Faber: a Hardware/Software Toolchain for Image Registration

Eleonora D'Arnese, *Student Member, IEEE,* Davide Conficconi, *Member, IEEE,* Emanuele Del Sozzo, *Member, IEEE,* Luigi Fusco, Donatella Sciuto, *Fellow, IEEE,* and Marco D. Santambrogio, *Senior Member, IEEE*

Abstract—Image registration is a well-defined computation paradigm widely applied to align one or more images to a target image. This paradigm, which builds upon three main components, is particularly compute-intensive and represents many image processing pipelines' bottlenecks. State-of-the-art solutions leverage hardware acceleration to speed up image registration, but they are usually limited to implementing a single component. We present Faber, an open-source HW/SW CAD toolchain tailored to image registration. The Faber toolchain comprises HW/SW highly-tunable registration components, supports users with different expertise in building custom pipelines, and automates the design process. In this direction, Faber provides both default settings for entry-level users and latency and resource models to guide HW experts in customizing the different components. Finally, Faber achieves from 1.5× to 54× in speedup and from 2× to 177× in energy efficiency against state-of-the-art tools on a Xeon Gold.

Index Terms—HW/SW Design Automation, Image Registration, FPGAs.

1 Introduction

In the imaging field, taking multiple images and integrating their contents to obtain more comprehensive information is the basic idea behind various image enhancement and data integration approaches [1], [2], [3]. A representative example is a surgical navigation system that guides surgeons during their tasks, thanks to image-based information. Such a case requires the integration of preoperative images into the ones acquired during the procedure. However, the subsequent fusion produces poor or even incorrect results without correctly aligning these images to the same reference system. For this reason, multiple approaches employ *image registration (IR)*, a well-known paradigm that maps one or more images, taken under different conditions, to a reference [4]. Traditional IR comprises three main components: a *geometric transformation*, an *optimizer*, which searches the best transformation parameters, and a *similarity metric*, which explains the goodness of the transformation [5].

As a fundamental context-specific pre-processing technique, many proposed IR solutions embody either flexibility or high performance. On the one hand, pure software (SW) solutions offer several implementations of the three main IR components, e.g., mutual information (MI) and cross-correlation for the similarity metric. For instance, SimpleITK [6] devises a large corpus of implemented algorithms to give users the freedom to build custom IR pipelines. Nevertheless, it requires programming knowledge and a deep understanding of the algorithm to tune the hyperparameters. On the other hand, hardware (HW) accelerators are gaining traction as an alternative for higher performance and energy efficiency with two main approaches: GPU-based and FPGA-based [3], [7], [8], [9], [10], [11], [12]. The former is a valuable solution when images, particularly volumes, are involved, but it lacks energy efficiency. The latter provides a good trade-off between performance and energy efficiency, though it is harder to program. Despite these benefits, few HW-based methods provide open-source and easy-to-use HW-accelerated solutions for IR. Moreover, they usually offer little customization, losing the flexibility SW libraries offer. Similarly, they often require HW knowledge, limiting their adoption by non-HW scientists. Thus, the State of the Art currently lacks a comprehensive and customizable solution that combines the advantages of SW and HW approaches and supports a wide range of users, from domain experts to HW specialists.

To overcome the current literature limitations and bridge the gap between the flexibility of SW libraries and the benefits of accelerated HW approaches, we extend our previous work focused on a single similarity metric (i.e., MI) [10], and propose Faber. To the best of the authors' knowledge, Faber is the first open-source[1] HW/SW toolchain tailored to IR, offering optimized and highly customizable FPGA-based accelerators. Faber targets users with different expertise levels and enables them to customize multiple HW/SW aspects of the target IR pipeline. On the one hand, Faber embeds the flexibility of selecting and configuring various algorithms for each of the three main image registration components. On the other, it eases the HW design by automatically integrating and configuring the HW accelerators.

• *The authors are with the Department of Electronic, Information and Bioengineering, Politecnico di Milano, Milano, IT, 20133. E-mail: eleonora.darnese@polimi.it, davide.conficconi@polimi.it, emanuele.delsozzo@polimi.it, luigi1.fusco@mail.polimi.it, donatella.sciuto@polimi.it, marco.santambrogio@polimi.it*

DOI: 10.1109/TPDS.2022.3218898

1. https://github.com/necst/faber_fpga

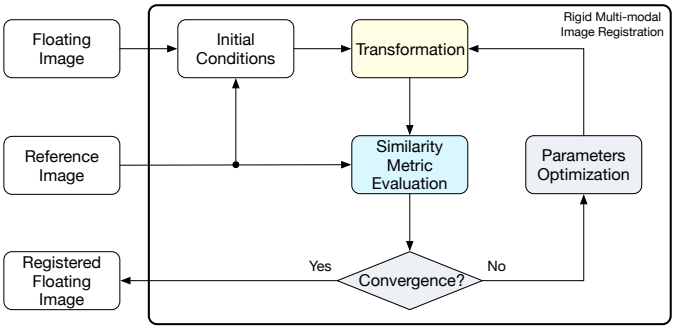


Fig. 1: A rigid multi-modal image registration workflow: it registers a floating image to a reference one.

The main contributions of this work are the following:

- The first open-source HW/SW toolchain[1] to automatically create custom IR pipelines (Section 4) exploiting FPGA-based accelerators (Section 4.2).
- Three levels of customization hyperparameters to support users in building IR pipelines (Section 4.4).
- A design automation methodology for non-FPGA experts to exploit default HW configurations as off-the-shelf SW (Section 4.4).
- A latency and resource model to guide HW expert users during the customization of the HW accelerators (Section 5).

Faber achieves up to 54× in speedup and 177× in energy efficiency improvements over State of the Art (Section 6).

2 Context Definition

The goal of IR is to align multiple images to a reference. Different approaches can be clustered based on *image modality* (mono or multi), and two classes of *geometrical transformation* (rigid or deformable) [5], [13]. Image modality denotes whether the input images come from the same sensor or not, while the type of transformation preserves the euclidean distance between each points pair or not. We focus on rigid transformation, which is also the basis for deformable registration. Moreover, we consider multi-modal techniques only, as they are more interesting than mono-modal ones for their ability to fuse different information sources into a single one. Based on the diverse origins of the employed images, classical solutions exploiting landmarks point or geometric correspondences are unfeasible, requiring an intensity-based approach.

As depicted in Figure 1, multi-modal IR is a heuristic that builds upon three main components: a geometric transformation, an optimization procedure, and a similarity metric. This heuristic searches a space to identify the best geometric transformation parameters to align the images based on the similarity metric and optimization procedure. Rigid registration comprises diverse transformations, which, in order of complexity, range from a pure translation to a complete affine, which, for 2D images, requires the identification of six parameters [13]. In this configuration, the most employed metrics are *cross-correlation*, *mean square error*, and *mutual information* [1] or its normalized version with non-parametric Parzen windows [14]. As for the optimization algorithms, *evolutionary strategies* [15] and *Powell's method* [16] are among the most used ones [1]. These components properly combined generate a registration pipeline that can be tailored to various applications.

TABLE 1: Summary of Literature Work.

	Platform	Open/Closed Source	Programming Knowledge	Type
SW	MATLAB [18]	Closed	Partial	Toolchain
	SimpleITK [6]	Open	Full	Toolchain
	SimpleElastix [17]	Open	Full	Toolchain
HW	GPU [21]	Open	Full	Single Configuration
	GPU [22]	Open	Full	Single Configuration
	GPU [24]	Open	Full	Single Configuration
	GPU [23]	Open	Full	Single Configuration
	FPGA [7]	Closed	Full	Single Configuration
	FPGA [19]	Closed	Full	Single Configuration
	FPGA [27]	Closed	Full	Single Configuration
	FPGA [10]	Open	Full	Single Configuration
	Faber (this work)	**Open**	**Partial**	**Toolchain**

3 Related Work

In the IR panorama, there are two main approaches: SW libraries and HW-based solutions. The first category includes SimpleITK [6], SimpleElastix [17], and MATLAB [18]. The first two are open-source and based on the Insight Toolkit (ITK), while the last one is closed-source and offers specific functions for IR. SimpleITK and SimpleElastix provide a wide range of algorithms and require the user to know one of the supported programming languages. At the same time, MATLAB proposes fewer algorithms and allows the user to register via GUI or via custom scripts.

