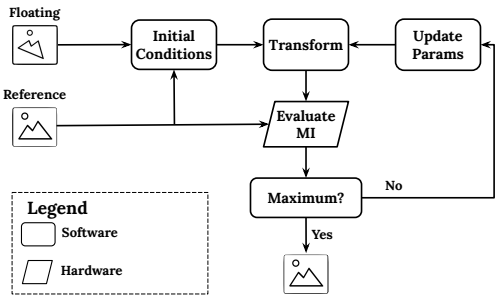

Figure 2: High-level view of the workflow and its components

4.1 Framework Overview

This work proposes a framework to assist users in the generation of different versions of a hardware accelerator for MI calculation according to their needs. It is important to note that, even though this work focuses on image registration, the MI calculation is an essential part of many fields of application. Therefore, other contexts, where users may have different requirements, could benefit from MI acceleration. For instance, the final user might choose to sacrifice performance to save resources or to target a high-end scenario where performance is the main goal. For this reason, we devised our framework as an open-source solution capable of guaranteeing a high level of flexibility in the generation of the MI accelerator.

Our framework provides different *customization parameters* to assist the user in the exploration of the design space and to tune the multiple features of the MI accelerator. In particular, such parameters have a direct impact on the performance the resulting accelerator can achieve, as well as its resource consumption. Section 4.3 accurately describes the customization parameters. After selecting the parameters, the framework applies the requested customizations to the base structure of our FPGA-based accelerator (more details in Section 4.2). The result is a tailored design devised to perform MI calculation suitable for High-Level Synthesis (HLS) tools. Besides, the framework generates specific scripts to automate both the HLS process and the synthesis flow. In particular, given an either embedded or high-end target device, the framework determines all the required steps towards the bitstream generation. Finally, once the synthesis process is over, the user can leverage on the transparent Python APIs the framework supplies, and easily integrate the accelerator usage within applications based on the Xilinx PYNQ framework [35]. Indeed, we provide Python APIs able to handle all the accelerator configurations independently, e.g., caching or not caching (see Section 4.3), hiding the differences of the embedded or high-end device, and in some cases to measure the power directly on-board. A new image registration procedure, or an

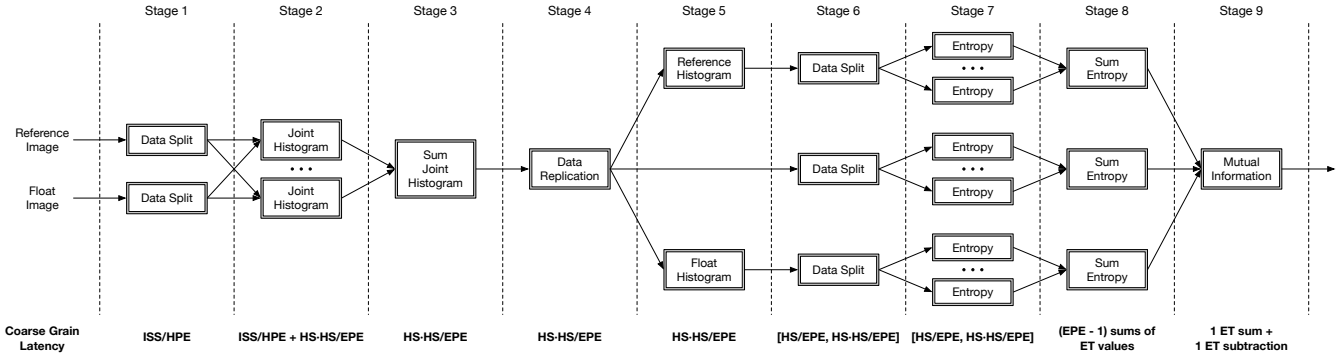

Figure 3: High level view of the architecture for MI computation and its latency (latency parameters defined in Tab. 1)

algorithm that employs MI computation, can exploit our accelerator with four simple additional steps: prepare the buffers with the data, start the MI computation, collect the results, and free the buffers.

4.2 Accelerator Architecture

Our accelerator adopts a dataflow computational model. Indeed, we built a multi-stage pipeline to perform the computation efficiently. Fig. 3 depicts the proposed pipeline and reports the coarse grain latency of each Stage, which we will analyze in Section 4.4. In particular, we identified two macro Stages within the pipeline. The first macro Stage (Stage 1 to 3) mainly consists of the joint histogram computation, while the second one (Stage 4 to 9) regards the entropy computations for MI calculation. Finally, each Stage is internally pipelined and streams the data to the following one through FIFOs.

Input fetching. At the very beginning of the execution, the architecture fetches the two input images (reference and floating). This step depends on the device physical memory ports and the available bandwidth. Since not all the FPGA-based devices offer multiple memory ports, to be as generic as possible, we consider a case with a single memory port that is multiplexed (in case of multiple ports available, accelerator can be replicated according to the number of ports). However, this scenario may harm the accelerator performance, especially in the case of memory-bound designs, as the current one. Hence, it is paramount to properly design the accelerator according to the bitwidth of the memory ports and the memory bandwidth. A solution to alleviate such a problem could be to prefetch one or both images on the local memories. This is particularly suitable for algorithms, like image registration. Indeed, to register two images, the reference image does not change throughout the whole optimization procedure, while the floating one continuously changes. Therefore, if the target FPGA has enough on-chip memory and we apply this feature, our architecture first prefetches the reference image and then reads the floating one in a streaming fashion; otherwise, it reads both images simultaneously.

Stage 1-3. Assuming an input image bitwidth (IBW) of 8-bit, and a 32-bit memory port bitwidth (MBW) on the target device, we can pack more data per single memory transfer, i.e., 4 pixels (8-bit wide each) (MBW/IBW). Thus, this Stage takes the data coming from either the off-chip memory or the on-chip one and splits them to

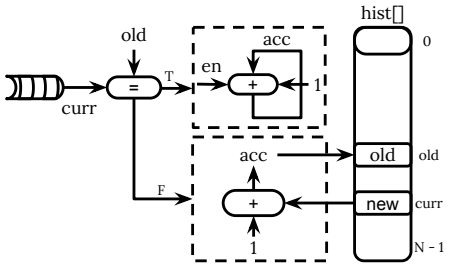

Figure 4: A generic Processing Element optimized for histogram computation. In the case of a joint histogram, *curr* is the current index computed for a flattened 2D array.

multiple Processing Elements (PEs) performing the joint histogram. HPE defines the number of parallel PEs.

