Hephaestus: Codesigning and Automating 3D Image Registration on Reconfigurable Architectures

GIUSEPPE SORRENTINO, MARCO VENERE, DAVIDE CONFICCONI, ELEONORA D'ARNESE, and MARCO DOMENICO SANTAMBROGIO,
Politecnico di Milano, Italy

Healthcare is a pivotal research field, and medical imaging is crucial in many applications. Therefore finding new architectural and algorithmic solutions would benefit highly repetitive image processing procedures. One of the most complex tasks in this sense is image registration, which finds the optimal geometric alignment among 3D image stacks and is widely employed in healthcare and robotics. Given the high computational demand of such a procedure, hardware accelerators are promising real-time and energy-efficient solutions, but they are complex to design and integrate within software pipelines. Therefore, this work presents an automation framework called Hephaestus that generates efficient 3D image registration pipelines combined with reconfigurable accelerators. Moreover, to alleviate the burden from the software, we codesign software-programmable accelerators that can adapt at run-time to the image volume dimensions. Hephaestus features a cross-platform abstraction layer that enables transparently high-performance and embedded systems deployment. However, given the computational complexity of 3D image registration, the embedded devices become a relevant and complex setting being constrained in memory; thus, they require further attention and tailoring of the accelerators and registration application to reach satisfactory results. Therefore, with Hephaestus, we also propose an approximation mechanism that enables such devices to perform the 3D image registration and even achieve, in some cases, the accuracy of the high-performance ones. Overall, Hephaestus demonstrates 1.85× of maximum speedup, 2.35× of efficiency improvement with respect to the State of the Art, a maximum speedup of 2.51× and 2.76× efficiency improvements against our software, while attaining state-of-the-art accuracy on 3D registrations.

CCS Concepts: • **Hardware** → **Hardware-software codesign**; *Hardware accelerators*; • **Computer systems organization** → **Reconfigurable computing**; • **Applied computing** → Imaging;

Additional Key Words and Phrases: 3D image registration, design automation, FPGA

ACM Reference format:
Giuseppe Sorrentino, Marco Venere, Davide Conficconi, Eleonora D'Arnese, and Marco Domenico Santambrogio. 2023. Hephaestus: Codesigning and Automating 3D Image Registration on Reconfigurable Architectures. *ACM Trans. Embedd. Comput. Syst.* 22, 5s, Article 134 (September 2023), 24 pages.
https://doi.org/10.1145/3607928

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Hardware/Software Codesign and System Synthesis, (CODES+ISSS), 2023.

Authors' addresses: G. Sorrentino, M. Venere, D. Conficconi, E. D'Arnese, and M. D. Santambrogio, Politecnico di Milano, Milan, Italy; emails: {giuseppe.sorrentino, marco.venere}@mail.polimi.it, {davide.conficconi, eleonora.darnese, marco.santambrogio}@polimi.it.

1539-9087/2023/09-ART134 $15.00
https://doi.org/10.1145/3607928

1 INTRODUCTION AND MOTIVATION

Healthcare is a central research field that has seen a surge in the development of different computational approaches [14, 29]. Moreover, in the last few years, medical imaging has become pivotal for many applications, such as disease prevention, diagnosis, and robotic/computer-aided surgery [24]. Indeed, creating efficient, accurate, and fast solutions is gaining traction even among tech companies not directly related to the healthcare world since it presents attractive open research challenges and opportunities [5, 13, 15, 17].

Therefore, finding new architectural and algorithmic solutions would benefit various highly repetitive image processing procedures such as registration, segmentation, and classification. Thus, two main orthogonal paths can be followed: one on identifying ad-hoc algorithms and the other on identifying the most suitable hardware substrate for their execution. *Image registration* is an excellent candidate for developing efficient, accurate, and accelerated solutions within this context.

Indeed, image registration is a well-known pre-processing step that aligns one or more stacks of images to a reference one allowing for correct data integration and fusion [3, 24]. More specifically, it iteratively searches the space of geometric transformation parameters to find, based on a similarity metric, the optimal ones that allow for the correct overlap of the considered images [31].

In clinical practice, it is essential to achieve a comprehensive understating of the analyzed area, opening to the correct alignment of different types of images (e.g., anatomical and functional). It is also widely employed in real-time robotic surgery, where pre-operative images are fused with intra-operative ones acquired in real-time during the procedure, thanks to a C-arm (i.e., a portable imaging device) [24].

In such a scenario, registering images in the lowest time and keeping energy consumption to the minimum, giving for granted the best possible accuracy, is essential to reduce its impact on surgery time. Additionally, from the State of the Art, it is clear how the computation of the similarity metric, which indicates how well the images to be registered align, is the most compute-intensive part of image registration [7], especially considering one of the literature's most employed and robust one such as the Mutual Information (MI) [3, 31].

Therefore, software- and hardware-based solutions are to be considered to allow for a flexible, fast, accurate, and efficient registration. The former accounts for solutions like Matlab [19] and SimpleITK [18], which allows for high flexibility in creating custom image registration pipelines. Indeed, they offer different algorithms for the similarity metric and optimization procedure, where the end users can customize some parameters, like the number of iterations, paying on the reached performance. On the contrary, on the hardware side, the literature highly reduces the available flexibility by devising custom accelerators tailored to a specific volume size, or registration configuration, to reach remarkable performance, preventing customization on the end users side [4, 8, 28]. Thus, to mix the benefits of both approaches, namely the software flexibility and the performance achievable through hardware acceleration, this paper, building upon our previous open-source solution [7, 9], presents a 3D-based framework for creating and developing software programmable FPGA-based accelerators for the MI similarity metric integrated into affine image registration pipelines. Indeed, the software programmability easily adapts to different volume dimensions and, thanks to the proposed hardware abstraction layer, transparently allows the end users to benefit from hardware acceleration without needing specific knowledge in hardware development. In summary, to bridge the current literature gap of 3D image registration, we propose an open-source framework[1] that, leveraging codesign strategies and an automation and abstraction layer, allows

[1]https://github.com/necst/hephaestus

for the deployment of fast and efficient 3D image registration pipelines benefiting from MI acceleration.

The main contributions of this work are:

- A software/hardware **codesign methodology** for highly **efficient** 3D image registration pipelines where all the components are **optimized** at both levels (Section 3).
- An **optimized**, **customizable**, and **reconfigurable architecture** for the MI computation designed to flexibly handle a **run-time variable** number of images to span from the 2D to the 3D registration case (Section 3.1.2 and 3.1.3).
- An **extensible automation framework** that, given the input design parameters and device, automatically generates the design towards the bitstream combined with a software layer that **abstracts the underlining hardware**, either embedded or datacenter devices (Section 3.2).
- Two device-class-specific codesign strategies: for embedded systems, where we design a mechanism to cope with main memory constraints, and for high-performance systems, where we scale out the accelerator and further parallelize the software to benefit from the hardware parallelization (Section 3.3).

We evaluate the proposed solution in terms of accuracy, energy efficiency, and execution time. Our experimental evaluation studies at first the scalability of the single accelerator under different aspects (Section 4.1). Then, we compare the whole application execution time and accuracy against a state-of-the-art software demonstrating top speedup of 1.85×, while improving our software of 2.51× (Section 4.2), and achieving state-of-the-art accuracy across all the platforms (Section 4.3). We explore the possible boosts that our embedded-specific mechanism could enable in the high-performance scenario, paving the way to novel strategies (Section 4.4). Finally, we compare Hephaestus with the literature showcasing top energy efficiency improvement of 2.35× against a state-of-the-art software while 2.76× improvement with respect to our software (Section 4.5).

2 BACKGROUND AND RELATED WORK

As stated, image registration is pivotal in many fields, and different approaches have been developed over the years. They can be clustered into two main categories: rigid/affine (preserves line parallelism) and deformable (loss of line parallelism) [31]. Since the scope of this work is the former class of registration approaches, and, given that it is also required as the first step in the deformable case, we provide a short introduction on relevant theoretical concepts and a description of the works in the literature on this topic.