Concerning the HW-based solutions, different efforts go towards the acceleration of the similarity metric computation and the transformation function, which are the most compute-intensive parts, being the optimizer mainly a lightweight control task [7], [19]. Shams et al. present a GPU-based bitonic sort and count approach for accelerating MI [20], which is then used by Ikeda at al. to develop a CUDA-based solution for part of MI computation [21]. Other researchers develop a CUDA-based acceleration of a novel spatially region-weighted correlation ratio (SRWCR) to achieve nonrigid image registration [22]. Others present a GPU-accelerated version [23], [24] integrated in open-source software frameworks. For instance, Bhosale et al. propose a stochastic gradient descent-based image registration tailored to GPU exploiting coalesced memory access to implement the image warping [23], integrated in SuperElastix [25]. Brunn et al. offer the acceleration of the interpolation and metric differentiation for 3D diffeomorphic registration [24], integrated in CLAIRE [26]. On the FPGA side, different approaches focus on accelerating similarity metrics, e.g., correlation [19] and MI [7]. Chakraborty et al. accelerated the transformation model estimation to perform a CORDIC-based rigid registration [27]. While these solutions proved the effectiveness of HW-based approaches to offload the most compute-intensive steps, they miss the flexibility of SW-like approaches. Indeed, they accelerate a single algorithm, preventing the development of other registration procedures and user customization. Differently, we proposed an open-source generator of MI accelerators to ease the integration in different solutions [10]. However, our work focused on MI acceleration only.

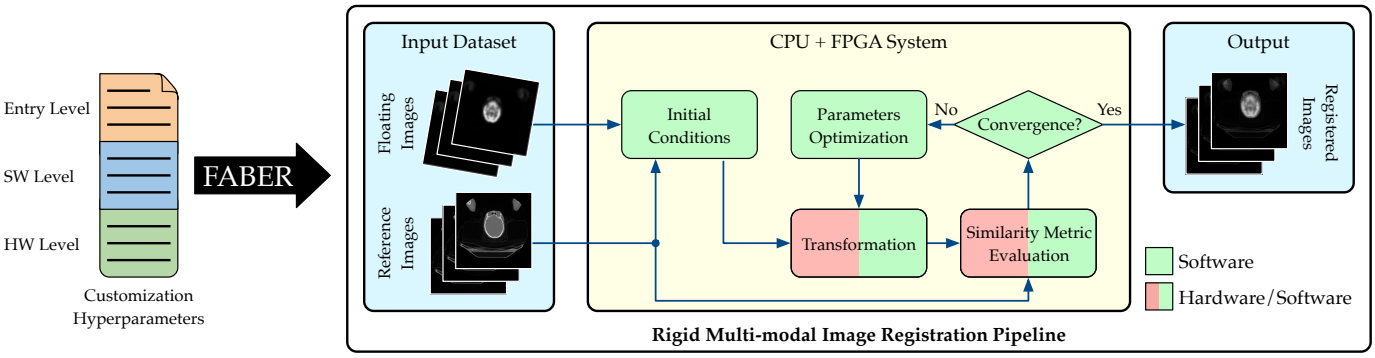


Fig. 2: Overview of the entire Faber flow, from the user's input to the final Image Registration pipeline.

Faber builds upon our previous work [10] and widens the range of solutions to overcome literature limitations and push toward the flexibility of SW library approaches. Thus, it provides the users with multiple open-source HW/SW implementations of the IR components. In particular, Faber is a complete HW/SW IR toolchain that comprehends different SW optimizers and various HW accelerators for both the similarity metric and the transformation block. Users can combine and customize such components at different levels according to their expertise. In this way, Faber supports a broad range of users, from non-FPGA ones to domain experts, and enables building IR pipelines tailored to specific requirements. Unlike many literature solutions, which only provide a single combination of IR components, we broaden the possible solution spectrum. To the best of our knowledge, no other hardware-based state-of-the-art approach offers such a user experience. Table 1 summarizes how Faber differs from literature solutions.

4 Faber Toolchain and System Design

This Section presents the components of the proposed toolchain. We first analyze Faber's main features (Section 4.1). Then, we describe the architecture of Faber HW accelerators (Section 4.2) and the architectural template we devised (Section 4.3). Finally, we define the supported customization levels (Section 4.4).

Figure 2 displays the entire Faber flow. Faber takes the user's customization hyperparameters from the command line and generates a rigid Image Registration (IR) pipeline comprising a transformation, similarity metric, and optimizer. Based on their knowledge and expertise, the users have access to different degrees of customization (i.e., Entry, SW, and HW), which enable selecting among various HW/SW implementations and algorithms for the pipeline components. In particular, the users can customize the entire pipeline or just part of it, keeping default settings for the remaining portion. Indeed, Faber also supplies default settings for the customization hyperparameters based on our latency and resource models, which we will show in Section 5. After selecting the hyperparameters, Faber automates both the pipeline design process and the synthesis of HW components. Finally, Faber outputs both the FPGA bitstream file and high-level Python APIs, implementing the whole procedure to register images and abstracting the accelerator management on the supported FPGAs.

It is important to note that, according to the literature [7], [10], [19], the similarity metric is the most time-consuming computation, followed by the transformation, while the optimizer is mainly a control task involving parameters update. Literature profiling of IR algorithms shows that the combination of transformation and similarity metric accounts for 99% of the overall time [19]. Our profiling analysis shows that this combination accounts for 84% to 99% of the IR procedure, highlighting a minimum of 74% for the similarity metric only, which aligns with literature results. Thus, we provide only FPGA-based versions of the supported transformation and similarity metric algorithms.

4.1 Faber Components

Faber offers multiple HW/SW implementations of the three main components of rigid multi-modal IR, as shown in Figure 3. Faber combines the user's customization hyperparameters to build the target pipeline. This Section describes the various components available within Faber.

4.1.1 *Optimizer Component*

Faber provides widely employed optimizers for two main classes of algorithms: gradient-free methods and evolutionary. Based on different criteria, these optimizers iteratively leverage the transformation and similarity metric components to converge to a parameter set that identifies the roto-translation matrix required to register the floating image to the reference one.

As *gradient-free* algorithm, we chose **Powell's method** [28] because is a robust direction-set method [29], which takes a vector of directions, i.e., the parameters, and iteratively optimizes each parameter with a bi-directional search algorithm (*golden section search* in Faber) in a given range.

The *evolutionary family*, starting from a parent vector, represented by the initial transformation values, generates children introducing a mutation in the population. We select the **1+1**, which, at each iteration, generates one child, and a gaussian random generator to rule the mutation [15].

Based on their mathematical formulation, Powell's is computationally heavier than 1+1. Indeed, one 1+1 iteration calls the metric and the transformation once, while a single Powell iteration computes both more than 200 times based on the initial degree of misalignment. Even though Powell converges in fewer iterations, it takes, on average, 2× the time of 1+1 to terminate fixing the metric and transformation execution time.

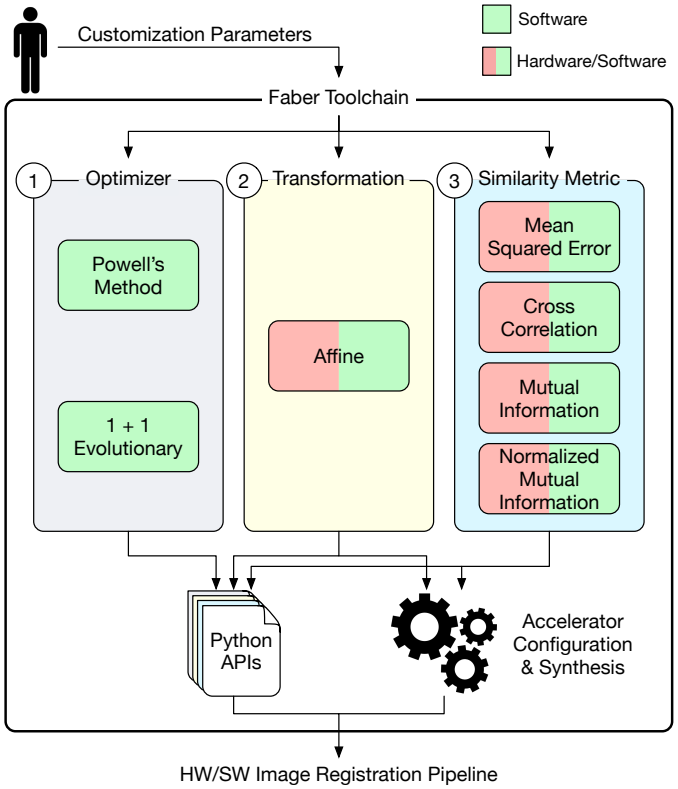


Fig. 3: Overview of Faber toolchain showing the various HW/SW components, the interaction with the user, and the generation flow of the registration pipeline.

4.1.2 Transformation Component

This component applies a rigid transformation, such as translation and rotation, to an input image, which means geometrically transforming the Euclidean image space while keeping lines parallelism, as $dst = M \cdot src$, where M is the 3×2 transformation matrix and dst is the src transformed.

4.1.3 Similarity Metric Component

The similarity metric provides a quantitative measure of how well the optimizer moves in the search space of the transformation parameters and represents the most intensive component. Since there is no a priori knowledge of which metric is optimal for a given scenario, Faber provides multiple metrics in their HW and SW versions, namely *Mean Squared Error*, *Cross-Correlation*, *Mutual Information*, and *Normalized Mutual Information*.