Stage 2 and Stage 3 compute the joint histogram of the two images with a map-reduce approach [19]. During Stage 2, the PEs receive two streams of unpacked data, from both the reference and floating images. As shown in Fig. 4, each PE creates its local histogram (in this case, a joint one) and stores the histogram values in BRAM. The input streams generate the joint indices of the joint histogram positions in which an increment by one occurs. As stated in Section 2, the joint histogram counts how many times a couple of intensities i, j appears, with i, j belonging to the reference, and the floating images, respectively. Practically, the images flow through Stage 2, which increments the intensity at position i, j. To avoid RAW hazard in the computation, each PE is optimized to accumulate the current intensity on a register while i, j are the same, on the contrary, if the current i, j are different from the previous ones, the PE reads the new value to accumulate from BRAM and writes back the previous one. As soon as the joint histogram is ready, this Stage subsequently sends it out to the following one. The output stream contains multiple histogram values packed together. EPE indicates the number of packed values. Stage 3 reduces the parallel computed joint histograms by summing the values within the HPE input streams. The adopted map-reduce approach improves parallelism of the joint histogram computation, therefore the latency, at the cost of storing HPE different joint histograms.

Table 1: Customization Parameters of our Architecture

Parameter	Description
CACHE	support caching of one input image
IBW	input bitwidth of the single data, e.g., pixel
MBW	input port bitwidth, or memory port bitwidth
ISS	maximum input stream size value
HS	single histogram size
HPE	number of parallel histogram PEs, depending on *MBW* value
EPE	number of parallel entropy PEs
ET	data type precision of entropy computation
NCORE	number of parallel cores

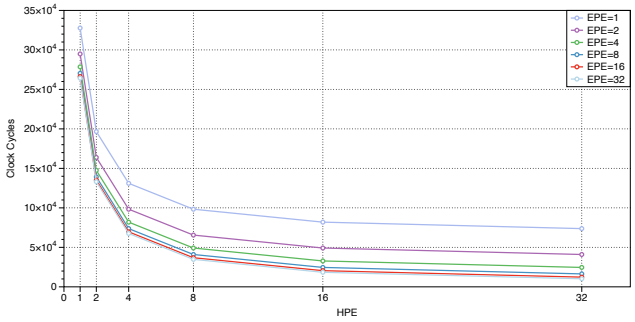

Figure 5: Estimation of the proposed architecture latency according to the *HPE* and *EPE* parameters.

Stage 4-9. As stated in Section 2, once the joint histogram is computed, it is possible to derive the separate histograms of the input images and their relative probabilities. This is crucial for the entropy computations, as the Shannon's formula of the entropy revolves around the sum of the probability times the logarithm of the probabilities (see eq. (2)). For this reason, Stage 4 replicates the joint histogram stream three times, so that Stage 5 extracts the histograms of the reference and floating images by reducing per rows or columns, respectively.

Stage 6 unpacks each input stream to *EPE* ones. Stage 7 is in charge of computing the entropy, starting from the three histograms. This Stage computes Shannon's entropy for the single input (eq. (2)), and it is the only one that requires `floating-point` values. To limit the number of floating-point operations, we defer the scaling of each input (to retrieve the probability) to Stage 9. Thus, we only need `floating-point` values for log operations. In particular, this Stage relies on either IEEE 32-bit `floating-point` or custom bitwidth `fixed-point` values as the required data type. Stage 8 accumulates the partial entropy and computes the final ones. Finally, Stage 9 receives the three entropy values, computes the MI value for the two input images, and writes it back to the host.

4.3 Assisted Exploration of Design Configuration Parameters

Our whole design is highly customizable according to the target scenario, different application requirements, and the target platform. *CACHE* parameter indicates whether the architecture can cache one image (like the reference) into BRAMs (or URAMs when available). Considering the input memory port bitwidth (*MBW*) and the input data type bitwidth (*IBW*), the joint histogram part is parallelizable through $HPE = MBW/IBW$ PEs, which impacts on the architecture memory footprint. Indeed, each PE stores a joint histogram whose size is $HS \cdot HS$, where $HS = 2^{IBW}$ is the size of a single histogram. Besides, a user can deploy the architecture configuration that fetches a maximum input stream size (*ISS*) per iterations of different sizes, e.g., a 512×512 reference and 512×512 floating. Likewise, the datapath is customizable to different input data types, as 8-bit pixel or 16-bit pixel. It is worth noticing that scaling to larger images, e.g., $ISS = 2048 \times 2048$, influences resource usage and image transfer times from the main memory. In particular, *ISS* slightly impacts the bitwidth of joint histogram elements, while it significantly affects BRAM/URAM usage of caching designs.

Different parameters affect the design of the second macro Stage. *EPE* describes the number of histogram values coming from Stage 2 packed together and, consequently, the number of parallel entropy modules per histogram. Another parameter is the data type (*ET*) used for the entropy computation. The data type can be either 32-bit IEEE `floating-point` or custom bitwidth `fixed-point`, which may introduce errors in the entropy due to the precision loss. In particular, the bitwidth of `fixed-point` values depends on *ISS*.

Finally, it is possible to deploy a multi-core version of the architecture to increase run-time performance. More specifically, we can instantiate *NCORE* accelerators and connect each one to a different memory port (one per port to avoid contention). This permits us to perform multiple image registrations in parallel.

Tab. 1 summarizes all the customization parameters, along with their description, as proposed in this Section.

4.4 Architecture Latency

The parameters in Tab. 1 enable to tune the proposed architecture according to the user's requirements. Starting from these parameters, we can analyze the latency of a given instance of our architecture and evaluate its theoretical performance.

Fig. 3 reports the latency breakdown of each Stage of our design. The purpose of the formulae in Fig. 3 is to model latency at a steady-state and consequently provide a coarse grain estimation of the clock cycles. Thus, we do not take into account either the clock cycles required to fill up the internal pipeline of each stage or the off-chip memory bandwidth. On the other hand, as stated before, the memory port bit-width (*MBW*) is a relevant parameter of our design. Indeed, the latency of Stage 1 and 2 depends on *MBW*, since it determines *HPE*. In particular, given *HPE* and *ISS*, each block of Stage 1 takes ISS/HPE clock cycles to read the input from the off-chip memory and split it in *HPE* streams of data. Similarly, each block of Stage 2 receives a stream of data and computes a partial joint histogram in ISS/HPE clock cycles. Then, these blocks write the partial joint histogram to the output FIFO in $HS \cdot HS/EPE$ clock cycles. Since these two computations within Stage 2 cannot overlap, the latency of each block of Stage 2 is the one reported in Fig. 3.