Affine image registration can be carried out in various ways, among which the most employed ones rely on landmark points or voxels intensity. While the former requires some a-priori knowledge of some image features, the latter does not and, therefore, is more general and commonly employed [3]. Based on these considerations, we decided to target the more general voxel intensity-based solutions that can be applied even when no direct correspondence can be found among image volumes, as it occurs in the medical field. In this configuration, the registration is framed as an iterative heuristic process that aims at maximizing a similarity metric (*Sim*) among the reference (R) and floating (F) stacks, as in Equation (1).

$$\hat{T} = \underset{T}{\operatorname{argmax}}\, Sim(R, F^T) \tag{1}$$

More into detail, the procedure starts by estimating the initial geometric transformation parameters (T_0) as in Figure 1. Their estimation is commonly based on the image moments, which are a well-established approach in 2D-based image registration [12]. Indeed, for the 3D affine case,

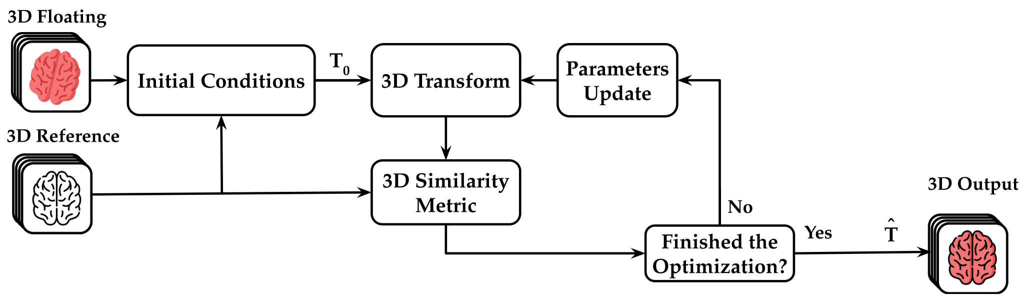


Fig. 1. High-level scheme of a generic 3D affine image registration pipeline. Starting from the input image volumes, it estimates the initial transformation parameters and, thanks to an optimization procedure guided by a similarity metric value, refines them until the best ones are found.

the transformation parameters, which are three translation factors t_x, t_y, and t_z, and three rotation angles θ, ϕ, and ψ, can be as well derived through the image moments. Unfortunately, in the 3D scenario, these moments are more complex to compute, given the lack of an established and shared definition of their computation. Therefore, the literature proposes different computational proxies [12] for the initial transformation matrix parameters, which constitute the transformation matrix in Equation (2).

$$T = \begin{bmatrix} \cos\theta\cos\psi & -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi & t_x \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi & t_y \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta & t_z \end{bmatrix} \quad (2)$$

Such parameters are then refined, through a heuristic procedure, based on an optimizer, which aims, as in Equation (1), at maximizing the value of a similarity metric computed among the R volume and the F one transformed with the current matrix T. Finally, the optimization converges when the optimal transformation parameters ($\hat{T}$) are found and used to obtain the registered floating volume $F^{\hat{T}}$. Additionally, for each of the three blocks, namely the transformation, the optimizer, and the similarity metric, different algorithms can be considered to create a wide range of configurations adaptable to a specific use case.

Considering the approach described above, the literature presents different solutions on both the software and hardware sides. In the software scenario, MATLAB [19] is a closed-source solution offering high-performing configurations but with minimal flexibility. Also, on the software side, SimpleITK [18], a Python-based library, offers a wide range of configurations, highly customizable by the end user but at the cost of delivered performance. Conversely, on the hardware side, different efforts have been put into accelerating the similarity metric computation and/or the transformation being the most compute-intensive parts [2, 7, 9, 10, 16, 25]. However, to reach the best performance in terms of execution time, the majority of the solutions in the literature present only one specific configuration, which is defined at design time and not adaptable unless the entire hardware design cycle is repeated, limiting the applicative flexibility to real scenarios [4, 8, 28]. Among them, we have previously proposed a first attempt towards customization with an FPGA-based acceleration of the Mutual Information (MI) similarity metric for 2D image registration [7, 9]. Although we propose customizable accelerators, they only deal with a couple of 2D images at a time, which is a limiting factor for the medical field, where, 3D image volumes, even though are acquired as set of 2D images, are then stacked into a 3D volume allowing for the creation of continuous structures. Indeed, working on 2D images, which means applying a different transformation matrix to each considered image, prevents the retention of the information along the image acquisition axes, possibly creating unreal discontinuities in the original overall volume.

3 HEPHAESTUS CODESIGN AND AUTOMATION METHODOLOGY

This Section details the proposed methodology, starting from the codesign of fast, efficient, and software-programmable MI accelerators (Section 3.1). These codesigned components are the foundation of Hephaestus framework, which features automation and abstraction layers to integrate them within 3D affine image registration pipelines seamlessly (Section 3.2). Finally, the Section ends by describing the tailoring of the proposed solutions to the embedded and the high-performance applicative cases (Section 3.3).

3.1 Hephaestus Codesign Strategy

Creating a flexible and fast solution for 3D image registration is pivotal to allow for the employment of performant hardware accelerators taking care of the most compute-intensive parts of the computation. At the same time, the possibility of matching the flexibility of software libraries, namely their possibility of customization at several levels of the registration pipeline (e.g., the number of slices composing the volumes to be registered), and their ease of usage requires an accurate study and a proper codesign of both the proposed accelerators and the overall registration software to achieve a tight coupling and best results ranging from accuracy, usability by a non-hardware expert end user, and performance. For all these reasons, we accurately studied and codesigned the proposed software-programmable MI accelerators to consider the constraint of the registration procedure and vice-versa.

3.1.1 Software Codesign: 3D Image Registration. Section 2 described a general framework for 3D image registration represented in Figure 1 where five components (the initial conditions, the 3D transform, the 3D similarity metric, the convergence condition, and the parameters updates) build the overall iterative heuristic process. The convergence and the update of the parameters are properties of the so-called optimizer component. We design two 3D-based optimizers to represent two broad classes of non-gradient-based approaches that explore the parameters space with different strategies. The designed optimizers are called Powell's method and 1+1 Evolutionary. The former optimizes each factor of the transformation matrix (see Equation (2)) at a time and converges based on a certain hyperparameter threshold, while the latter explores with an evolutionary strategy of parent-child all the parameters simultaneously, and, usually, it converges when reaching the maximum amount of iterations. These two optimizers fall at opposite sides of the time-accuracy trade-off, which is the primary image registration dilemma, thus, being highly representative.

Algorithm 1 describes the search algorithm employing Powell's optimizer. The algorithm works on the input reference and floating volumes, their slice number, the parameters of the transformation matrix, a vector of ranges (*rng*) within which explore the parameters, and a threshold. The algorithm computes a baseline MI value on the initial transformation parameters and starts exploring the search space in a loop for convergence. Each transformation parameter (i.e., translation and rotation along x, y, and z axes) invokes the *Gold_srch* procedure that represents the golden section search algorithm. This algorithm evaluates the best approximate transformation value for a given parameter in a range. Specifically, the algorithm evaluates the golden section interval for the given parameter and saves its extremes, *start* and *end*. Then, it simultaneously explores two possible transformations through c and d by computing their MI value. The procedure iteratively executes and resizes the interval until the distance between c and d is lower than a hyperparameter called ϵ.

After the golden section search procedure, the Powell's method receives an optimized parameter for the given range and its current MI. Suppose the current MI and the previous (or the initial)

ALGORITHM 1: Powell's method search for the best parameters

```
procedure POWELL ALGORITHM($ref\_volume$, $flt\_volume$, $volume\_size$, $params$, $rng$, $thr_{powell}$)
    while $Not\ converged$ do
        for i in range(len(params)) do
            $cur\_rng, cur\_param = get\_cur(params[i], rng[i])$
            $param\_opt, cur\_mi = Gold\_srch(cur\_param, cur\_rng, ref\_vol, float\_vol, params, i)$
            if $cur\_mi - last\_mi > thr_{powell}$ then
                $params[i] = param\_opt$
                $last\_mi = cur\_mi$
                $converged = False$
            else
                $params[i] = cur\_par$
                $converged = True$
    return params
procedure GOLD_SRCH($cur\_param, cur\_rng, ref\_volume, flt\_volume, volume\_size, params, idx$)
    $best\_mi = 0$
    $(start, end, c, d) \leftarrow initializeParams()$        ▷ Initialize search parameters with golden section
    while abs(c-d)>$\epsilon$ do
        $matrix1 \leftarrow update(params, c)$        ▷ First transformation matrix to apply
        $matrix2 \leftarrow update(params, d)$        ▷ Second transformation matrix to apply
        $MI\_matrix1 = computeMI(ref\_vol, flt\_vol, volume\_size, matrix1)$        ▷ First MI
        $MI\_matrix2 = computeMI(ref\_vol, flt\_vol, volume\_size, matrix2)$        ▷ Second MI
        if $MI\_matrix1 > MI\_matrix2$ then        ▷ Focus on first portion of interval
            $end = d$
            $best\_mi = MI\_matrix1$
        else        ▷ Focus on the second portion of the interval
            $end = c$
            $best\_mi = MI\_matrix2$
        $(c, d) \leftarrow updateParams(c, d)$        ▷ Update transformation values
    return $mean(end, start), best\_mi$        ▷ Return the best transformation value and the MI
```

one differ from a certain user-defined threshold. In that case, the current MI is saved into *last_mi*, and the corresponding transformation parameter is kept in the transformation matrix. After considering transformations for all the search parameters, the search restarts from the first parameter. The convergence is reached when none of the search parameters is modified anymore.