Mean Squared Error (MSE) computes the mean squared pixel-wise difference of the input images outputting the final value, as: $MSE(X,Y) = \frac{1}{N}\sum_{i=1}^{N}(X_i - Y_i)^2$, where X_i and Y_i are the i-th pixels of the input images, and N is the number of pixels.

Cross-Correlation (CC) measures the similarity of two signals. Faber implements a pixel-wise CC, which is then normalized by the square root of the autocorrelation of the input images [6], as: $CC(X,Y) = -1 \cdot \frac{\sum_{i=1}^{N}(X_i \cdot Y_i)}{\sqrt{\sum_{i=1}^{N} X_i^2 \cdot \sum_{i=1}^{N} Y_i^2}}$, where X_i and Y_i are the i-th pixels of the input images, and N refers to the number of pixels.

Mutual Information (MI) is a similarity metric from information theory that measures the statistical dependence of two random variables without a priori knowledge of the kind of dependence. At first, Faber computes both the single and joint (bi-dimensional) histograms of the input images. Then, it extracts the probability density functions from image histograms and calculates their entropies. Finally, it computes the MI as: $MI(X,Y) = E(X) + E(Y) - E(X,Y)$, where $E(X)$ and $E(Y)$ are the single entropies of the input images, whereas $E(X,Y)$ is the joint one.

Faber also supplies a **Normalized Mutual Information** (NMI), which estimates the probability density functions using the Parzen window method [14]. In particular, NMI convolves the histograms to extract the probability density functions with kernel derived from B-spline functions.

4.1.4 Software Implementation

The SW versions of each component are implemented in Python. In particular, the transformation relies on the `warpAffine` function from the OpenCV library, while the similarity metrics on Numpy.

4.2 Faber Hardware Accelerators

Faber offers HW implementations of the transformation and similarity metric components. We chose them due to their computational intensity. The accelerators are customizable in multiple ways according to the hyperparameters selected by the user (Section 4.4). Generally, each HW accelerator works in a pipelined dataflow fashion, and the pipeline stages communicate via FIFOs.

4.2.1 Transformation Accelerator

The HW transformation component is based on a modified set of FPGA-based kernels coming from the Xilinx Vitis Vision Library [30], which we adapted to enable seamless integration with the HW similarity metrics. The accelerator implements a generic component for affine transformations, thus fully covering the rigid registration scenario. It operates in a streaming fashion and reads the input floating image and the transformation matrix from the off-chip memory, processing one pixel per clock cycle. To apply the affine transformation and compute one output row of the transformed image, the accelerator has to access many different rows of input data. For this reason, the accelerator starts the computation after storing some input image rows in BRAMs in advance. In our case, we chose to store 100 rows, which is also the default value proposed by Xilinx, as it proved to be a valid trade-off between result accuracy and BRAM usage. In addition, the accelerator supports two different user-configurable interpolation methods, namely bi-linear and nearest-neighbor. Finally, the accelerator can either write the output image back to the off-chip memory or stream it to the similarity metric accelerator.

4.2.2 Similarity Metric Accelerators

Faber supplies a HW implementation for all the supported similarity metrics. Each accelerator exploits the map-reduce computational pattern, which remarkably boosts the performance at the cost of resource usage. In particular, every HW metric may contain a tunable number of multi-stage Processing Elements (PEs) that independently work on an input data portion. The PE number influences the internal

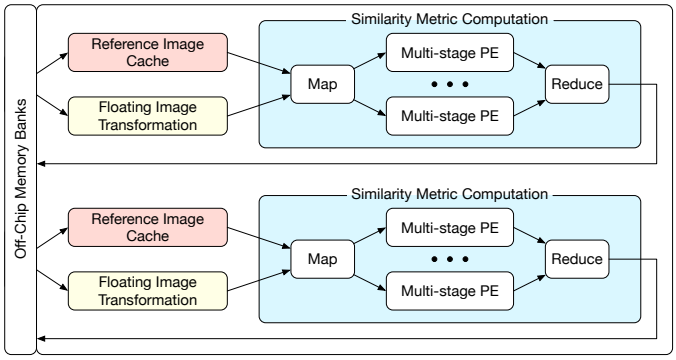


Fig. 4: An example of the Faber architectural template wrapping two HW pipelines employing transformation (yellow box) and similarity metric (light blue box) accelerators.

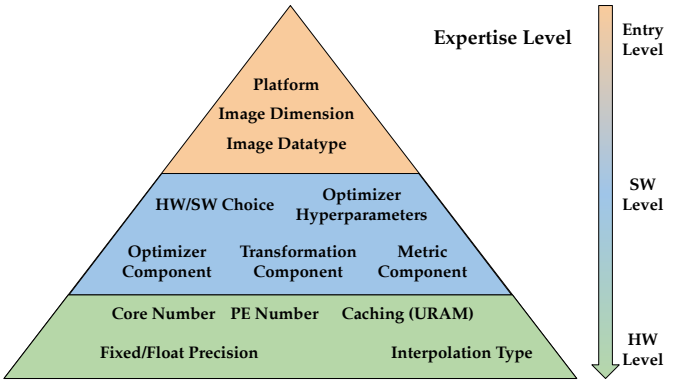


Fig. 5: The three levels of customization hyperparameters available within Faber to match user needs and expertise.

parallelism degree. It depends on the number of pixels coalesced in a packet read from the off-chip memory (or the transformation accelerator) per clock cycle. The accelerator unpacks the pixels within the input packet, maps them to different PEs, and then reduces the PE outputs.

The *MSE* accelerator leverages the map-reduce pattern to parallelize the computation, mapping pairs of pixels from the reference and floating images to the available PEs, which perform the squares of differences. The accelerator then reduces the PE outputs and calculates the MSE.

The *CC* accelerator operates similarly. Indeed, each PE receives a pair of pixels and computes a partial contribution to the final pixel-wise CC and the input image autocorrelations. The reduce step collects such contributions and calculates the CC value.

Concerning *MI*, we rely on our state-of-the-art open-source accelerator, which already offers various customization parameters [10]. Such an accelerator exploits the map-reduce pattern twice to calculate the MI. First, given the input images, the accelerator unpacks the coalesced pixels and maps them onto different PEs, which compute a partial joint histogram. Then, after reducing the joint histogram and extracting the single histograms, the accelerator also parallelizes the entropy computations and reduces their partial values to obtain the final MI. While the map-reduce pattern is applicable to the two accelerator macro stages, in this work, we focus mainly on the joint histogram parallelization, as our previous work demonstrated as the most performance-impacting parameter [10].

Starting from this MI structure, we implemented the accelerator for the *NMI*. In particular, after the joint histogram reduction, this component applies a $K \times K$ convolution kernel to extract the probability density function. This step reads and stores the first $K - 1$ rows of the joint histogram within a line buffer and then starts convolving the input. The following steps extract the single histograms, compute the entropies, and, eventually, the NMI.

We parametrized the described HW components to automate the customization process. Such automation involves instantiating the PEs, connecting the output of the HW transformation block to the HW similarity metric, and so on. In this way, as the accelerator implementation depends on the user's requirements, Faber can efficiently adapt the initial parametrized design and its internals accordingly. For instance, given the input data type, Faber automatically tunes the various internal bitwidths to prevent overflow and reduce resource usage. Section 4.4 describes the HW customization hyperparameters available within Faber.

4.3 Architectural Template

We devised an architectural template that wraps the described accelerators. This template implements standard interfaces that facilitate the design automation process and the interaction with the software APIs in charge of managing the FPGA. Indeed, the architectural template standardizes the HW/SW interfaces with an AXI-lite for control and AXI-masters to read the images (one if using caching, two otherwise) and write the similarity metric. Besides, such interfaces permit the seamless connection of both the transformation and similarity metric if the user selects to have both in HW. Finally, since the reference image is an immutable object when two images are registered (Figure 1), we designed an optional cache within the template to prefetch the reference image and thus increase data locality, avoiding energy wasting in off-chip memory access. Given the proposed architectural template, the user can choose to replicate Faber accelerator building a multi-core architecture. Each core has its physical or logical link (depending on the number of available physical ports) to the off-chip memory to retrieve the input. Figure 4 shows an example of the template.

4.4 Customization Hyperparameters

Figure 5 illustrates Faber's customization hyperparameters ordered by users' expertise level, from novice (top) to HW experts (bottom). This feature is crucial for enabling the generation of customized HW-based IR procedures. Indeed, given the customization hyperparameters, Faber co-designs a pipeline that seamlessly integrates HW and SW components. The outcome is an IR flow in Python that exploits HW acceleration abstracting all the management aspects.

The broad number of hyperparameters prevents an exhaustive design space exploration in a reasonable time, given that an HW synthesis may require several hours. For this reason, to speed up this process and support different kinds of users, Faber offers three levels of customization hyperparameters, each one expanding the available

features of the output image processing pipeline. In this way, Faber targets users interested in using an out-of-the-box IR algorithm, users with a basic understanding of the problem, and experienced users who want to customize the HW accelerator details. Faber employs the remaining default hyperparameters based on the selected features and automatizes the target system generation.