Stage 3 reads the incoming *HPE* streams, sums the partial joint histograms, and writes the complete joint histogram to the output FIFO in $HS \cdot HS/EPE$ clock cycles. The following Stage 4 takes the same amount of clock cycles to read the joint histogram and replicate it three times. The computations performed in the two blocks of Stage 5 differ, but they share the same coarse grain latency.

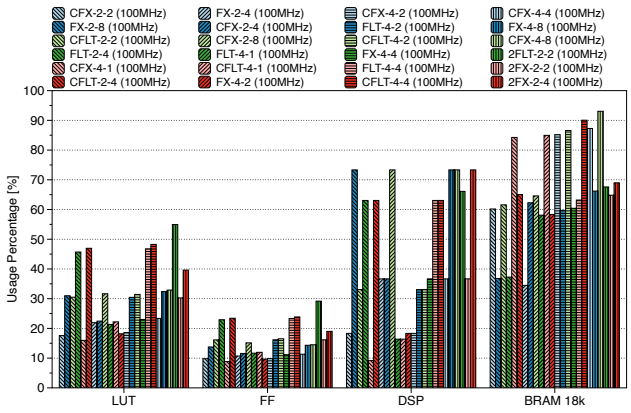

Figure 6: Ultra96 resource utilization. Resources: 70560 LUT, 141120 FF, 432 BRAM 18k, 360 DSP

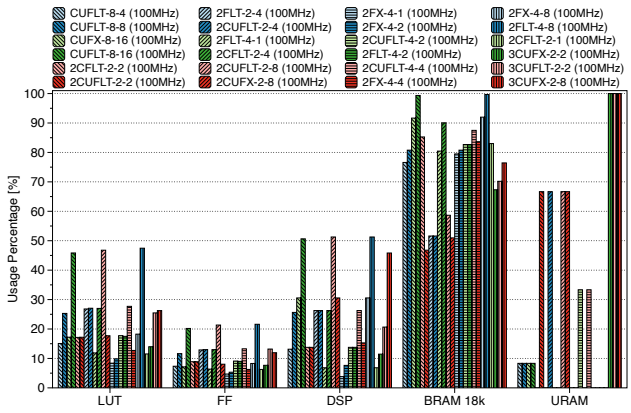

Figure 7: ZCU104 resource utilization. Resources: 230400 LUT, 460800 FF, 624 BRAM 18k, 1728 DSP, 96 URAM

Starting from Stage 6, the latency of each block varies, even though they are identical internally, as reported in Fig. 3. Indeed, the blocks directly connected to Stage 5 read a smaller amount of data than the one directly connected to Stage 4. In particular, the blocks of Stage 6 directly connected to Stage 5 receive a single histogram, while the one directly connected to Stage 4 the joint one. Consequently, the same holds for the blocks in Stage 7.

The output of each block of Stage 7 is a single *ET* value. Thus, the latency of the blocks of Stage 8 is almost negligible, as they only sum the incoming values. Likewise, Stage 9 reads the three input entropies and outputs the mutual information in few clock cycles.

The proposed architecture works in a dataflow fashion, and each stage is internally pipelined. Hence, the coarse grain latency of a given instance of our architecture is $ISS/HPE + HS \cdot HS/EPE$. This value mainly depends on both the joint histogram calculation and its entropy computation. Specifically, considering a specific *ISS* and *HS*, both *HPE* and *EPE* provide a theoretical performance boost that scales as $1/x$. Fig. 5 shows how the coarse grain latency scales with $ISS = 512 \times 512$ and $HS = 256$. We extracted these values via the cycle-accuracy cosimulation of Vivado HLS. Fig. 5 does not take into account the memory bandwidth, for it does not illustrate the effects of caching. The values of *HPE* and *EPE* are a power of 2 for the sake of simplicity. In this case, *HPE* has a greater impact on the estimated latency than *EPE*, as $ISS > HS \cdot HS$.

5 EXPERIMENTAL RESULTS

This Section details the environment we used to validate our approach (Section 5.1), the results of our Design Space Exploration, and the performance evaluation (Section 5.2).

5.1 Experimental Setup

The proposed framework generates a C++ accelerator architecture suitable for HLS tools. In this work, we relied on the Xilinx Vivado HLS 2019.2 toolchain to produce the RTL of our accelerator. We target four boards from different scenarios: A Pynq-Z2 board based on the Xilinx Zynq SoC, an Ultra96 v2 board powered by a Xilinx MPSoC Ultrascale+ ZUEG3, a Zynq UltraScale+ MPSoC ZCU104, and an accelerator card, namely an Alveo u200 board with a Xilinx Ultrascale+ XCU200. These boards range from low-power embedded devices (Pynq-Z2) to high-end accelerator cards (Alveo u200). Through the scripts generated by the framework, we employ the Xilinx Vitis and Vivado HLx toolchains to generate the bitstream, both version 2019.2. The image registration application is implemented in Python in a multi-threaded version, and the hardware-software interfacing part is handled through our Python APIs, which interact with the PYNQ [35] framework v2.5 for both the embedded and the high-end devices. The host processors are a dual-core ARM A9 (Pynq-Z2), a quad-core ARM A53 (Ultra96 and ZCU104), and a quad-core Intel i7-4770 (Alveo u200). While the first three reconfigurable fabrics are on the same chip with the host and share the DDR, the Alveo is connected through PCIe to the host device. We evaluated the proposed solution with a stack of 247 Computed Tomography (CT) images, with a dimension of 512×512 pixels, and a corresponding number of Positron Emission Tomography (PET) ones, resized from 128×128 pixels to a dimension of 512×512 pixels, from the medical field, each down-scaled to 8-bit wide (Patient: C3N-00704, Study: Dec 10, 2000 NM PET 18 FDG SKULL T, CT: WB STND, PET: WB 3D AC) [10, 25].