Conversely, Algorithm 2 illustrates our pseudocode of the 1+1 Evolutionary optimizer. Apart from the input volumes and their features, the other 1+1 hyperparameters define the searching strategy within the parameter space, e.g., the maximum number of iterations and the radius for parameter exploration. The algorithm starts by initializing specific data structures and estimating the initial transformation matrix for the search. Then, it extracts the translation and rotation around x, y, and z axes from such a matrix and assigns them to the starting *parent* node of the search tree. Once the new transformation matrix is applied to the floating volume, it computes the MI of the reference and floating volumes. After this setup stage, it begins a loop until the maximum iteration number is reached. In this loop, starting from the selected parent node, noise is added to the parameters in *parent*, thus generating a child node in which the *child* label contains the new parameters. Then, the original floating volume is transformed using the child parameters, and the MI between this latter volume and the reference is computed. The child node is discarded if the

ALGORITHM 2: 1+1 Evolutionary search for the best parameters

```
procedure 1+1 Evolutionary Algorithm($ref\_volume$, $flt\_volume$, $volume\_size$)
    parameters
        $MaximumIters$: maximum number of iterations to perform
        $SpaceDim$: number of parameters to explore (dimension of the search space)
        $InitRadius$: radius for parameters exploration
        $GrowthFactor$: parameter for expansion of the explored subspace
        $\epsilon$: threshold for evaluating Frobenius Norm of A
    end parameters
    $FrobeniusNorm = 0.0$
    $ShrinkFactor = GrowthFactor^{-0.25}$
    $(A, child, delta, f\_norm) \leftarrow initializeData(initRadius)$
    $parent \leftarrow estimateInitial(ref\_volume, flt\_volume, volume\_size)$
    $P\_value = computeMI(ref\_volume, flt\_volume, volume\_size, parent)$
    for  i in range(MaximumIteration) do
        $f\_norm \leftarrow normalVariateGenerator()$                ▷ Generate random updates
        $delta = A \cdot f\_norm$
        $child = parent + delta$                ▷ Generate child node by applying updates
        $C\_value = computeMI(ref\_volume, flt\_volume, volume\_size, child)$
        $adjust = ShrinkFactor$
        if $C\_value > P\_value$ then                ▷ Select best node for search
            $P\_value = C\_value$
            $child = parent$
            $adjust = GrowthFactor$
        $FrobeniusNorm = frobeniusNorm(A)$
        if $FrobeniusNorm < \epsilon$ then break
        $A \leftarrow update(adjust, f\_norm, delta)$
    return parent
```

obtained MI is lower than the MI value computed at the previous iteration, and a new child will be generated at the next iteration. Otherwise, the considered child node is promoted to *parent*, and the next iteration will generate a child node starting from it. Notably, the matrix A is transformed at each iteration to modify the explored subspace's dimension dynamically. The procedure ends if the maximum number of iterations is reached or if the Frobenius norm of the matrix A is lower than a given threshold.

Although the optimizer's searching strategy and convergence are essential, a bad initialization of the starting conditions has already been demonstrated to affect many optimization problems negatively [20]. Therefore, computing the 3D moments of the stack of images would represent a reasonable estimate of the required initial conditions for a transformation matrix related to translations and rotations around x, y, and z axes. However, the literature demonstrated that dealing with 3D image moments corresponds to highly complex computations, impacting the final procedure performance dramatically [12]. Therefore, given that the literature lacks a unique computational proxy of 3D moments as stated in Section 2, within Hephaestus, we propose an approximate solution with a lightweight computational load and a geometrical interpretation.

More specifically, we split the two input volumes (i.e., the reference and the floating) into a sequence of 2D slices along the z axis that matches the acquisition direction for the considered case. Then, for each pair of 2D images, we compute their 2D moments to estimate the initial translation parameters along x and y axes and rotation angle around z axis. Afterward, we estimate the rotation angle around y and x with the same computation by rotating the original volume of 90 degrees counterclockwise around x and clockwise around y, respectively. Although this formulation is the most general one allowing for the estimation of all the six degrees of freedom for a 3D affine transformation, it can be simplified based on the final applicative scenario. Indeed, having a prior understanding of the settings and constraints present during the image acquisition process can facilitate the initialization of specific transformation parameters. For example, in the considered applicative medical scenario, the employed images present the human body lying over a rigid surface. Such configuration implies that the problem geometry prevents some rotations and translations, thus estimating that parameters may be redundant and excessively time-consuming. Indeed, in our specific case, rotation around x and y axes, as well as translation along z axis, tend to be negligible. We, therefore, devised an additional procedure to compute the initial moments, which only takes into account translations along x and y and rotation around z while initializing to zero the remaining parameters. Such a procedure requires reduced execution time without causing a noticeable accuracy loss. Moreover, the search algorithm still assures the exploration and optimization of all six degrees of freedom without limitation, preventing the loss of generality.

The only remaining components of a general 3D image registration framework are the 3D transformation and the 3D similarity metric. Though 3D image transformations are widely employed in several image-processing pipelines, we struggle to find a software implementation available as a golden reference. Kornia library [23] provides a highly optimized software implementation of the 3D affine transformation that fits our scenario perfectly, being able to transparently use CPU or GPU devices and providing batch processing capabilities. The 3D similarity metrics that fit the optimizer computing requirements are several in the State of the Art [3, 26]. Within this context, the Mutual Information (MI) demonstrated to be one of the most promising similarity metrics for multi-modal intensity-based registration procedures [3, 7, 25, 26, 31]. However, the literature also demonstrated that the computational complexity of the MI combined with the huge number of its evaluations represent a clear bottleneck in the iterative registration procedures [7, 16, 25]. Therefore, the following Section will focus on the codesign of the hardware accelerator that Hephaestus exploits to optimize the 3D image registration application.

3.1.2 *Hardware Codesign: Mutual Information Accelerator.*

We based the current 3D accelerator design based on our previous open-source effort [7], where we developed a highly customizable MI accelerator for 2D image registration. Also in this case, we leverage the High-Level Synthesis (HLS) as coding style, since it eases the circuit generation for different use cases based on some customization parameters. We first devise a design that respects the MI computational structure, which is based on the following definition:

$$MI(X,Y) = H(X) + H(Y) - H(X,Y) \tag{3}$$

where $H(X)$, $H(Y)$, and $H(X,Y)$ are the entropies of the reference volume, the floating volume, and their cross-entropy, respectively, and defined as follows:

$$H(X) = -\sum_{x \in X} P(x) \log P(x) \tag{4}$$

$$H(X,Y) = -\sum_{x,y \in X,Y} P(x,y) \log P(x,y) \tag{5}$$

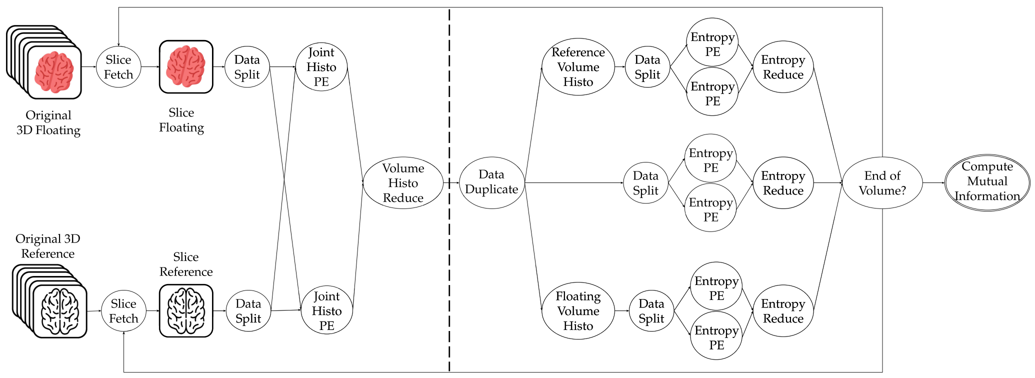


Fig. 2. 3D Mutual information dataflow architecture: the accelerator fetches the slices, computes the joint histogram in the first macrostage, the entropies in the second macrostage, and once processed the whole volume writes the resulting mutual information value.