The *entry customization level* requires the user to specify mandatory details of the IR algorithm, namely the input image dimension, data type, and the target platform. If the user selects a non-FPGA-based platform, the output pipeline is a pure SW one. Otherwise, Faber produces a predefined HW/SW pipeline for the target FPGA.

The *SW customization level*, which is entirely optional, goes into the details of the target registration pipeline. The user can specify each component implementation and choose among HW or SW versions when available. Moreover, the user can define the optimizer hyperparameters, such as maximum iterations and the ending condition with an optimization threshold. If the user chooses to offload the metric/transformation to the FPGA, Faber takes care of configuring and generating the HW accelerator(s).

The *HW customization level* allows customization of different HW hyperparameters. First of all, the user can select the number of accelerators to instantiate on the FPGA. Given N independent cores, Faber will produce SW APIs that can register N parallel pairs of images in a multi-threaded fashion. If available, the user may then choose to enable the accelerator cache, allocated in the FPGA URAMs, or BRAMs. Another customization hyperparameter is the PE number, which affects the accelerator parallelism and is directly proportional to the size of the packet read from the off-chip memory. Finally, in specific metrics (e.g., MI), the user can specify whether to use floating- or fixed-point computations, while the transformation accelerator is tunable in the interpolation type.

Faber HW customizations are orthogonal and transparent to the SW-level ones. The only exceptions are the image dimensions and the pixel's datatype, since the final SW application and the HW accelerator(s) are both tailored for those parameters.

5 Latency and Resource Models

As previously mentioned, the HW customization level enables users, usually FPGA experts, to tune all the possible hyperparameters available within Faber, from SW to HW ones. In this way, users may decide to configure the HW accelerators instead of relying on the default ones. However, given the vast range of combinations, manual design space exploration is prohibitive, mainly due to the long synthesis time. For this reason, Faber provides latency and resource models tailored to our target domain. The main goal of such models is to assist HW-level users in rapidly evaluating a given HW accelerator for IR. Hence, the models take into account solely the hyperparameters that affect the accelerator latency and resources. In particular, the models consider all the hyperparameters in Figure 5 but the ones related to the optimizer; indeed, since the optimizer always runs on the host side, it does not alter the HW performance.

5.1 Latency Model

Faber offers a simple yet effective model for latency estimation. Even though such a model does not consider aspects impacting the final latency (e.g., off-chip memory bandwidth, pipeline warm-up), its purpose is to supply a coarse-grain measure of the number of clock cycles the HW accelerator takes to produce the output. According to the chosen metric, the presence of the transformation in HW, and the hyperparameters, the final latency may significantly vary. The circuit clock cycles mainly depend on the input image dimension, the number of PEs, and the input data type, especially for MI and NMI. Given the pipelined dataflow nature of the accelerators, we can model their latency in clock cycles as follows:

$$\begin{aligned} CC_{latency} &= D^2/PE \\ MSE_{latency} &= D^2/PE \\ MI_{latency} &= D^2/PE + (2^B)^2 \\ NMI_{latency} &= D^2/PE + (2^B + K - 1)^2 \end{aligned} \tag{1}$$

where D is the input image dimension, PE the number of PEs, B the input data bitwidth, and K the convolution kernel size. The latency decreases as the number of PEs (i.e., the parallelism level) increases. Besides, MI and NMI latencies depend on both joint histogram and entropy calculations, as they cannot overlap for data dependencies. Finally, please note that B also affects the number of instantiable PEs. Indeed, given the off-chip memory port bitwidth MBW (e.g., 512 bits), we can use, at most, MBW/B PEs before oversaturating the bandwidth. However, the more PEs, the more employed FPGA resources.

Including the transformation within the accelerator has peculiar effects on the latency. As mentioned in Section 4.2, unlike the similarity metric components, the transformation component processes a single pixel per clock cycle. Thus, this design choice implies that coalescing the input pixels does not reduce the clock cycles, causing the higher latency of the transformation to hide the part of the similarity metric latency that depends on D since these two computations overlap. The only exceptions are MI and NMI due to the previously mentioned data dependency. For these reasons, we model the composite accelerator latency as:

$$\begin{aligned} WCC_{latency} &= (D + R) \cdot D \\ WMSE_{latency} &= (D + R) \cdot D \\ WMI_{latency} &= (D + R) \cdot D + (2^B)^2 \\ WNMI_{latency} &= (D + R) \cdot D + (2^B + K - 1)^2 \end{aligned} \tag{2}$$

where D is the input image size, R the input image rows the HW transformation needs before starting computing, B the input data bitwidth, and K the convolution kernel size. Equation (2) is independent of PE, implying that increasing the number of PEs in the similarity metric and coalescing input data do not benefit the overall latency, as previously mentioned. However, moving most of the computation to HW permits reducing the IR execution time (Section 6).

5.2 Resource Model

Faber also implements a resource usage model, whose goal is to provide hints about the feasibility of the HW accelerator. Given a specific configuration, the model reports the

Algorithm 1 Count BRAM18k Blocks

Input: $Array_{size}$, $Bitwidth$
Output: $BRAM_{Blocks}$

$BRAM_{confs}$=[(18, 1024), (9, 2048), (4, 4096), (2, 8192), (1, 16384)]
1: $curr$ = $Bitwidth$
2: $BRAM_{Blocks}$ = 0
3: **for** b, s in $BRAM_{confs}$ **do**
4: q = $curr$ // b
5: $curr$ = $curr$ % b
6: $BRAM_b$ = $\lceil q \cdot Array_{size}/s \rceil$
7: $BRAM_b$ += $BRAM_b$ % 2
8: $BRAM_{Blocks}$ += $BRAM_b$
9: **end for**

resources estimated usage, indicating whether they exceed the available budget. To this end, we analyzed the usage behavior of FPGA resources and derived a model to describe their scaling according to the selected hyperparameters. For DSPs and logic resources (FFs and LUTs), we chose a linear regression model to characterize them separately as follows:

$$Resource = \alpha \cdot x + \beta \tag{3}$$

where x is the number of PEs, α the slope, and β the intercept. The model linearly depends on the number of PEs, which is the main hyperparameter affecting resource usage. Then, we selected multiple slopes and intercepts covering the other hyperparameters. In particular, for each resource, we identified a slope for each accelerator composition, while intercept values reflect the remaining hyperparameters (e.g., interpolation, data precision).

Considering on-chip memory resources (BRAMs and URAMs), we measured their usage based on the number of arrays within the accelerator and their bitwidth. In this way, we can examine various scenarios (e.g., caching enabled) adopting the same approach. A Xilinx's BRAM usually accommodates up to 36Kb and works as two independent 18Kb RAMs or a single 36Kb RAM. A BRAM supports different port width configurations (e.g., 1K $\times$ 36, 2K $\times$ 18, 4K $\times$ 8), some of which reduce the BRAM capacity. For these reasons, we implemented Algorithm 1 to measure the BRAM usage, which maps the input array to the on-chip memory employing the available configurations. This algorithm estimates BRAM usage well when dealing with large arrays (e.g., joint histogram, local cache). Conversely, we approximate the BRAM usage of small arrays (i.e., less than 1024 elements) based on empirical measurements, as we observed that Xilinx tools tend to aggregate data within a few BRAMs in such a case. On the other hand, a URAM is larger than a BRAM (it contains up to 288Kb) but supports a single 4k $\times$ 72 configuration. The following formula defines how we measure URAM usage:

$$URAMs = \sum_{i=0}^{N} S_i \cdot \lceil 72 \cdot \lceil B_i/72 \rceil / C \rceil \tag{4}$$

where N is the number of arrays, S_i their size, B_i their bitwidth, and C URAM capacity. Both Algorithm 1 and Equation (4) do not directly consider techniques like partitioning or reshaping. Nonetheless, the model is aware of how each accelerator allocates its arrays; thus, it automatically manages such scenarios.

6 Experimental Setup and Results

This Section presents the experimental setup for evaluating Faber, an analysis regarding the benefits of the transformation on FPGA, followed by the scalability analysis of the default configurations. Then, we provide a wider comparison of deployable accelerators in terms of execution time and accuracy against MATLAB and SimpleITK. We also validate the proposed models for latency and resource usage. Finally, we compare Faber with the literature.

Faber automatically generates C++ accelerators for High-Level Synthesis toolchains, namely Xilinx Vitis and Vivado HLx 2019.2. Currently, Faber targets three boards (two Zynq-based platforms, namely Ultra96v2 and ZCU104, and an Alveo u200 accelerator card), even though it can seamlessly support other Xilinx-based platforms with minor additions. SW applications are multi-threaded Python code communicating with the HW through custom APIs on top of the Pynq 2.5 framework. The code runs on Zynq ARM CPUs, which share the DDR with the reconfigurable fabric, and an Intel i7-4770 linked via PCIe to the Alveo.