Each image has various misalignments depending on the patientsâĂŹ movements or acquisition protocols. We compare our solutions against both works available in the literature and an optimized state-of-the-art MATLAB implementation [24], i.e., the one available in the MATLAB Image Processing Toolbox, which exploits as many cores as are available, and can be further optimized thanks to the Parallel Computing Toolbox. The tool version is 2019b and runs on both a dual-core Intel i5-7267U CPU and a 40 core Intel Xeon Gold 6148 CPU. Our solution and the MATLAB implementation share the transformations, the similarity metric, and the input dataset. With respect to the optimizer, we exploit both Powell and (1+1) Evolutionary, while MATLAB the (1+1) [31]. Powell optimizes one parameter at a time in a given range using MI, while the (1+1) evolves genetically from a parent transformation vector to a child using MI and a normal random generator.

5.2 Experimental Evaluation

We first evaluate the target hardware-software framework through a design parameter exploration across the several customization parameters of the proposed architecture on the Ultra96, ZCU104, and Alveo board. Then, we compare the execution times of our hardware designs to MATLAB, with and without the Parallel Computing Toolbox. Finally, we analyze the best performance of the four boards against state-of-the-art works. Given the target case study, the framework allows us to quickly produce several designs adopting $IBW = 8$ and $ISS = 512$ for all the considered devices. We consider *MBW* as the product of *IBW* and *HPE*, thus it is not reported in any of the following analyses. We describe a configuration as the concatenation of the following parameters: *NCORE* (if missing, we assume 1), *CACHE* (if missing, no caching, otherwise *C* for BRAM caching, *CU* for URAM caching), *ET* (*FLT* for 32-bit `floating-point`, *FX* for `fixed-point`), *HPE*, *EPE*. For instance, FLT-2-1 describes a design with 1 core, no caching, 32-bit `floating-point`, 2 *HPE*, and 1 *EPE*.

5.2.1 ***Resources Design Space Exploration.*** Fig. 6, 7, and 8 report the most relevant synthesis results for Ultra96, ZCU104, and Alveo we achieved throughout several runs. While we can see that the amount of LUTs and FFs used generally remains reasonably low, BRAMs and DSPs usage varies a lot based on the configuration.

From the several single core versions, we can appreciate how BRAMs usage mainly comes from *HPE* scaling or the reference image caching, making BRAMs the critical resources of the proposed accelerator. Indeed, the devised PE for the joint histogram requires a local, though smaller, partial joint histogram memory. Therefore, increasing the level of parallelism dramatically boosts the performance, as we will discuss in the following Section, at the cost of higher impact on memory footprint. On the other hand, caching requires an on-chip memory able to fit $512 \times 512 \times 8$ bits. Besides, designs on Ultra96 using caching require a significant amount of BRAMs, while ZCU104 and Alveo board not only have more BRAMs but also URAMs in the reconfigurable fabric. Thus, all the configurations exploiting URAMs (the ones with the *U*), drastically reduce the BRAM usage and pave the way to configurations that otherwise would not fit the FPGA. On the other hand, *EPE* mainly impacts on the amount of required DSPs (particularly when using 32-bit `floating-point` values) due to the usage of logarithms (we rely on the implementation available within Vivado HLS) and `floating-point` multiplications. In the case of `fixed-point`, the integer and decimal parts are a function of *ISS*, and here are 23 and 19 bits. We analyzed the impact of `fixed-point` precision on the MI calculation and measured an MSE of 3.46E-10 compared to `floating-point` (100 tests with random inputs).

Considering the multi-core version, we should notice how the Ultra96 scales to few cores, while ZCU104 and Alveo fabrics can scale to multiple cores with different combinations of *HPE* and *EPE*. Upscaling the core design number is limited by BRAM, DSP, and the number and wideness of physical DRAM ports. For this reason, increasing the number of cores on Ultra96 or ZCU104, which have a single DDR with a 32-bit port, creates contention and leads to performance degradation, as there are no more logical resources. On the other hand, the Alveo u200 has 4 DDR memory banks, each of which has a capacity of 16GB and 64-bit ports. Thus, on the Alveo,

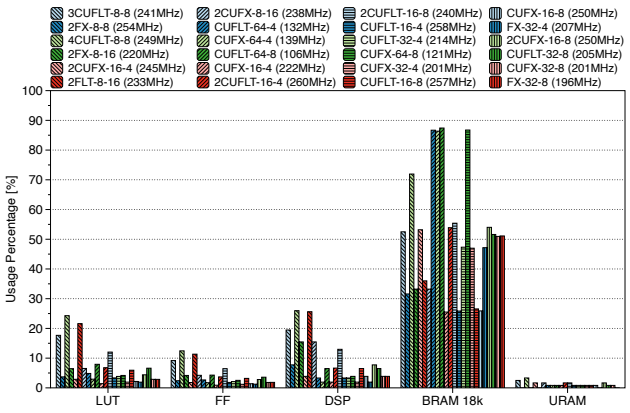

Figure 8: Alveo u200 resource utilization. Resources: 1019968 LUT, 2128354 FF, 3532 BRAM 18k, 960 URAM, 6833 DSP

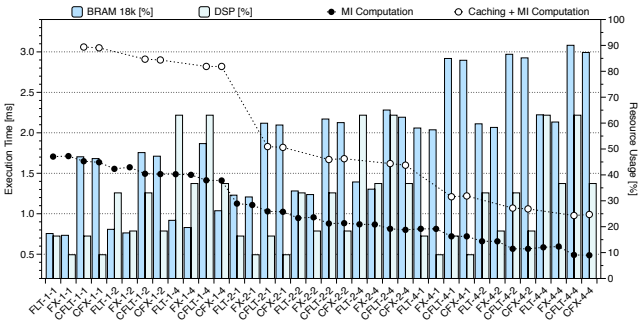

Figure 9: Average execution time and resource usage scaling according to the main parameters that affect the architecture, i.e., latency *HPE, EPE, ET*, and *CACHE*. Standard deviation not reported as negligible (from $2.61E-03$ to $3.12E-02$).

we exploit one core per memory bank whenever it is possible, and, as such, the *HPE* value multiplied by 8 gives the resulting bit-width the cores would require. It is worth noticing that, in this case, the data transfer via PCIe between the host and the Alveo board may become a bottleneck when using multi-core designs.

Finally, while, for the Alveo designs, Vitis automatically scales all the design at the maximum frequency it can handle, and thus does not require manual intervention, Vivado does not. Hence, we synthesize Pynq-Z2, Ultra96, and ZCU104 designs at 100MHz, but then we exploit the PLL of the Processing System to hand-tune at run time the frequency and check the consistency of our results. As a result, Alveo designs run at the frequency reported in Fig. 8, while all the others at 200MHz.