Therefore, to compute the MI value, it is necessary to evaluate the entropies ($H(X)$, $H(Y)$) of the two images and their cross-entropy ($H(X, Y)$). These values, based on probabilities $P(x)$, $P(y)$, and $P(x, y)$, can be evaluated by computing the histograms of the 2D images.

Following the mathematical definition of the MI which is the same for the 3D case, we substitute the X and Y 2D images with two 3D image volumes (acquired in the medical domain as a sequence of N 2D images acquired along the z-axis). Therefore, the proposed accelerator within Hephaestus, mimicking the mathematical definition of the MI, splits the computation into two macrostages (divided by the dashed line in Figure 2): the first stage estimates the histograms of the images, while the second retrieves the entropy based on the content of such histograms. Though the two macrostages are blocking one with respect to the other, we adopt the dataflow design style to fetch, consume, and compute as fast as possible, and we internally pipeline each macrostage through different intermediate stages, with a map-reduce approach.

Starting from the first macrostage, i.e., the histograms computation, it is convenient to compute the joint histogram as the first step, since the two single histograms can be extracted from it. The volumes' joint histogram can be seen as a squared matrix of $L \times L$, where L is the number of gray levels in the 3D volumes (e.g., 256 levels in our case). Additionally, the value of each element of the joint histogram $hist(x, y)$ is equal to the total number of voxels of volume X with intensity x corresponding to the voxels of volume Y with intensity y. Given its definition, after evaluating two volume slices, the processing elements (PEs) must keep the partial joint histogram until all the slices are evaluated. Therefore, we needed to modify the histogram processing elements (HPEs) that we previously proposed [7] to benefit from data parallelism in the 3D context.

These PEs are designed to cache the partial results for the overall volume of the joint histogram computation to preserve the memory up to the end of the volume's slices. The derivation of the marginal histograms, which delineates the starting point of the second macrostage, follows the joint one, to enable the computation of the reference, floating, and joint entropies required for the final MI. Once the joint histogram is correctly computed, the two single histograms, the one of the reference and the one of the floating volumes can be extracted by a row and column reduction respectively. Then, the proposed accelerator, starting from the histograms, computes the entropies needed for the computation of the MI value.

Even this second macrostage features the map-reduce paradigm for $H(X)$, $H(Y)$, and $H(X, Y)$ computations with several independent PEs. These map-reduce adoptions enable a higher degree of parallelism, requiring fewer clock cycles at the cost of more memory or arithmetic resources for the joint histogram and entropies macrostages. To further improve the performance and reduce as much as possible the number of floating point operations, we limit their usage for the sole entropy macrostage, by computing the entropies with the following approximation:

$$H(X) = -\frac{1}{R \cdot C \cdot D} \sum_{x \in X} J(x) \log J(x) + \log(R \cdot C \cdot D) \tag{6}$$

where $J(X)$ is the joint histogram, R is the number of rows, C is the number of columns of each slice, and D is the depth of the volume, i.e., the number of slices per volume. Finally, the term $\frac{1}{R \cdot C \cdot D}$ must be added to Equation (3) to complete the approximation. We refer to this term as the scale factor. Moreover, we analyze the impact of this approximation across several possible volume sizes experimentally, measuring a maximum error in the computed MI values in the order of 10^{-6}, which is negligible and, therefore, will be ignored in further discussions.

Overall, this pipeline iteratively processes the input slices of the given volumes to build incrementally the joint histogram of the volume pairs. Once consumed all the slices, the accelerator will complete the MI computation, write back this similarity metric value for the registration process, and reset its internal state.

3.1.3 Further Pushing the Hardware Codesign: Run-time Variable Volume Sizes. Till now the proposed software development and the developed MI accelerator are sufficient to perform a correct 3D image registration procedure. Although the development process has reached a working point, it is yet to be considered a flexible and transparently adoptable solution by end users. The current solution deals with a predefined number of images composing the volumes to be registered, while the applicative scenario, namely the medical field, requires for flexibility in the depth of the volumes. Indeed, the number of images composing the volumes is a parameter selected at acquisition time and is tailored to best represent the relevant characteristic from a clinical perspective. For these reasons, to be useful, the registration software requires more flexible hardware, which can easily adapt at run-time to volumes with dimensions varying in a given range of the number of slices. Such adaptability is essential to move from a fixed hardware accelerator integrated into a software framework to a flexible hardware component usable as a software object instantiable at run-time to deal with different input depths. Such degree of flexibility offered by Hephaestus represents a novel contribution to the State of the Art of 3D image registration and a step towards its integration in real applicative cases.

Given the above-highlighted necessity, to match them and make our MI accelerator software-programmable, we have modified the scale factor introduced in Equation (6) to adapt to a variable number of slices (D) composing the input volumes. Indeed, in the original definition of the scale factor, as $\frac{1}{R \cdot C \cdot D}$, D represented the fixed number of 2D slices composing the input 3D volumes supported by the MI accelerator. Now, to account for different possible volume depths, we add a further corrective factor, which takes into account the difference, in bits, between the maximum volume size supported by the accelerator, D_{max}, and the current size D of the input volumes. The Hephaestus scale factor is therefore redefined as:

$$\frac{1}{R \cdot C \cdot D} - (\log_2(D_{max}) - \log_2(D)) \tag{7}$$

where D_{max} is the maximum supported volume depth, while D is the effective depth of the given input volumes. Thanks to this further correction Hephaestus allows the creation of software-programmable MI accelerators adaptable to different use cases. Also in this case, we have investigated the effect of introducing an additional corrective factor over the computed MI value and seen that it is negligible, and, specifically, it tends to zero as the number of considered slices tends to the maximum supported volume depth D_{max}. Moreover, we modify the whole pipeline to be

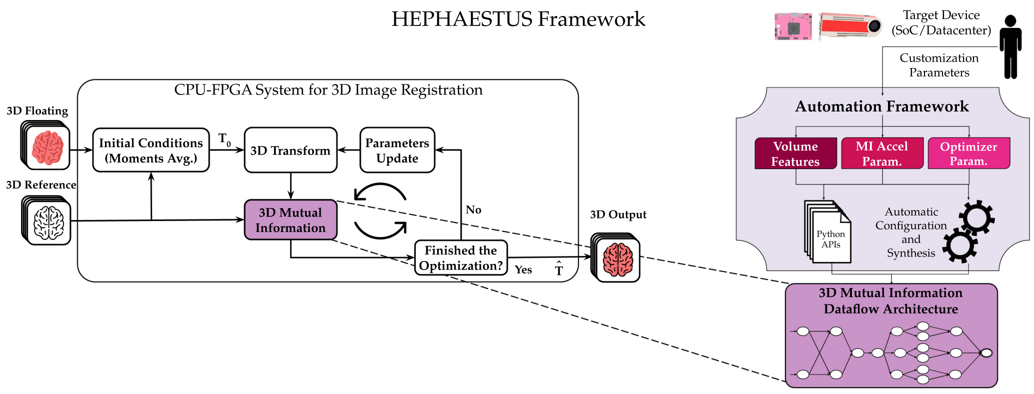


Fig. 3. Hephaestus framework: it is based on a codesigned SW/HW image registration pipeline for 3D volumes of images based on a run-time configurable mutual information accelerator. Hephaestus features an automation framework that, based on the target deployment system and user-based customization parameters, automatically build the overall SW/HW registration pipeline.

adaptable to the actual D at run-time. Indeed, D is provided as input to the accelerator and used to stop the processing of couples of slices once the D-th couple has been received. Hence, the entropy macrostage produces the correct MI value at the end of the D slices, reducing the latency at the necessary clock cycles, while increasing the flexibility.