Throughout the discussion, we present results on two representative classes: the Ultra96, an embedded board powered by a Zynq Ultrascale+ ZUEG3 (with the ZCU104 to compare against [10]), and the Alveo, a high-end accelerator card, with an Ultrascale+ XCU200. Moreover, we reduce the design searching space to the following hyperparameters: 32-bit floating-point for the MI and NMI accelerators, nearest neighbor as Interpolation Type, and no caching.

Our experimental evaluation targets the medical field since it would highly benefit from an IR acceleration with the simplicity of a SW application. As a testing dataset, we used a medical one [31], [32][2], composed of 512$\times$512$\times$247 Computed Tomography images and a corresponding number of Positron Emission Tomography ones, up-scaled from the original 128$\times$128 to 512$\times$512 pixels. Since most SW solutions suggest not exceeding 256 levels for the histograms in the MI computation, we down-scaled the images to 8-bit. Hence, we tailor Faber for this scenario fixing these hyperparameters as well. The images contain misalignments due to acquisition protocols and patients' movements.

We compare our solutions against both literature work and optimized state-of-the-art tools, namely MATLAB 2019b [18], and SimpleITK [6], which exploit as many cores as are available in the CPU. We run the tests on a 40-core Intel Xeon Gold 6148 CPU and compare it with Faber HW-accelerated versions. For a fair comparison, both SimpleITK and Faber implementations share the optimizer, transformation, similarity metric, and hyperparameters, while MATLAB uses 1+1 and NMI. Additionally, we measured power consumption with a Voltcraft 4000 energy logger for Ultra96 and ZCU104 and with onboard sensors for Alveo.

We define an accelerator configuration as the concatenation of the following free hyperparameters: *core number* (if missing, we assume 1), *transformation component (W)* (if missing, SW transformation), *metric component (MSE, CC, MI,*

2. Patient: C3N-00704, Study: Dec 10, 2000 NM PET 18 FDG SKULL T, CT: WB STND, PET: WB 3D AC)

TABLE 2: Performance analysis of the MI similarity metric with and without the HW transformation at 200MHz. Performance in $[\frac{ms}{MVoxels \cdot iters}]$ (the lower the better), energy efficiency in $[\frac{iters \cdot MVoxels}{kW \cdot ms}]$ (the higher the better).

Board	Config	Powell's		1+1	
		Perf.	Energy Eff.	Perf.	Energy Eff.
Ultra	2MI2	4.1158	33.75	0.0484	2888.73
	2WMI1	**2.4295**	**55.62**	**0.0296**	**4667.56**
ZCU104	2MI2	3.8770	20.31	0.0462	1717.28
	2WMI1	**2.2694**	**34.70**	**0.0290**	**2750.49**
Alveo	MI1	3.1664	10.67	0.0378	892.22
	WMI1	3.5928	9.44	0.0402	844.33
	MI2	2.2498	14.96	0.0275	1225.71
	WMI2	3.5761	9.53	0.0403	829.90
	MI4	1.8083	18.43	0.0225	1474.35
	WMI4	3.5916	9.47	0.0404	828.01
	MI8	1.5778	20.95	0.0198	1672.26
	WMI8	3.5862	9.45	0.0402	827.37
	MI16†	**1.4639**	**22.56**	**0.0183**	**1792.97**
	WMI16	3.5803	9.43	0.0404	820.12
	MI32	**1.3918**	23.14	**0.0177**	**1830.78**
	WMI32	3.5842	9.26	0.04008	813.39

†Achieves higher frequency than MI32

and NMI), and *PE number*. For instance, 2WMI1 describes a design with two cores, both transformation and similarity metric (MI) in HW, and one PE.

6.1 Benefits Analysis of the HW Transformation

To analyze the benefit of accelerating the transformation, we use the metric proposed by Shams et al. [33] ($ms/MVoxels/iters$, the lower, the better), enriched by the energy efficiency ($(MVoxels \cdot iters)/(ms \cdot kWatt)$, the higher, the better) as we previously proposed [10]. For this analysis, we concentrate on a single similarity metric (i.e., MI) for space reasons. Besides, we chose MI because the HW transform does not entirely hide its latency (along with NMI, as reported in Equation (2)). Indeed, MI has two non-overlapping macro computational stages whose overall latency surpasses the transform one. Nonetheless, we argue that we would observe a similar trend with any metric. Finally, we determined experimentally on our setup that a PYNQ-based host can take advantage of a multi-core on Zynq while the PCIe locks time-multiplexing on the Alveo.

Table 2 shows the results of our analysis. When considering Zynq-based solutions, combining the transformation with the metric accelerator improves both performance and energy efficiency. Conversely, scaling to a different processor and FPGA class shows a different situation. The processor scaling delivers remarkable improvements on the software transformation component, while the transformation accelerator does not provide gains, even when the number of PEs increases. Indeed, as indicated in Section 4.2.1 and Section 5.1 and Equation (2), the HW transform does not benefit from input data coalescing since it processes only one pixel at a time. Thus, MI-based configurations outperform WMI-based ones with both Powell's and 1+1 optimizers, especially when the PE number grows, following the performance trend described in Equation (1). For these reasons, we consider the W+MI combination as the best option (or, in general, the combined accelerator) for Zynq-based devices. At the same time, the single metric accelerator with scaled PEs is the best way whenever moving to the Alveo device, as suggested by our previous work [10].

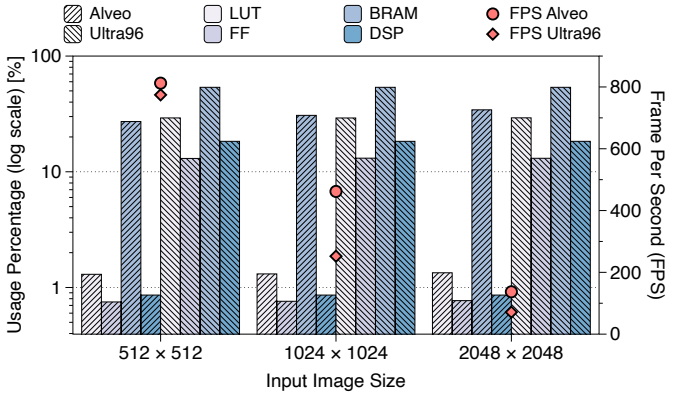


Fig. 6: Resource and FPS scaling for Faber default configurations (Powell's - MI16 for Alveo, 1+1 - 2WCC1 for Ultra96) on different image dimensions.

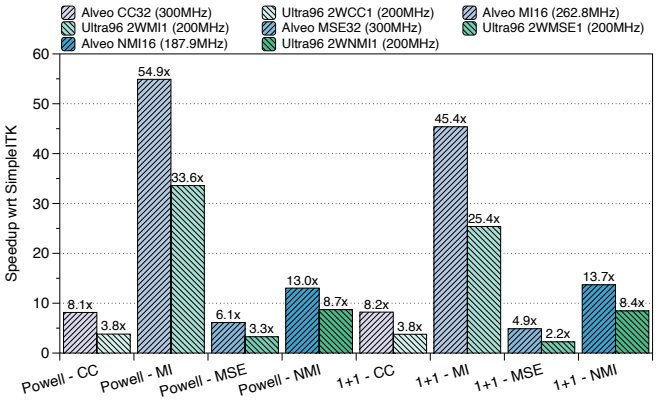


Fig. 7: Speedup comparison with SW state-of-the-art tools normalized to the respective SimpleITK registration time.

6.2 Default Configurations Discussion

Based on the previous considerations and the proposed latency and resource models, which we validated in Section 6.5, Faber provides default configurations for entry-level users, namely Powell's with MI16 for the Alveo and 1+1 with 2WCC1 for the Ultra96, which represent the best trade-offs of resources, accuracy, and execution times per platform. Figure 6 reports the resource usage and the Frame Per Second (FPS) rate of the default accelerators. Figure 6 shows how these quantities scale with increasing image dimensions, from 512×512 to 2048×2048. Resources remain primarily constant while the FPS rate decreases by around three times with a quadratic increment of the number of pixels processed since the data transfer is the bottleneck.