5.2.2 ***Performance Analysis.*** We evaluate the architecture performance for an image registration application on the execution time of the single value of MI, the single image registration, and the overall stack of images.

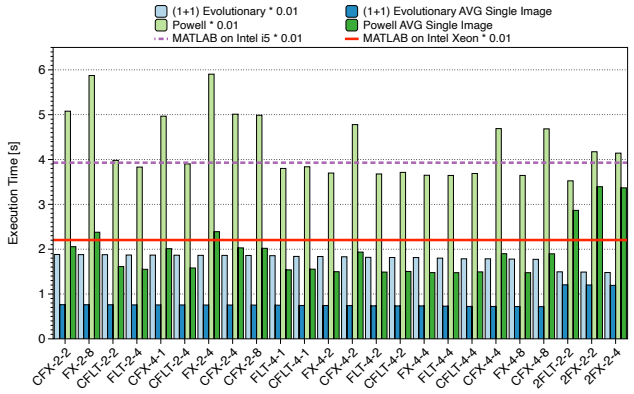

Figure 10: Ultra96 execution times of the whole application using both Powell and (1+1) Evolutionary, and their average time per image, compared against MATLAB on both an Intel i5 and an Intel Xeon. We scaled the execution times of MATLAB and our image registration applications by a 0.01 factor.

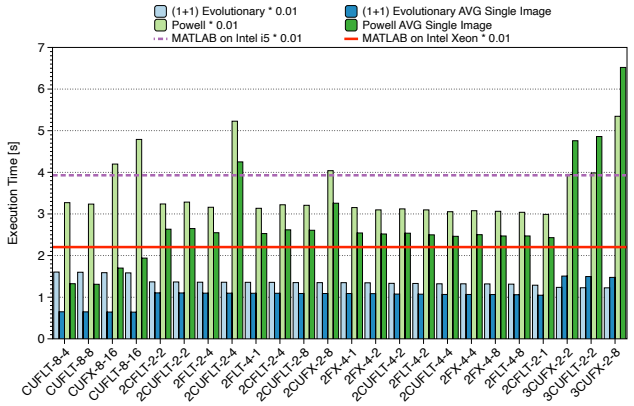

Figure 11: ZCU104 execution times of the whole application using both Powell and (1+1) Evolutionary, and their average time per image, compared against MATLAB on both an Intel i5 and an Intel Xeon. We scaled the execution times of MATLAB and our image registration applications by a 0.01 factor.

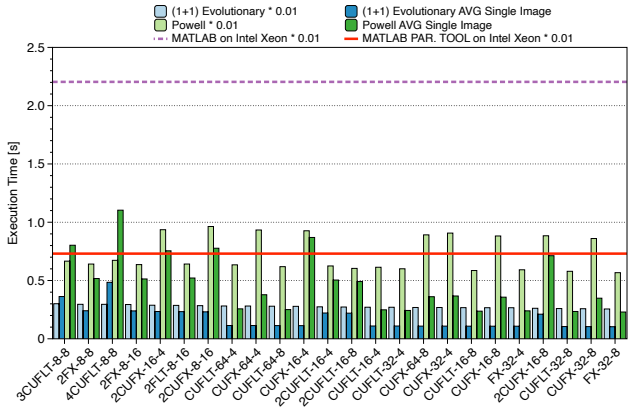

Figure 12: Alveo u200 execution times of the whole application using both Powell and (1+1) Evolutionary, and their average time per image, compared against MATLAB on an Intel Xeon. We scaled the execution times of MATLAB and our image registration applications by a 0.01 factor.

Fig. 9 shows how the single MI computations vary according to *ET*, *HPE*, *EPE*, and *CACHE* while keeping a single core (multi-core would not impact the single MI computation), on the Ultra96 running at 200MHz. We can notice how, by increasing either *HPE* or *EPE* or enabling *CACHE*, we reduce the average execution time for a single computation. In particular, we can notice how *HPE* and *EPE* parameters impact the performance scaling in line with the latency analysis of Section 4.4. Conversely, when *CACHE* is available, we have to account for both the caching time and MI computation time. Fig. 9 reports both the single MI execution time and the aggregate one. Although this solution may be inefficient for a single MI computation, it helps to decrease the overall time when one of the two images does not change for several iterations, e.g., during the image registration process. On the other hand, Fig. 9 shows the direct connection between *HPE* and BRAM usage and between *EPE* and DSP usage. It is interesting to highlight that it is possible to distribute the level of parallelism between *HPE* and *EPE* and partially reduce performance while balancing resource usage (configurations FLT-4-1 and FLT-2-2). Finally, Fig. 9 shows that the *FX* data type significantly reduces DSP usage, as well as slightly improving both the average computation time and BRAM usage. Indeed, it would be possible to instantiate configurations with $EPE = 8$ or $EPE = 16$.

We also measured the average execution time per image, and the overall registration time for the entire stack of images. Fig. 10, 11, and 12 show how these times scale for the most significant configurations on Ultra96, ZCU104, and Alveo, respectively. We sorted the configurations in decreasing order according to the overall registration times of the (1+1) Evolutionary. In general, the most impacting factor in the execution time comes from the *HPE* value, which impacts the bandwidth and the time required to process and compute the joint histogram (one of the biggest bottlenecks). Another relevant factor is caching. Indeed, most of the configurations exploit caching to reduce the execution time required by the application. The entropy data type *ET* affects the execution time of the registration process as well. Finally, we can notice how the *NCORE* parameter has a limited effect on the execution times, due to both the unique physical port (on embedded boards) and the run-time overhead of Python when managing multithread applications. Thus, a careful balance of these parameters enables to optimize the application run-time. Considering (1+1) Evolutionary, Ultra96 and ZCU104 easily outperform the MATLAB reference on the Intel i5, reaching speedups up to 2.66× and 3.21×, respectively. Similarly, both boards result faster than the MATLAB on the Intel Xeon without the Parallel Computing Toolbox enabled, and the best speedups are 1.49× and 1.80×, respectively. On the other hand, the selected designs running on the Alveo u200 outtake the Intel Xeon by a factor up to 2.86× (Parallel Computing Toolbox enabled) and 8.62× (Toolbox not enabled), and 15.36× against the Intel i5 (here not