At this point, within Hephaestus we have codesigned fast, efficient, and software-programmable MI accelerators to be integrated into 3D image registration pipelines. Such achievement allows for the creation of performant hardware accelerated solutions while keeping the flexibility of pure software implementations. To further ease the usage of the proposed accelerators Hephaestus proposes an automation flow and an abstraction layer, described in detail in the following Section. In this way, we can integrate them transparently within 3D affine image registration pipelines.

3.2 Hephaestus Automation and Abstraction Framework

Given the steepness of the learning curve of FPGAs technology in both development and usage [11, 27, 30], we designed an automation framework and abstraction layer to ease the final FPGA-based 3D image registration pipelines development. Figure 3 illustrates what Hephaestus offers to the final user: an automated accelerator deployment and application design that exploits hardware accelerators transparently.

The automation layer is essential to configure the highly customizable 3D MI accelerator in different degrees. Indeed, it provides a collection of parameters, shown in Table 1, that enable developers to customize our framework at several levels of abstraction. In particular, at the hardware design level, it allows tuning the number of bits used to represent pixels of the input image (*IB*), the image dimensions (*ID*), the macrostages parallelism degrees (i.e., the PEs number for the histogram (*HPE*) and the entropy (*EPE*)), the number of cores (*CORE_NR*), and the maximum volume depth (D_{MAX}) that our accelerator can handle. Moreover, given this maximum value, we tailor the scale factor, the arithmetic operations, and the bitwidth of different structures, such as the joint histogram PEs bitwidth. At the software level, developers can set specific parameters for the optimizers, as for software libraries, to tune their execution time-accuracy tradeoff, such as the maximum number of iterations to perform in 1+1 Evolutionary or the accuracy threshold in Powell's method. Finally, the automation and hardware abstraction layer can integrate additional

Table 1. Customization Parameters of Hephaestus Accelerator Exposed Through the Automation

Parameter	Description
IB	Number of bits for the target input image
HPE	Number of parallel histogram PEs
EPE	Number of parallel entropy PEs
D_{MAX}	Maximum number of 2D slices composing the 3D volume
ID	Dimension R and C of the 2D slice
$CORE\_NR$	Number of parallel independent cores

optimizers or target boards, further extending the degree of flexibility provided to the end user of the framework.

Given these customization parameters, Hephaestus targets a wide number of FPGA-based systems that we cluster in the embedded and high-performance devices, by producing a specialized Register Transfer Level (RTL) description of the final architecture. These two device classes imply two different design flows: one for the embedded and one for the high-performance. The former, i.e., the embedded design flow, performs the High-Level Synthesis (HLS) process (i.e., from C++ to RTL generation) to create the IP, generate the system-level connections, and finally all the steps towards the bitstream generation. The latter design flow, the one for high-performance devices, performs analogous yet different steps with a specialized toolchain: it performs the steps of IP creation and packing, system devise, and bitstream creation. Moreover, since each device can present differences in physical parts or memory subsystems, we design specific automation components that are not exposed to the end user.

Although the steps for bitstream generation are device-specific, we design a modular automation infrastructure that macroscopically differs for the system class but exposes the very same interface to the final user for bistreams production. Therefore, users simply input the customization parameters and the device type, as in Figure 3, and Hephaestus takes care of all the necessary steps for bitstream generation. Additionally, designers can extend Hephaestus with new devices by adding the device part, and, if needed, custom automation parts, as long as they respect the current structure. In summary, Hephaestus can generate run-time software adaptable accelerators that can seamlessly work with 3D volumes or single 2D images in an automated way.

The system kind (i.e., embedded or high-performance) introduces both differences in the building process and in the software APIs employed to manage the host-accelerator overall communication. Therefore, since the software management for reconfigurable devices is strictly dependent on a given board, we designed a software component to relieve this burden from the application designer. Indeed, Hephaestus offers a Python-based software layer that abstracts the underlying low-level device features, such as buffer and execution management and exposes a unique interface for the high-level image registration application regardless of the target hardware.

Having the automation and the abstraction layers at disposal, Hephaestus can generate 3D image registration pipelines based on two different classes of optimizers (Section 3.1.1), on the MI accelerator (Section 3.1.2) as similarity metric that can accept run-time variable volume sizes (Section 3.1.3) in a seamless and integrated way regardless the target system.

3.3 Hephaestus Deployment Tailoring to Device Class

Since Hephaestus targets both embedded and high-performance devices, some further improvements can be reached by keeping in mind the strengths and limitations of each class of devices. For example, embedded devices are energy efficient but show constrained memory compared to high-performance cards. Therefore, in some cases, they do not show enough memory to carry out the entire registration procedure on the whole 3D volume since both the similarity metric and the registration software run on the memory-constrained device. Conversely, high-performance cards are commonly connected to high-end hosts allowing for higher performance even on bigger 3D input volumes.

Given that high-performance devices present more resources, we decide to explore the Hephaestus scalability on such devices. On the other hand, the embedded world is highly attractive for clinical practice, but memory limitations are a de-facto bottleneck for 3D image registration, and the current State of the Art has not yet solved this physical constraint. Therefore, to tackle such a limitation, we propose a novel mechanism allowing for 3D image registration on very memory-constrained devices while guaranteeing accuracy. Based on these considerations in the following Sections, we describe the tailoring made to match the characteristics of the two device classes considered, embedded and high-performance.

3.3.1 Hephaestus Tailoring to Embedded Devices. Given the traction of embedded solutions for developing clinical practice in constrained scenarios such as at-home exams, the ability to execute high-computational demanding procedures on small and memory-constrained devices is essential but not fully resolved. Indeed, as we have already introduced, 3D image registration requires both computational capabilities and increasing memory as the volume size increases. Therefore, when the registration is run on embedded devices, which integrate both a Processing System (PS) and a Programmable Logic (PL) with limited memory, the entire execution of the whole registration procedure may be impossible, specifically for high-resolution input volumes.

Given these considerations, we devise a new mechanism to overcome the aforementioned limitation and open 3D image registration to run on memory-constrained devices. More in detail, we tailor the searching algorithm to focus on a sampled subvolume of the original 3D input image stacks. Indeed, given that a 3D volume represents a continuous subject, it is reasonable to assert that all the slices are highly correlated with one another. Therefore, evaluating a portion of them is enough to find satisfactory translation and rotation parameters for registering the floating volume. Figure 4 gives an example of the process proposed to deal with input subvolumes; indeed, thanks to the previously introduced abstraction layer, for the end user, nothing changes, as Hephaestus takes care of the sampling and the correct deployment of the MI accelerator and the tailoring of the registration process to the selected subvolume size. To best exploit all the available resources on the target device, Hephaestus identifies the available memory and, according to such information, defines the size of the volume to be sampled, as shown in Figure 4. Additionally, Hephaestus uniformly samples the images starting from the central slice in the 3D input volume, given that we model the information within a 3D volume as a normal distribution. In this way, we aim to consider the subvolume retaining the most information to identify the best transformation among the reference and the floating subvolumes. After the sampling, the 3D image registration procedure works as previously introduced (Section 3.1) and highlighted in Figure 4 till convergence is reached. Finally, given the optimal 3D transformation parameters, the transformation is applied to the floating volume. The transformation can be applied directly to the whole 3D floating volume if there is enough memory, or if necessary, Hephaestus can sample subsequent floating subvolumes and separately transform them till the whole volume is registered. Also in this configuration, Hephaestus exploits its abstraction layer to define the optimal dimension for the subvolume to transform. A final tailoring of Hephaestus is that it can consider either disjoint or overlapping subvolumes to apply the transformation to maximize the final registration accuracy while penalizing the execution time. Thanks to the proposed tailoring solution, Hephaestus also allows for 3D affine image registration on embedded devices, and later on, we will analyze the effect of such tailoring on Hephaestus performance.