6.3 Performance Evaluation

Moving to the evaluation of Faber in terms of execution time, we compare our best-performing combinations of CC, MSE, MI, and NMI accelerators against SimpleITK and

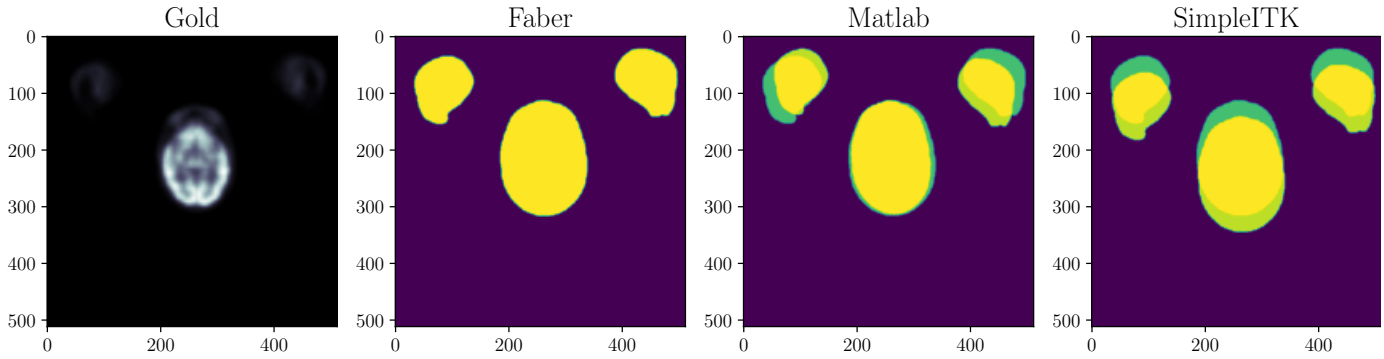


Fig. 8: A visual example of image registration results. From left to right, the gold standard, the overlap between gold standard (represented in green) and the registered image with Faber, MATLAB, and SimpleITK (all in yellow).

TABLE 3: Intersection over Union comparison between Faber and state-of-the-art SW tools for registration.

Opt. - Metric	Faber SW	Faber HW	Simple ITK	MATLAB
1+1 - CC	**0.94 ± 0.07**	**0.94 ± 0.07**	0.93 ± 0.10	-
1+1 - MI	**0.97 ± 0.03**	**0.97 ± 0.04**	0.76 ± 0.20	-
1+1 - NMI	**0.96 ± 0.04**	**0.96 ± 0.04**	0.88 ± 0.20	0.89 ± 0.08
1+1 - MSE	**0.92 ± 0.18**	**0.92 ± 0.17**	0.67 ± 0.09	-
Powell's - CC	0.93 ± 0.07	0.93 ± 0.07	**0.96 ± 0.05**	-
Powell's - MI	**0.98 ± 0.06**	**0.95 ± 0.09**	0.75 ± 0.20	-
Powell's - NMI	**0.95 ± 0.08**	**0.95 ± 0.08**	0.92 ± 0.16	-
Powell's - MSE	**0.93 ± 0.07**	**0.93 ± 0.07**	0.65 ± 0.07	-

MATLAB. In particular, since MATLAB implements a registration process based on 1+1 and NMI, our comparison with it considers that configuration only. We extend the subset presented in the previous Section, showing a superset of possible configurations. Figure 7 reports the speedup results normalized to SimpleITK registration times obtained with the Alveo and the Ultra96. The reported results account for the overall execution times; hence, we include in our results also all the data movement necessary for the IR computation. Indeed, the image pair is initially located in the host memory. Then, the application allocates buffers to the accelerator DDR memory (which may be the same physical bank with different address spaces in Zynq devices), fills them, and runs the accelerator.

Clearly, the accelerators dramatically impact registration times. As mentioned in Section 4.1, the 1+1 optimizer is more straightforward than Powell's method and less computationally intensive; thus, 1+1 is generally faster in both HW and SW versions. Faber's approach shows higher performance than state-of-the-art SW tools. Precisely, when compared to SimpleITK, Faber delivers speedups that range from ∼4.9× to ∼54.9× on Alveo and from ∼2.2× to ∼33.6× on Ultra96. Similarly, Faber reaches a 2.39× (Alveo) and 1.47× (Ultra96) speedup over MATLAB on the Xeon Gold.

6.4 Accuracy Evaluation

To validate the Faber toolchain accuracy, we compare the obtained registered images against a gold standard, extracted by a semi-automatic registration, and MATLAB and SimpleITK implementations. Figure 8 shows a first visual evaluation, where we reported the gold standard and a comparison between the binarized gold standard (in green) and the output obtained with Faber, SimpleITK, and MATLAB (in yellow), while the background is in purple. We select for the visual inspection the configuration exploiting NMI and (1+1) since it is the one available in all the considered tools.

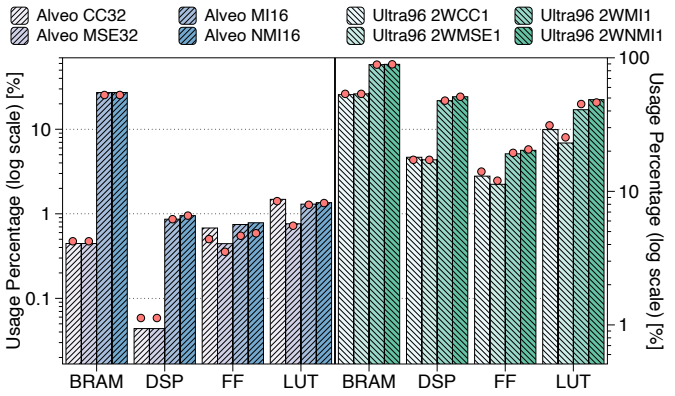


Fig. 9: Accelerator resource usage in logarithmic scale (the dot indicates the model prediction).

The quantitative validation is based on the calculation of the Intersection over Union (IoU) metric computed pairwise between the gold standard stack and the registered ones obtained by Faber, MATLAB, and SimpleITK. This metric choice over the standard definition of accuracy derives from the necessity to evaluate the overlap of the registered images on the gold standard instead of identifying identical pixel values between the two stacks. To compute IoU, we set all the content that differs from the background, which has a zero value, to one. In this way, an IoU of one indicates a perfect overlap, while zero value means no overlap and a failed registration.

As shown in Table 3, Faber, exploiting the sole metric accelerators, reaches better IoU results compared to MATLAB and SimpleITK. Moreover, our accelerators maintain the accuracy reached by their SW counterparts, providing a faster yet more accurate alternative. The inclusion of the HW-based version of the transformation impacts the metric accelerator's IoU value, introducing a mean reduction in accuracy of around 0.026. These results further support the adoption of HW accelerated toolchain in the IR field.

TABLE 4: Comparison with Related Work with Performance (Perf.) measured as proposed by Shams et al. [33] (the lower the better), and Energy Efficiency (Energy Eff.) as we proposed in our previous work [10] (the higher the better).

Arch.	Work	Transform	Metric	Optimizer	Hardware	Perf. $[\frac{ms}{MVoxels \cdot iters}]$	Energy Eff. $[\frac{iters \cdot MVoxels}{kW \cdot ms}]$
FPGA	**Faber 2WMI1**	Rigid†	MI†	Powell	Ultra96 (16nm)	2.4295⊛	55.62
	Faber 2WCC1	Rigid†	CC†	1+1	Ultra96 (16nm)	**0.0283**⊗	**5156.87**
	Faber 2WNMI1	Rigid†	NMI†	Powell	Ultra96 (16nm)	2.5255⊛	54.24
	Faber 2WNMI1	Rigid†	NMI †	1+1	Ultra96 (16nm)	0.0311⊗	4492.02
	Faber MI16	Rigid†	MI†	Powell	Alveo u200 (16nm)	1.5502⊛	21.47
	Faber CC32	Rigid†	CC †	1+1	Alveo u200 (16nm)	**0.0130**⊗	**2632.91**
	Faber NMI16	Rigid†	NMI†	Powell	Alveo u200 (16nm)	1.6876⊛	19.81
	Faber NMI16	Rigid†	NMI †	1+1	Alveo u200 (16nm)	0.01921⊗	1756.58
	Faber 3WMSE1	Rigid†	MSE†	Powell	ZCU104 (16nm)	0.7430⊛	104.33
	Faber 3WMSE1	Rigid†	MSE†	1+1	ZCU104 (16nm)	**0.0184**⊗	**4224.14**
	[7]	MultiRigid	MI†	Simplex	Altera EP2S180 (90nm)	13.4⋆	N/A
	[19]	Affine†	Corr.†	Simplex	Zybo (28nm)	9.15⊙	45.54≀
	[27]	Rigid†	RMSE†	-	ZCU104 (16nm)	0.1895♡	467.86♣
GPU	[20]	Rigid	MI†	Powell	GTX 280 (65nm)	4.06⋆	1.04‡
	[21]	Nonrigid	NMI†	Grad. Desc.†	GTX 580 (40nm)	0.1378▽◇	31.52‡
	[23]	Nonrigid†	NMI†	Stoch. Grad. Desc.†	Tesla K40c (28nm)	2.5750♠	1.58
	[22]	Nonrigid	SRWCR†¶	LBFGS	GTX 1060 (16nm)	309.7996^	0.03
	[24]	Diffeomorphic↺	L^2 Norm†¶	Gauss-Newton-Krylov	Tesla V100 (12nm)	25.5372	0.13
CPU	MATLAB PAR. TOOL	Affine	NMI	1+1	Intel Xeon Gold 6148 (14nm)	0.0457⊗	145.93‡
	Simple ITK	Rigid	NMI	Powell	Intel Xeon Gold 6148 (14nm)	0.6621⊗•	10.07‡
	Simple ITK	Rigid	NMI	1+1	Intel Xeon Gold 6148 (14nm)	0.2641⊗•	25.24‡
	Simple ITK	Rigid	MI	Powell	Intel Xeon Gold 6148 (14nm)	2.5636⊗•	2.60‡
	Simple ITK	Rigid	MI	1+1	Intel Xeon Gold 6148 (14nm)	0.7806⊗•	8.54‡
	Simple ITK	Rigid	MSE	Powell	Intel Xeon Gold 6148 (14nm)	0.1214⊗•	54.89‡
	Simple ITK	Rigid	MSE	1+1	Intel Xeon Gold 6148 (14nm)	0.0638⊗•	104.52‡
	Simple ITK	Rigid	CC	Powell	Intel Xeon Gold 6148 (14nm)	0.2297⊗•	29.03‡
	Simple ITK	Rigid	CC	1+1	Intel Xeon Gold 6148 (14nm)	0.1071⊗•	62.22‡