Table 2: Comparison with Related Works with Perf. measured as [*ms/MVoxels/iterations*], as proposed by [28], and Power Eff. as [*(MVoxels Âů iterations)/(ms Âů kWatt)*]

Arch.	Work	Transform	Metric	Optimizer	Hardware	Perf. (lower is better)	Power Eff. (higher is better)
FPGA	**FX-4-4**	Affine	MI †	Powell	PYNQ-Z2 (28nm)	49.90	8.02
	2FLT-2-2	Affine	MI †	Powell	Ultra96 (16nm)	11.02	12.52
	2CFLT-2-1	Affine	MI †	Powell	ZCU104 (16nm)	9.34	8.56
	FX-32-8	Affine	MI†	Powell	Alveo u200 (16nm)	1.78	18.52
	FX-4-4	Affine	MI †	1+1	PYNQ-Z2 (28nm)	0.42	979.76
	2FX-2-4	Affine	MI †	1+1	Ultra96 (16nm)	0.09	1534.00
	3CUFX-2-8	Affine	MI †	1+1	ZCU104 (16nm)	0.08	981.60
	FX-32-8	Affine	MI†	1+1	Alveo u200 (16nm)	**0.02**	**2058.98**
	[8]	Rigid	MI†	N/A	2xAltera 1K100 (n.a.)	101★	N/A
	[14]	MultiRigid	MI†	Simplex	Altera EP2S180 (90nm)	13.4★	N/A
	[29]	Affine†	Corr.†	Simplex	Zybo (28nm)	9.15⊙	N/A
GPU	[9]	N/A	MI†	N/A	FX 5800 (55nm)	39.04/1.07▽•	0.13/4.94‡
	[27]	Rigid	MI†	Powell	GTX 280 (65nm)	4.06★	1.04‡
	[18]	Nonrigid	NMI†	Grad. Desc.†	GTX 580 (40nm)	0.13▽⋄	31.52‡
CPU	Matlab	Affine	MI	1+1	Intel i5-7267U (14nm)	0.24⊗	176.97‡
	Matlab	Affine	MI	1+1	Intel Xeon Gold 6148 (14nm)	0.14⊗	48.37‡
	Matlab Par. Tool	Affine	MI	1+1	Intel Xeon Gold 6148 (14nm)	0.05⊗	145.93‡
	Simple ITK	Rigid	MI	Grad. Desc.	Intel Xeon Gold 6148 (14nm)	0.25⊗	27.01‡
	Simple ITK	Rigid	MI	Powell	Intel Xeon Gold 6148 (14nm)	2.51⊗	2.65‡
	Simple ITK	Rigid	MI	1+1	Intel Xeon Gold 6148 (14nm)	0.89⊗	7.46‡

†Implemented in hardware ★Computed from [28] ‡Computed with Termal Design Power (TDP) as power ⊙ Assuming maximum iteration of 500
▽ Exploits the binning to reduce joint histogram sizes • The first number includes presorting time ⊗ With maximum 100 iterations
⋄ This value is the result of several dataset-specific approximations and preprocessing that reduce the computation to 1/6 and lead to misregistrations [18]

reported in the chart to ease the visualization). Moving to Powell's optimization method, we notice that this procedure is more sensitive to the entropy data type. Indeed, the configurations employing fixed-point data type take more iterations to converge Nonetheless, various designs of the Ultra96 and ZCU104 surpass MATLAB on the Intel i5. The same holds for multiple Alveo configurations against MATLAB on the Intel Xeon. However, it is crucial to note that we are considering two different optimization methods; indeed, Powell optimizes a parameter at a time within a given range per iteration, while (1+1) generates a new child vector, comprehensive of all parameters, per iteration, hence the evolutionary algorithm requires fewer computations.

5.2.3 ***Accuracy Analysis***. We compare the accuracy of our solutions against the Matlab procedure with the Dice score metric, which evaluates how good is the overlap between two region of interests. In Fig. 13 we reported an example of visual comparison of our registrations and the Matlab one along with the gold standard. We extracted the gold standard with a supervised semi-automatic procedure, based on the interactive Matlab Registration Application exploiting the multi-modal registration model. To evaluate the Dice score, and hence the accuracy, we binarize both the gold standard and output images. As a consequence, if the border of registered structures are correctly overlapped to the gold standard, also the internal structures will be correctly registered, as the employed geometric transformation does not insert deformations. Based on this analysis, our top-performing Alveo implementation reached a mean Dice score of 94% and 78% with Powell and (1+1) respectively in line with both the optimized Matlab implementation of 85% and the manually extracted gold standard.

5.2.4 ***State-of-the-Art Comparison***. In the image registration field, comparing against state of the art is extremely hard given the absence of a standard dataset, the availability of the source codes, the broad combinations of the image registration methodologies, and the different hardware platforms. For these reasons, we open-source our solution at this link [11]. FPGA-based approaches in the literature [8, 14, 29] all compare against their single-thread software implementation only. Besides, they mainly perform mono-modal image registration on 256×256 images, using Simplex as the optimizer. While [8, 14] use MI as similarity metric and deformable transformations, [29] uses correlation and affine ones. Conversely, we use a 512x512 multi-modal image registration with Powell as the optimizer, MI as the similarity metric, affine transformations, and we compare against a multi-threaded Matlab optimized software. [28] shows the struggle to provide a standard performance metric to compare different works, and proposes $ms/MVoxels/iterations$ as a solution (the lower, the better)., where these parameters refer to the overall registration process of N images. Tab. 2 reports the metric proposed from [28], where we took numbers for [8, 14, 27],