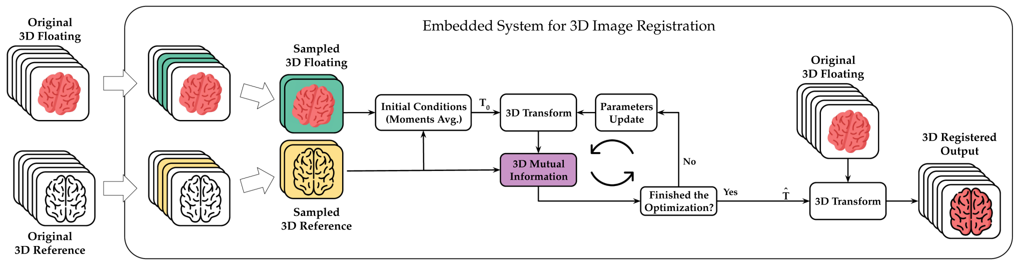


Fig. 4. Hephaestus workflow for embedded devices: the memory constraint imposes sampling for large volumes on which we then apply the registration procedure.

3.3.2 Hephaestus Tailoring to High-Performance Devices. On the opposite side of embedded systems, the high-end FPGA-based ones feature many more resources on the FPGA fabric and decoupled host-FPGA memory banks with a greater capacity (more than 8GB). Therefore, we design new class-specific optimizations taking advantage of these features. Firstly, more decoupled memory banks directly mean more parallel off-chip memory access, for instance, one bank for the reference and one for the floating volume. In this way, we reduce the contention on the off-chip memory banks, improving the overall bandwidth, and decreasing the MI computation latency. However, this bank's parallelization has a minimal impact on the reconfigurable fabric usage; thus, we can further push on the accelerator design.

The resource budget available on high-performance devices opens to two paths: scaling-up or scaling-out the MI accelerator. Thanks to the automation part of Hephaestus, the scaling-up of the accelerator just requires a simple change in the customization parameters. Section 4.1 will detail the results reachable by following the scaling-up path. On the other hand, scaling-out to multiple cores of the accelerators has an impact on the throughput and the application latency if the workload can take effective advantage of this choice. The 1+1 Evolutionary optimizer evaluates the MI one time per iteration and thus does not feature suitable characteristics. Instead, the Powell's method internally explores the search space of the parameters through the golden section search that evaluates two different sets of transformation parameters in parallel and repeatedly calls the MI to evaluate their goodness. Therefore, we codesign a solution that can take advantage of such parallelism by replicating the MI accelerator and creating an architecture of two independent MI cores. We also adapt Hephaestus automation to automatically generate the bitstreams for such configuration adding a new customization parameter. Moreover, we codesign Powell's-based registration pipeline to be able to exploit such Dual Core design effectively. Now, the software application sets up the execution for two accelerators in parallel and executes them in an asynchronous parallel way. Finally, since the 3D transformation block implemented through Kornia [23] can exploit batch processing by design, the Powell's-based pipeline can now seamlessly exploits the batch processing of the 3D software transformation and the parallelization of the MI hardware accelerator on the high-performance device class.

4 EXPERIMENTAL SETUP AND EVALUATION

In this Section, we describe the experimental setup employed for the implementation and validation of Hephaestus. Starting from a brief description of the experimental setup and the employed dataset, we then provide a systematic evaluation starting from analyzing the sole MI accelerator scalability, the whole 3D image registration performance in terms of execution time and accuracy, and a discussion over the achievable improvements when exploiting the embedded tailoring also on high-performance devices. Finally, we present a comparison with related work.

Table 2. Target Machine Configurations Reported with their Unique Code Name

Code Name	CPU	Machine RAM	Board DDR RAM	FPGA
ZCU	ARM A53	2GB	Shared	XCZU7EV
PZU	ARM A53	4GB	Shared	XCZU5EG
AU200	Intel i7-4760	16GB	64GB	Alveo U200
AU280	AMD Ryzen 7 3700X	32GB	32GB	Alveo U280

Within Hephaestus we developed the proposed MI reconfigurable accelerator in C++ exploiting Vivado HLS, Vivado, and Vitis suites version 2019.2. For what concerns the Hephaestus automation framework we mixed Python, TCL, bash, and Makefile utilities, while for the creation of the proposed abstraction layer we leverage the PYNQ framework v2.7 [1]. Additionally, we design the Hephaestus 3D image registration software in Python leveraging Kornia [23] and PyTorch [22] libraries for the 3D transformation component of our final application. Moreover, we select as representatives of the high-performance class the AMD-Xilinx Alveo U200 and the AMD-Xilinx Alveo U280 cards, while for the embedded class we target the AMD-Xilinx ZCU104 and the AMD-Xilinx PYNQ ZU. For convenience, Table 2 reports the features of the employed machines in this evaluation. Finally, to evaluate the power consumption of the proposed solutions, different strategies have been considered depending on the board used. In particular, for the boards with on-board sensors, measurements have been taken exploiting the PYNQbased APIs for PMBus. When this was not possible, they have been taken employing a Voltcraft 4000 energy logger. As testing dataset for the 3D image registration pipeline we select a medical dataset [6, 21],[2] composed of $512 \times 512 \times 246$ Computed Tomography images and a corresponding number of Positron Emission Tomography ones, up-scaled from the original 128×128 to 512×512 pixels.

4.1 Hephaestus MI Accelerator Scalability Analysis

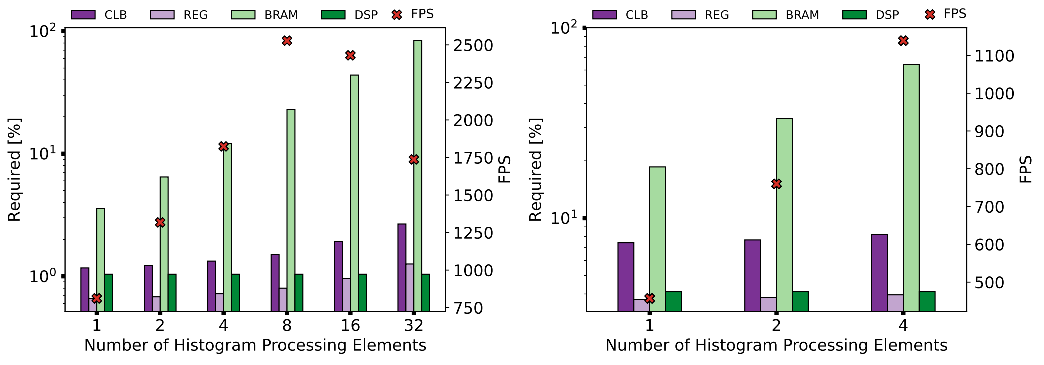
We have carried out a systematic analysis to evaluate the scalability of the MI accelerator, thanks to the proposed automation framework that allows for an easy generation of multiple MI accelerator configurations. We have evaluated the impact of the number of PEs employed for the two map-reduce stages of the accelerator, namely the histogram phase (HPE) and the entropy one (EPE), as in Figure 2, over the occupied hardware resources and the corresponding Frame per Second (FPS) rate reached by each accelerator. Specifically, the FPS rate is defined as:

$$FPS = \frac{Size_{Volume}}{ExeTime_{Avg}} \tag{8}$$

Given that, as already introduced, the histogram computation is more compute-intensive than the entropies computation, we start the analysis by exploring how varying the number of histogram PEs reflects on the FPGA resource usage and the reached FPS rate. Figure 5 reports the obtained results for the two classes of considered devices, and, in particular, for the Alveo U200 as representative of the high-performance class and for the ZCU 104 as the example of the embedded class. Specifically, we have synthesized accelerators with an increasing number of PEs for the histogram computation stage following the power-of-two pattern from one to maximum device occupation.