†Implemented in hardware ⋆Computed from [33] ⊙ Assuming maximum iteration of 500 ‡Computed with Thermal Design Power (TDP) as power ▽ Exploits the binning to reduce joint histogram sizes ⊗ With maximum 100 iterations •Metadata-based preprocessing applied ⊛ 3 iterations avg. ◇ This value is the result of several dataset-specific approximations and preprocessing that reduce the computation to 1/6 and lead to misregistrations [21] ≀Power is 2.1W (Zybo baseline) plus 0.3mW ^Based on 200 iteration ¶Metric derivatives acceleration ↺Interpolator in hardware ♣Power is 11.25W (ZCU104 baseline) plus 23.59mW ♠Working on 15% of image pixels ♡On 512×512 images and applying feature matching only

6.5 Model Validation

We validated our latency and resource models on the previously introduced design space. As stated in Section 5, the latency model provides a coarse-grain measure of the accelerator clock cycles, ignoring external aspects like the off-chip memory bandwidth. Thus, a direct comparison with the execution times on FPGA would be unfair. Instead, we chose the RTL simulation results as our baseline. We extracted such values from Vivado HLS, which performs a cycle-accurate simulation of the target accelerator only. We measured the model error as the absolute difference between the predicted latency and the simulated one, normalized to the latter. On average, the error introduced by our model is 0.939%, and it mainly derives from the pipeline warm-up clock cycles, which our model does not examine.

We employed the utilization values reported by Vivado after the RTL synthesis of various accelerators to build the resource model and validated it on the post place & route results of the previously introduced ones. We computed the model error as before, comparing the predicted and actual resource usage and normalizing them to the available resource budget. We first derived the model for Ultra96 and then ported it on Alveo u200. The Ultra96 model fits the resource usage well, and the average error is, at most, 2.56% (LUTs). Porting the model to Alveo is quite straightforward, thanks to the FPGA technology they share, namely Ultrascale+. Indeed, the model achieves accurate estimates of the resources even when keeping the Ultra96 model parameters. In this case, the average error on Alveo u200 ranges from 0.007% (DSPs) to 3.570% (BRAMs). The higher BRAM error derives from a larger usage than predicted on the NMI16 configuration, which does not occur on the Ultra96. A possible reason is the multi-die structure of Alveo cards. Indeed, Vitis may decide to spread the design among the dies to avoid congestion, adding extra logic. Figure 9 shows the accelerator resource usage and model predictions.

Finally, we investigated possible corner cases within the resource model. We noticed that the error increases for some resources when we consider unfeasible HW designs. The main reason is that Vivado tries to optimize such designs further if they exceed the resource budget. For instance, being BRAMs the critical resource for MI and NMI, Vivado shares their usage among different submodules of the accelerator. This behavior is hardly predictable as it heavily depends on the internal closed-source synthesis algorithms of Vivado. Thus, the model does not consider such optimizations, conservatively predicting that the design is unfeasible.

6.6 Comparison with related work

To compare Faber to other solutions in the literature, we have selected a subset of metric combinations per board exploiting Powell's and 1+1 optimizers to facilitate a comprehensive discussion. Fairly comparing IR methodologies is extremely difficult considering all the variable parameters that impact the results, like different platforms, transformations, optimizers, metrics, and datasets. For this reason,

we use the metric described in Section 6.1, as it tries to normalize the differences among approaches (e.g., IR components, dataset size, execution time). However, this metric does not consider initial dataset misalignments, the different difficulties in aligning anatomic parts, and the hyperparameters (e.g., convergence threshold). We also know that a fair architectural comparison is difficult; hence, we reported the technology nodes. Still, many factors limit the fairness of the comparison. For instance, open-source solutions are limited, and when available, they are either tailored to a specific dataset (not provided) or employ deprecated features. Consequently, these aspects limit reproducibility and portability to newer platforms.

Table 4 shows the comparison with literature work on both FPGA and GPU, while, for the CPU, we report two examples that exploit MATLAB (with the Parallel Toolbox) and SimpleITK. Considering FPGA-based work, Faber shows better performance, also providing SW-like flexibility. In addition, if we focus on the work proposed by [27] and compare it to Faber on the same board, namely ZCU104, we can see how the 1+1 implementation provides better results. Moreover, while [27] exploits a feature-based approach for mono-modal IR, we propose a heuristic-based multi-modal IR that implies more computations to reach the final results.

Considering GPU-based solutions, Faber's best configurations employing the same similarity metric (Ultra96 1+1-WNMI1 and Alveo 1+1-WNMI1) achieve better performance than the top-GPU one, even without precomputations, from 2.33× to 4.37×. Focusing on work employing newer GPUs, they also present different transformations compared to Faber, which automatically implies that the comparison is hindered fairly. We can highlight that such studies [22], [23], [24], even though they require more computations, achieve lower results of performance and energy efficiency. Indeed, the benefits of GPU acceleration are attractive only when all the available computational resources are employed, justifying the power consumption. Unfortunately, this is not the IR case, considering the limited number of images constituting medical volumes.

Finally, Faber achieves generally better performance than its CPU-based counterparts. However, considering Powell's optimizer, the reader can notice how the performance of SimpleITK seems slightly better than Faber, even though Figure 7 shows a significant speedup of Faber. This performance difference depends on the large impact of the optimizers' iterations and internals searching procedures. Nevertheless, Faber outperforms CPU counterparts in energy efficiency (reaching up to 30.78× and 177.97× against MATLAB and SimpleITK, respectively) and accuracy.

7 Conclusion

We presented Faber, an open-source HW/SW toolchain for IR that offers optimized FPGA-based accelerators. Faber is devised to flexibly combine the three main image registration components and customize them. Moreover, it offers various levels of customization based on the user's expertise, also providing HW experts with latency and resource models guiding the customization of the different hyperparameters. We evaluate Faber against SW library-like approaches achieving from 1.5× to 54× and from 2× to 177× in speedup and energy efficiency. Faber showcases a noticeable registration accuracy compared to state-of-the-art solutions with a top of 0.97 of IoU while delivering improved execution times and energy efficiencies. Faber paves the way for seamless integration of FPGAs within standard and custom IR pipelines and makes them accessible to a wider audience.

Acknowledgments

Data used in this publication were generated by the National Cancer Institute Clinical Proteomic Tumor Analysis Consortium (CPTAC). The Authors would like to thank the Xilinx University Program for the hardware donations.

References

[1] F. P. Oliveira and J. M. R. Tavares, "Medical image registration: a review," *Computer methods in biomechanics and biomedical engineering*, vol. 17, no. 2, pp. 73–93, 2014.

[2] E. D'Arnese, G. Di Donato, E. Del Sozzo, and M. D. Santambrogio, "Towards an automatic imaging biopsy of non-small cell lung cancer," in *2019 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*. IEEE, 2019, pp. 1–4.

[3] K. C. Alpay, K. B. Aydemir, and A. Temizel, "Accelerating translational image registration for hdr images on gpu," *arXiv preprint arXiv:2007.06483*, 2020.

[4] M. V. Wyawahare, P. M. Patil, H. K. Abhyankar *et al.*, "Image registration techniques: an overview," *International Journal of Signal Processing, Image Processing and Pattern Recognition*, vol. 2, no. 3, pp. 11–28, 2009.

[5] L. G. Brown, "A survey of image registration techniques," *ACM computing surveys (CSUR)*, vol. 24, no. 4, pp. 325–376, 1992.

[6] B. C. Lowekamp, D. T. Chen, L. Ibáñez, and D. Blezek, "The design of simpleitk," *Frontiers in neuroinformatics*, vol. 7, p. 45, 2013.

[7] O. Dandekar and R. Shekhar, "Fpga-accelerated deformable image registration for improved target-delineation during ct-guided interventions," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 1, no. 2, pp. 116–127, 2007.

[8] O. Fluck, C. Vetter, W. Wein, A. Kamen, B. Preim, and R. Westermann, "A survey of medical image registration on graphics hardware," *Computer methods and programs in biomedicine*, vol. 104, no. 3, pp. e45–e57, 2011.