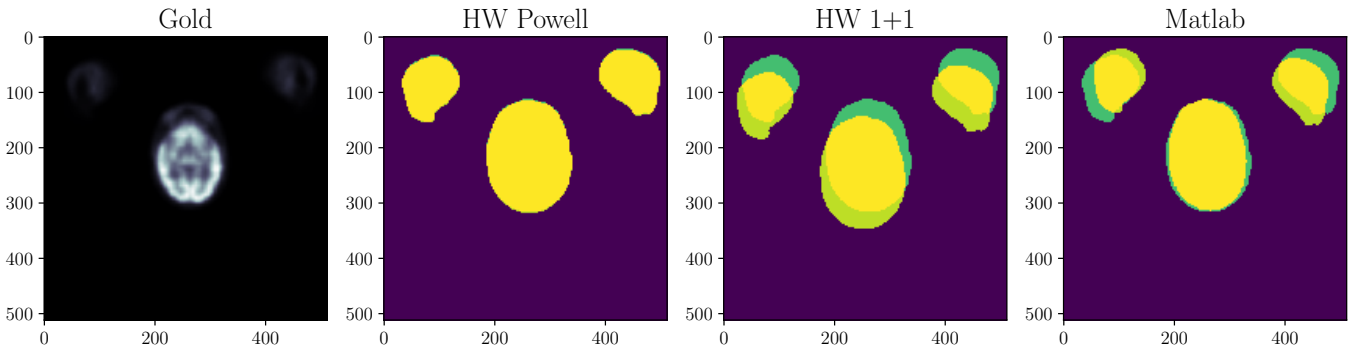

Figure 13: A visual example of image registration results. From left to right, the gold standard, the overlap between gold standard (represented in green) and the registered image wih Powell, (1+1), and MATLAB (all in yellow)

and we compute the same metric for this work and other works reported in Section 3. For many of those works, we have computed the performance according to the information available in [28] and to be as much fair as possible. We exploit the performance metric to compute the power efficiency, last column of Tab. 2, as $1/(Perf. \times PowerConsumption)$. Considering Alveo and GPUs solutions, we account for the board power only, while the others also account for the CPU power, especially our embedded boards.

Regarding the FPGA-based solution, this work achieves better performance against all the selected FPGA approaches, reporting 0.02 and 1.78 $ms/MVoxels/iterations$, with FX-32-8 as configuration for the Alveo u200, respectively for (1+1) and Powell.

By looking at the GPU-based approaches reported in Tab. 2, most of them rely on both precomputation techniques and binning strategies to reduce the computational load of the algorithm. While the former is mainly algorithmic and dataset dependent, exploiting binning levels makes the joint histogram computation, and the following ones, less time consuming, by sacrificing the accuracy. In this context, [18] reports encouraging numbers, though it seems they apply many precomputation and dataset-specific techniques without which the computation would be way more expensive. The authors also report various cases where the registration process fails to align the images. Indeed, our best performance result with the (1+1) on all the boards but PYNQ-Z2 are better than [18], without computation reduction techniques. Moreover, we outtake all the considered GPU implementation in terms of power efficiency, with maximum of 65× (Alveo (1+1) versus [18]).

Finally, we have compared our implementations against two software applications that we deployed exploiting two well-known image processing tools, namely MATLAB Image Processing Toolbox and SimpleITK. As we can see from Tab. 2, MATLAB achieves better results compared to SimpleITK, and in particular it reaches results in line with our (1+1) top implementations, when the Parallel Computing Toolbox is enabled on the Xeon Gold. However, considering the power efficiency, all the MATLAB implementations prove to be less efficient than our solutions with similar performance.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we presented an open-source hardware-software framework to automate the design and synthesis of a configurable architecture for mutual information calculus oriented to the possible constraints for the end use case. We evaluated our framework in both the embedded and high-end scenarios through the given transparent APIs, showing how our designs scale according to the parameters and how it provides remarkable performance. Indeed, we compared our accelerators against an optimized MATLAB version on an Intel Xeon Gold reaching a speedup of up to 2.86× for the registration of 247 images and a Dice score of 94% and 78% with Powell and (1+1) respectively. We compared our performance with state-of-the-art approaches employing different techniques, overwhelming FPGA-based solutions, with the metric proposed in [28], and achieving remarkable power efficiency results.

Our architecture can be easily customized and deployed in other application fields such as genomics computations, financial computations for high-frequency trading, and telecommunication field. As future work, we will validate the architecture onto these other use cases and explore different customization parameters, such as the binning scheme. Finally, we plan to provide our APIs also in C++ to enable a more performing software infrastructure.

ACKNOWLEDGMENT

Data used in this publication were generated by the National Cancer Institute Clinical Proteomic Tumor Analysis Consortium (CPTAC).

REFERENCES

[1] Zeynettin Akkus, Alfiia Galimzianova, Assaf Hoogi, Daniel L Rubin, and Bradley J Erickson. 2017. Deep learning for brain MRI segmentation: state of the art and future directions. *Journal of Digital Imaging* 30, 4 (2017), 449–459.

[2] MA Aswathy and M Jagannath. 2017. Detection of breast cancer on digital histopathology images: Present status and future possibilities. *Informatics in Medicine Unlocked* 8 (2017), 74–79.

[3] Lalit Bahl, Peter Brown, Peter De Souza, and Robert Mercer. 1986. Maximum mutual information estimation of hidden Markov model parameters for speech recognition. In *ICASSP'86. IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 11. IEEE, 49–52.

[4] Anton Bardera, Miquel Feixas, Imma Boada, Jaume Rigau, and Mateu Sbert. 2006. Medical image registration based on BSP and quad-tree partitioning. In *International Workshop on Biomedical Image Registration*. Springer, 1–8.

[5] Youcef Bentoutou, Nasreddine Taleb, Kidiyo Kpalma, and Joseph Ronsin. 2005. An automatic image registration for applications in remote sensing. *IEEE Transactions on Geoscience and Remote Sensing* 43, 9 (2005), 2127–2137.

[6] Lisa Gottesfeld Brown. 1992. A survey of image registration techniques. *ACM Computing Surveys (CSUR)* 24, 4 (1992), 325–376.

[7] Atul J Butte and Isaac S Kohane. 1999. Mutual information relevance networks: functional genomic clustering using pairwise entropy measurements. In *Biocomputing 2000*. World Scientific, 418–429.

[8] Carlos R Castro-Pareja, Jogikal M Jagadeesh, and Raj Shekhar. 2003. FAIR: a hardware architecture for real-time 3-D image registration. *IEEE Transactions on Information Technology in Biomedicine* 7, 4 (2003), 426–434.

[9] Shifu Chen, Jing Qin, Yongming Xie, Wai-Man Pang, and Pheng-Ann Heng. 2009. CUDA-based acceleration and algorithm refinement for volume image registration. In *2009 International Conference on Future BioMedical Information Engineering (FBIE)*. IEEE, 544–547.

[10] Kenneth Clark, Bruce Vendt, Kirk Smith, John Freymann, Justin Kirby, Paul Koppel, Stephen Moore, Stanley Phillips, David Maffitt, Michael Pringle, et al. 2013. The Cancer Imaging Archive (TCIA): maintaining and operating a public information repository. *Journal of digital imaging* 26, 6 (2013), 1045–1057.