Figure 5(a) shows how increasing the number of HPEs from 1 to 32 for the Alveo U200 leads expectedly to a linear increase in the BRAM usage. Indeed, during the histogram computation phase, the accelerator has to store information regarding the voxels volume content, and, therefore,

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



(a) Alveo U200 (b) ZCU104

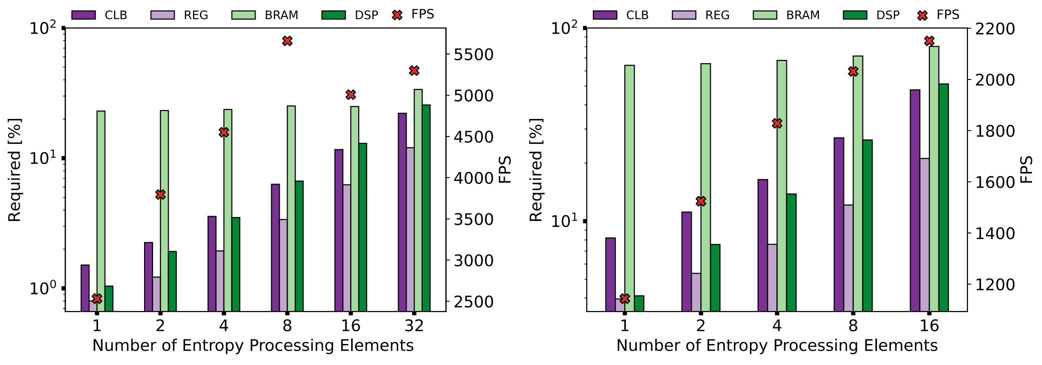
Fig. 5. Resource analysis scaling the number of processing elements from 1 to maximum device occupation with a power-of-two scheme for a high-performance board (AU200) and an embedded board (ZCU).

this step proves to be highly memory expensive. Moreover, to compute the FPS values, we have fed each accelerator with the same three random 3D input volumes of 512 × 512 × 512 generated with three different seeds, and the reported FPS values are the mean over thirty executions (10 runs per random seed). Looking at the FPS trend, they increment more than linearly when scaling from 1 to 8 HPEs, then decreasing when implementing 16 and 32 HPEs. Such behavior is imputable to a lower running frequency of the 16 and 32 HPEs accelerators compared to the 8 HPEs one. This analysis shows that the configuration exploiting 8 HPEs for the histogram computation stage is the best for the Alveo U200. Moreover, carrying out the same experiment on the Alveo U280, the 16 HPEs solution is the best one. Finally, exploring the impact of the Dual Core implementation on resource usage, it doubles the amount of resources required by the whole accelerator, while the best configuration remains unchanged for the Alveo U200. Conversely, for Alveo U280, the best configuration becomes the one exploiting 8 HPEs since the one with 16 HPEs is no more synthesizable due to a lack of resources.

Moving to the embedded case, Figure 5(b) presents a similar behavior to the high-performance case with a nearly linear trend in the BRAM increase, except for the absence of a drop in the FPS rate. Indeed, on the ZCU 104, we could scale only to 4 HPEs before saturating the resources, which is insufficient to trigger the frequency drop seen in the Alveo U200 above 8 HPEs. Based on these results, we selected the configuration exploiting 4 HPEsfor the histogram stage as the best one and the one exploiting 2 HPEs for the PYNQ ZU.

Further consideration can be made on the number of HPEs employed for the computation of the histogram; indeed, the accuracy in the computation of the MI value is independent of the number of HPEs since this step only involves integer computations that are commutative. Having identified the best configuration for all the tested boards, we analyze the scalability of the best MI accelerators in terms of PEs deployable for the second stage, namely the entropies computation (EPE). To carry out this exploration, we fix the HPEs, and as before, we increment the number of EPEs for the entropy computation stage following the power-of-two pattern from one to maximum device occupation. Also, in this case, Figure 6 reports the results for the Alveo U200 and the ZCU 104 as examples from the high-performance and embedded classes.

Figure 6(a) shows how the BRAM utilization remains pretty much constant in this second phase while increasing the number of PEs. Additionally, given that the entropies computation presents much more mathematical operations, we see a linear increment in the usage of almost all components, particularly DSPs. Concerning the FPS rate, it is computed as in the previous experiment,



(a) Alveo U200 (b) ZCU104

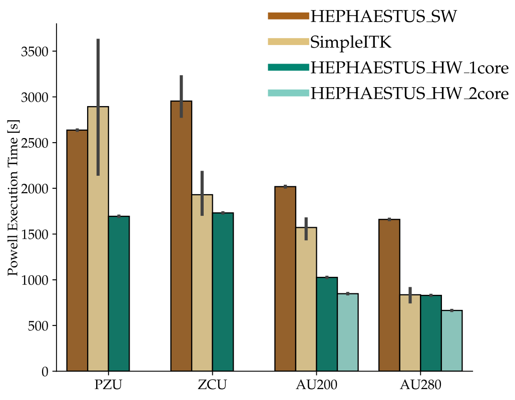
Fig. 6. Resource analysis scaling the number of processing elements for the entropy computation from 1 to maximum device occupation with a power-of-two scheme for a high-performance board (AU200) and an embedded board (ZCU). This analysis assumes 8 HPEs for the Alveo U200 and 4 HPEs for the ZCU104.

and it shows a similar trend to the one in Figure 5(a), with a peak in correspondence on 8 EPE and then a decrease when considering 16 and 32 EPEs. It is interesting to notice how moving from parallelizing only the histogram phase to both the histogram and the entropies allows for incrementing the FPS rate more than two times. Therefore, we select as the best configuration for the Alveo U200 the one with 8 PEs for both stages (8HPE-8EPE), while for the Alveo U280, we select the one exploiting 8 PEs for the histogram and 16 for the entropies (8HPE-16EPE).

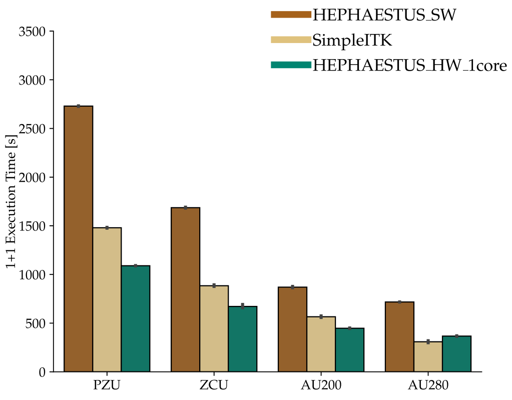
Regarding the embedded case, Figure 6(b) shows a similar trend to the high-performance case where increasing the number of EPEs keeps the BRAM usage nearly constant while it shows a linear increase of the other resources and specifically DSPs. Since the entropy phase relates more to mathematical operations than to storing vast data, we could scale the EPE number to 16. Additionally, the FPS rate show, as in Figure 5(b), a consistent increase moving from 1 to 16 EPEs reaching nearly two times more FPS compared to the histogram-only parallelized version. Therefore, based on the obtained results, we select, as the best model for the ZCU 104, the one with 4 PEs for the histogram computation and 16 PEs for the entropy one (4HPE-16EPE). Finally, based on the same consideration we selected for the PYNQ ZU the configuration with 2 PEs for the histogram and 16 PEs for the entropy computation (2HPE-16EPE).

4.2 Hephaestus 3D Image Registration Execution Time Evaluation

After identifying the best configuration per board, thanks to the previously described scalability analysis, we now evaluate the performance in terms of the execution time of the overall 3D image registration pipelines deployable thanks to Hephaestus. Specifically, as introduced before, we tested the proposed solution on a medical dataset, and we evaluated the execution time of the whole registration process by exploiting the two available optimizers, namely Powell's method and 1+1 Evolutionary. In particular, we tested Hephaestus in its two configurations: our pure software one and our accelerated version exploiting the proposed MI accelerator for each of the four considered boards. We compare these configurations against SimpleITK, a state-of-the-art cross-platform software [18] run on the on-board processors of the embedded devices and on the host CPU for the high-performance card, as in Table 2. Figure 7 reports the comparison among our software, our hardware accelerated version, and SimpleITK in terms of execution time, also considering any possible data transfers for boththe supported optimizers. As a general



(a) Powell's Execution Time



(b) 1+1 Evolutionary Execution Time

Fig. 7. Execution time analysis: comparison of SimpleITK and Hephaestus software and hardware implementation exploiting the two proposed optimizers and running on different devices.

consideration, we can see how Powell's method (Figure 7(a)), as already explained, requires more time to converge than 1+1 Evolutionary (Figure 7(b)), leading to a longer registration time.

Focusing on Powell's optimizer Figure 7(a), it is clear how all the solutions exploiting the MI accelerators outperform the corresponding SimpleITK implementation. Moreover, considering that our software implementation is generally slower than SimpleITK, our MI accelerators have a considerable positive impact on the overall execution time, opening an exciting possibility of integrating it into SimpleITK to reach even faster solutions. Additionally, our software and hardware versions show a negligible variance (except for one configuration) compared to SimpleITK, demonstrating better stability over ten runs. More in detail, Hephaestus reaches a maximum speedup over SimpleITK in the high-performance class of 1.53× on the Alveo U200 and of 1.71× for the embedded class on the PYNQ ZU, with an overall geomean speedup of 1.31×. Finally, when looking at the Dual Core implementation on the high-performance boards, since we are exploiting a batching approach parallelizing two MI computations and the correspondent transformations, we reach a speedup over SimpleITK of 1.85× and 1.26× on the Alveo U200 and Alveo U280, respectively, with an increased overall geomean speedup of 1.53×.