[9] M. Barrow, S. M. Burns, and R. Kastner, "A fpga accelerator for real-time 3d non-rigid registration using tree reweighted message passing and dynamic markov random field generation," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018, pp. 335–3357.

[10] D. Conficconi, E. D'Arnese, E. Del Sozzo, D. Sciuto, and M. D. Santambrogio, "A framework for customizable fpga-based image registration accelerators," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 251–261.

[11] E. Del Sozzo, D. Conficconi, A. Zeni, M. Salaris, D. Sciuto, and M. D. Santambrogio, "Pushing the level of abstraction of digital system design: a survey on how to program fpgas," *ACM Computing Surveys (CSUR)*.

[12] E. D'Arnese, E. Del Sozzo, D. Conficconi, and M. D. Santambrogio, "Exploiting heterogeneous architectures for rigid image registration," in *2021 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 2021, pp. 1–5.

[13] B. Zitova and J. Flusser, "Image registration methods: a survey," *Image and vision computing*, vol. 21, no. 11, pp. 977–1000, 2003.

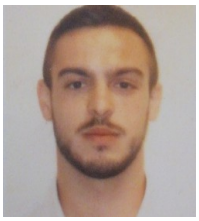
[14] E. Parzen, "On estimation of a probability density function and mode," *The annals of mathematical statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.

[15] M. Styner, C. Brechbuhler, G. Szckely, and G. Gerig, "Parametric estimate of intensity inhomogeneities applied to mri," *IEEE transactions on medical imaging*, vol. 19, no. 3, pp. 153–165, 2000.

[16] M. J. Powell, "A fast algorithm for nonlinearly constrained optimization calculations," in *Numerical analysis*. Springer, 1978, pp. 144–157.

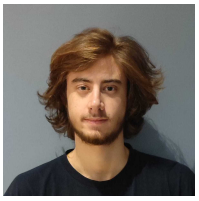
[17] K. Marstal, F. Berendsen, M. Staring, and S. Klein, "Simpleelastix: A user-friendly, multi-lingual library for medical image registration," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2016, pp. 134–142.
[18] I. MathWorks, "Image processing toolbox," 1994-2020. [Online]. Available: https://mathworks.com/products/image.html
[19] I. Stratakos, D. Gourounas, V. Tsoutsouras, T. Economopoulos, G. Matsopoulos, and D. Soudris, "Hardware acceleration of image registration algorithm on fpga-based systems on chip," in *Proceedings of the International Conference on Omni-Layer Intelligent Systems*, 2019, pp. 92–97.
[20] R. Shams, P. Sadeghi, R. Kennedy, and R. Hartley, "Parallel computation of mutual information on the gpu with application to real-time registration of 3d medical images," *Computer methods and programs in biomedicine*, vol. 99, no. 2, pp. 133–146, 2010.
[21] K. Ikeda, F. Ino, and K. Hagihara, "Efficient acceleration of mutual information computation for nonrigid registration using cuda," *IEEE Journal of Biomedical and Health Informatics*, vol. 18, no. 3, pp. 956–968, 2014.
[22] L. Gong, C. Zhang, L. Duan, X. Du, H. Liu, X. Chen, and J. Zheng, "Nonrigid image registration using spatially region-weighted correlation ratio and gpu-acceleration," *IEEE journal of biomedical and health informatics*, vol. 23, no. 2, pp. 766–778, 2018.
[23] P. Bhosale, M. Staring, Z. Al-Ars, and F. F. Berendsen, "Gpu-based stochastic-gradient optimization for non-rigid medical image registration in time-critical applications," in *Medical Imaging 2018: Image Processing*, vol. 10574. International Society for Optics and Photonics, 2018, p. 105740R.
[24] M. Brunn, N. Himthani, G. Biros, M. Mehl, and A. Mang, "Fast gpu 3d diffeomorphic image registration," *Journal of Parallel and Distributed Computing*, vol. 149, pp. 149–162, 2021.
[25] F. F. Berendsen, K. Marstal, S. Klein, and M. Staring, "The design of superelastix–a unifying framework for a wide range of image registration methodologies," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2016, pp. 58–66.
[26] A. Mang, A. Gholami, C. Davatzikos, and G. Biros, "CLAIRE: A distributed-memory solver for constrained large deformation diffeomorphic image registration," *SIAM Journal on Scientific Computing*, vol. 41, no. 5, pp. C548–C584, 2019.
[27] A. Chakraborty and A. Banerjee, "A novel vlsi architecture of cordic based image registration," in *2020 Sixth International Conference on Bio Signals, Images, and Instrumentation (ICBSII)*. IEEE, 2020, pp. 1–6.
[28] M. J. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *The computer journal*, vol. 7, no. 2, pp. 155–162, 1964.
[29] J. Bernon, V. Boudousq, J. Rohmer, M. Fourcade, M. Zanca, M. Rossi, and D. Mariano-Goulart, "A comparative study of powell's and downhill simplex algorithms for a fast multimodal surface matching in brain imaging," *Computerized medical imaging and graphics*, vol. 25, no. 4, pp. 287–297, 2001.
[30] Xilinx Inc., "Vitis Vision Library," 2019. [Online]. Available: https://github.com/Xilinx/Vitis_Libraries/tree/master/vision
[31] K. Clark, B. Vendt, K. Smith, J. Freymann, J. Kirby, P. Koppel, S. Moore, S. Phillips, D. Maffitt, M. Pringle *et al.*, "The cancer imaging archive (tcia): maintaining and operating a public information repository," *Journal of digital imaging*, vol. 26, no. 6, pp. 1045–1057, 2013.
[32] National Cancer Institute Clinical Proteomic Tumor Analysis Consortium (CPTAC), "Radiology data from the clinical proteomic tumor analysis consortium lung adenocarcinoma [cptac-luad] collection [data set]," *The Cancer Imaging Archive*, 2018. [Online]. Available: https://doi.org/10.7937/k9/tcia.2018.pat12tbs
[33] R. Shams, P. Sadeghi, R. A. Kennedy, and R. I. Hartley, "A survey of medical image registration on multicore and the GPU," *IEEE Signal Processing Magazine*, vol. 27, no. 2, pp. 50–60, 2010.

Eleonora D'Arnese received her B.Sc. and M.Sc. in Biomedical Engineering from Politecnico di Milano in 2016 and 2018 respectively. She also received in 2018 M.Sc. degree in Bioengineering from the University of Illinois at Chicago, Chicago, IL, USA. She is currently a Ph.D. Student in Information Technology at Politecnico di Milano. Her research focuses on pipeline generation for medical image processing and machine learning.



Davide Conficconi got his Ph.D. in Information Technology at Politecnico di Milano in 2022. He received his B.Sc. and M.Sc. in Computer Engineering from Politecnico di Milano in 2015 and 2018 respectively. His research interesets revolves around reconfigurable architectures, especially FPGAs, design methodologies, computer architectures, and design automation techniques.

Emanuele Del Sozzo got his Ph.D. in Information Technology from Politecnico di Milano in 2019. He received his B.Sc. and M.Sc. in Computer Engineering from Politecnico di Milano in 2012 and 2015 respectively. He also receives in 2015 M.Sc. degree in Computer Science from the University of Illinois at Chicago (UIC), and Alta Scuola Politecnica Diploma. His research focuses on reconfigurable architectures, code generation and optimization. He is currently a PostDoc at Politecnico di Milano.



Luigi Fusco is a M.Sc. student in Computer Engineering at Politecnico di Milano and in the Alta Scuola Politecnica program. He received his B.Sc. in Computer engineering from Politecnico di Milano in 2020. His interests include mathematics and high performance computing.

Donatella Sciuto received her Laurea in Electronic Engineering from Politecnico di Milano and her PhD in Electrical and Computer Engineering from the University of Colorado, Boulder, and an MBA from Bocconi University. She is currently the Executive Vice Rector of the Politecnico di Milano and Full Professor in Computer Science and Engineering. Her main research interests cover the methodologies for the design of embedded systems and multicore systems. She has published over 300 scientific papers. She is a Fellow of IEEE and has served as President of the IEEE Council of Electronic Design Automation from 2011 to 2013 and in different capacities in IEEE Committees and conferences.

Marco Domenico Santambrogio (SM'05) received the Laurea (M.Sc. equivalent) degree in computer engineering from the Politecnico di Milano, Milan, Italy, in 2004, the M.Sc. degree in computer science from The University of Illinois at Chicago, Chicago, IL, USA, in 2005, and the Ph.D. degree in computer engineering from the Politecnico di Milano, in 2008. He was a Post-Doctoral Fellow with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA. He has been with the NECST Laboratory, Politecnico di Milano, where he founded the Dynamic Reconfigurability in Embedded System Design project in 2004 and the CHANGE (self-adaptive computing system) project in 2010. He is an Associate Professor with the Politecnico di Milano. His current research interests include reconfigurable computing, self-aware and autonomic systems, hardware/software co-design, embedded systems, and high-performance processors and systems.