[11] Davide Conficconi, Eleonora D'Arnese, Emanuele Del Sozzo, Donatella Sciuto, and Marco D. Santambrogio. 2020. A Framework for Customizable FPGA-based Image Registration Accelerators. https://github.com/necst/iron. https://doi.org/10.5281/zenodo.4432762

[12] Thomas M Cover and Joy A Thomas. 2012. *Elements of information theory*. John Wiley & Sons.

[13] Omkar Dandekar, William Plishker, Shuvra Bhattacharyya, and Raj Shekhar. 2008. Multiobjective optimization of FPGA-based medical image registration. In *2008 16th International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 183–192.

[14] Omkar Dandekar and Raj Shekhar. 2007. FPGA-accelerated deformable image registration for improved target-delineation during CT-guided interventions. *IEEE Transactions on Biomedical Circuits and Systems* 1, 2 (2007), 116–127.

[15] Pablo A Estévez, Michel Tesmer, Claudio A Perez, and Jacek M Zurada. 2009. Normalized mutual information feature selection. *IEEE Transactions on Neural Networks* 20, 2 (2009), 189–201.

[16] James D Foley, Foley Dan Van, Andries Van Dam, Steven K Feiner, John F Hughes, Edward Angel, and J Hughes. 1996. *Computer graphics: principles and practice*. Vol. 12110. Addison-Wesley Professional.

[17] Derek LG Hill and David J Hawkes. 2000. Across-modality registration using intensity-based cost functions. *Handbook of Medical Imaging: Processing and Analysis* (2000), 537–553.

[18] Kei Ikeda, Fumihiko Ino, and Kenichi Hagihara. 2014. Efficient acceleration of mutual information computation for nonrigid registration using CUDA. *IEEE Journal of Biomedical and Health Informatics* 18, 3 (2014), 956–968.

[19] Ryan Kastner, Janarbek Matai, and Stephen Neuendorffer. 2018. Parallel programming for FPGAs. *arXiv preprint arXiv:1805.03648* (2018).

[20] Taejung Kim and Yong-Jo Im. 2003. Automatic satellite image registration by combination of matching and random sample consensus. *IEEE Transactions on Geoscience and Remote Sensing* 41, 5 (2003), 1111–1117.

[21] Flavio Lichtenstein, Fernando Antoneli, and Marcelo RS Briones. 2015. MIA: Mutual Information Analyzer, a graphic user interface program that calculates entropy, vertical and horizontal mutual information of molecular sequence sets. *BMC Bioinformatics* 16, 1 (2015), 409.

[22] Bradley Christopher Lowekamp, David T Chen, Luis Ibáñez, and Daniel Blezek. 2013. The design of SimpleITK. *Frontiers in neuroinformatics* 7 (2013), 45.

[23] Frederik Maes, Andre Collignon, Dirk Vandermeulen, Guy Marchal, and Paul Suetens. 1997. Multimodality image registration by maximization of mutual information. *IEEE Transactions on Medical Imaging* 16, 2 (1997), 187–198.

[24] Inc MathWorks. 1994-2020. MATLAB, Image Processing Toolbox. https://mathworks.com/products/image.html

[25] National Cancer Institute Clinical Proteomic Tumor Analysis Consortium (CPTAC). 2018. Radiology Data from the Clinical Proteomic Tumor Analysis Consortium Lung Adenocarcinoma [CPTAC-LUAD] collection [Data set]. *The Cancer Imaging Archive* (2018). https://doi.org/10.7937/k9/tcia.2018.pat12tbs

[26] Francisco PM Oliveira and Joao Manuel RS Tavares. 2014. Medical image registration: a review. *Computer Methods in Biomechanics and Biomedical Engineering* 17, 2 (2014), 73–93.

[27] Ramtin Shams, Parastoo Sadeghi, Rodney Kennedy, and Richard Hartley. 2010. Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images. *Computer methods and programs in biomedicine* 99, 2 (2010), 133–146.

[28] Ramtin Shams, Parastoo Sadeghi, Rodney A Kennedy, and Richard I Hartley. 2010. A survey of medical image registration on multicore and the GPU. *IEEE Signal Processing Magazine* 27, 2 (2010), 50–60.

[29] Ioannis Stratakos, Dimitrios Gourounas, Vasileios Tsoutsouras, Theodore Economopoulos, George Matsopoulos, and Dimitrios Soudris. 2019. Hardware Acceleration of Image Registration Algorithm on FPGA-based Systems on Chip. In *Proceedings of the International Conference on Omni-Layer Intelligent Systems*. 92–97.

[30] Colin Studholme. 1997. *Measures of 3D medical image alignment*. Ph.D. Dissertation. University of London.

[31] Martin Styner, Christian Brechbuhler, G Szckely, and Guido Gerig. 2000. Parametric estimate of intensity inhomogeneities applied to MRI. *IEEE transactions on medical imaging* 19, 3 (2000), 153–165.

[32] Petra A Van den Elsen, E-JD Pol, and Max A Viergever. 1993. Medical image matching-a review with classification. *IEEE Engineering in Medicine and Biology Magazine* 12, 1 (1993), 26–39.

[33] Paul Viola and William M Wells III. 1997. Alignment by maximization of mutual information. *International Journal of Computer Vision* 24, 2 (1997), 137–154.

[34] Peiyuan Wang, Yong Fan, Lingfei Lu, Lu Liu, Lingling Fan, Mengyao Zhao, Yang Xie, Congjian Xu, and Fan Zhang. 2018. NIR-II nanoprobes in-vivo assembly to improve image-guided surgery for metastatic ovarian cancer. *Nature Communications* 9, 1 (2018), 1–10.

[35] Xilinx. 2016. PYNQ: Python for Productivity for ZYNQ. http://www.pynq.io/.

[36] Ziv Yaniv, Bradley C Lowekamp, Hans J Johnson, and Richard Beare. 2018. SimpleITK image-analysis notebooks: a collaborative environment for education and reproducible research. *Journal of digital imaging* 31, 3 (2018), 290–303.

[37] Xiyang Zhi, Junhua Yan, Yiqing Hang, and Shunfei Wang. 2019. Realization of CUDA-based real-time registration and target localization for high-resolution video images. *Journal of Real-Time Image Processing* 16, 4 (2019), 1025–1036.