Moving to the 1+1 Evolutionary optimizer, Figure 7(b) shows that our accelerated implementations consistently outperform their SimpleITK counterparts except on the Alveo U280, where we reach comparable results. Also, in this case, our software version is always significantly slower than SimpleITK, demonstrating the considerable impact of accelerating the similarity metric. On the other hand, the lower gap between the hardware versions and SimpleITK when selecting 1+1 Evolutionary compared to Powell's method is given by the lower number of executions of the similarity metric computation. For these reasons, the impact of accelerating the similarity metric over the overall registration process is reduced, and, therefore, this lower gap is imputable to the gap (in favor of SimpleITK) between SimpleITK and our searching and transformation software part. Finally, Hephaestus, exploiting 1+1 Evolutionary, reaches a maximum speedup over SimpleITK in the high-performance class of 1.26× on the Alveo U200 and of 1.36× for the embedded class on the PYNQ ZU, with an overall geomean speedup of 1.17×.

As a final remark, considering only the MI computation and comparing its accelerated version to the software one over a volume of 512 × 512 × 512 we achieve a speedup that reaches up to 258× across the different machines. Such numbers further support the high positive impact of

Table 3. Accuracy Analysis: Comparison of SimpleITK and HEPHAESTUS in Terms of Intersection over Union (IoU), Clustered Per Device Class, and Reporting the Best Results Per Class in Bold

Work	Optimizer	Embedded			High-Performance	
		ZCU	PZU		AU200	AU280
HEPHAESTUS	1 + 1	$0.853 \pm 0.08^{\wedge}$	$0.870 \pm 0.079^{\wr}$	$\mathbf{0.889 \pm 0.026}^{\odot}$	**0.992 ± 0.004**	**0.992 ± 0.004**
SimpleITK [18]		$0.742 \pm 0.267^{\wedge}$	$0.665 \pm 0.304^{\wr}$	–	0.827 ± 0.152	0.863 ± 0.06
HEPHAESTUS	Powell	$0.946 \pm 0.056^{\wedge}$	$0.953 \pm 0.037^{\wedge}$	$\mathbf{0.996 \pm 0.001}^{\odot}$	**0.996 ± 0.001**	**0.996 ± 0.001**
SimpleITK [18]		$0.722 \pm 0.267^{\wedge}$	$0.732 \pm 0.27^{\wedge}$	–	0.816 ± 0.15	0.851 ± 0.06

$^{\odot}$Registration on 160 slices. $^{\wr}$Registration on 100 slices. $^{\wedge}$Registration on 60 slices.

HEPHAESTUS accelerators over the similarity metric computation and, therefore, over the overall registration process.

4.3 HEPHAESTUS Accuracy Evaluation

Having evaluated the execution time, we now move to assess HEPHAESTUS performance in terms of accuracy in carrying out the 3D affine image registration and compare it to SimpleITK on the same dataset. To evaluate the results' goodness, we select the Intersection over Union (IoU) as metric, which indicates how well two objects overlap. Indeed, if an optimal registration is achieved, the gold standard volume (achieved with a semi-automatic registration procedure) and the registered one should perfectly overlay, leading to a unitary IoU. Moreover, given that affine transformations preserve line parallelism, if the external structures coincide, the internal ones will also.

Table 3 reports the obtained IoU for the HEPHAESTUS hardware accelerated version for each available device and for their SimpleITK counterparts. Regarding the high-performance class, HEPHAESTUS consistently achieves better results than SimpleITK with a top improvement of 0.180 for the Alveo U200 employing Powell's optimizer. Moving to the embedded class, Table 3 shows the IoU results and the number of slices composing the considered volume. Indeed, as described in Section 3.3.1, given the low memory of the considered devices, we had to deal with a subvolume of the actual input to compute the optimal transformation parameters, then employed to transform the entire floating volume. Regarding the ZCU 104, our solution and SimpleITK work on 60 slices, but we can see how our implementation consistently outperforms SimpleITK with a maximum improvement of 0.224 when considering Powell's method. The same trend is visible when looking at the PYNQ ZU board, where considering both Powell with 60 slices and 1+1 Evolutionary with 100 slices, our solution consistently achieves higher accuracy, with Powell's method reaching a 0.221 improvement. Although such results, HEPHAESTUS can also deal with bigger subvolumes compared to SimpleITK, and considering a subvolume of 160 slices, it reaches a top IoU of 0.996 employing Powell's method. Such a result is a remarkable achievement considering that SimpleITK could not scale up to bigger volumes in such a constrained scenario.

A visual comparison also supports all these results; indeed, Figure 8 shows how overlapping the gold standard binarized (in green) and the obtained results (in yellow) with SimpleITK, and HEPHAESTUS hardware, the former shows a mismatch, while the ones obtained through HEPHAESTUS nearly perfectly align to the gold standard.

4.4 High-Performance Improvement Exploiting HEPHAESTUS Embedded Subsampling

Given the possibility of HEPHAESTUS to work on subvolumes of the input, as described in Section 3.3.1, we decide to explore the possibility of employing it not only in the embedded cases, where it is fundamental to perform the registration but also in the high-performance case, and, in particular, on the Alveo U200 to see how it impact the execution time and reachable accuracy.

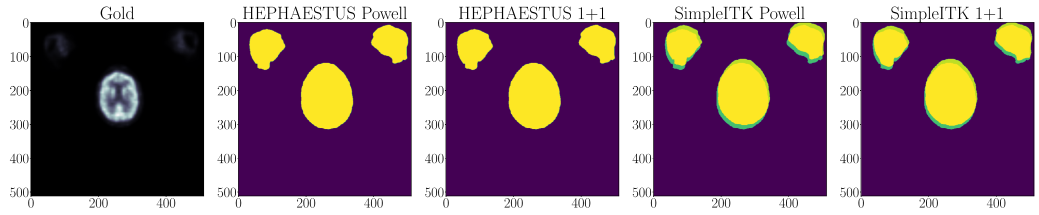


Fig. 8. A visual example of 3D affine image registration results. From left to right, the gold standard, the overlap between the gold standard (represented in green) and the registered image with Hephaestus Powell, Hephaestus 1+1 Evolutionary, and SimpleITK Powell and SimpleITK 1+1 Evolutionary (all in yellow). For visualization simplicity, we report a single 2D slice.

Therefore, we perform the 3D affine image registration on volumes of increasing dimensions starting from 20 slices to 160 slices with a step of 20 slices, and then we reported the results obtained considering the whole input volume (246 slices). Such slices were used to carry out the entire search process, while applying the final 3D transformation to the original input floating volume was directly made on it without volume subsampling since high-performance devices have more than enough memory.

Figure 9 clearly shows how, for the employed dataset, a subvolume of 20 slices is already enough to reach more than 0.96 of IoU while leading to an execution time reduction of 9× (only 0.031 lower IoU). Additionally, considering configurations that lead to an IoU above 0.98, we can see from Figure 9 that the one with 40, 140, and 160 slices show a reduction in execution time of 4.81×, 1.91×, and 1.68×, respectively. Although these improvements demonstrate how it would be beneficial to identify an automatic selection method for the subvolume dimension, this optimal value is highly dependent on unpredictable parameters related to the data to be registered. Indeed, it depends on the information distribution along the volume, where a subvolume of $N < N_{max}$ slices taken at the beginning, middle, or end of the 3D volume can impact the registration accuracy. Additionally, increasing the subvolume dimension but locating it in the low-informative part of the volume could not outperform a smaller subvolume that is extracted from a high-informative region. Indeed, if the considered volumes showcase enough information around their centers to represent the global misalignment among them correctly, small subvolumes will lead to highly accurate registration allowing for a considerable execution time reduction thanks to the fast convergence of the searching procedure. Based on these results, we believe that applying Hephaestus embedded tailoring on the high-performance board could be an interesting approach to reduce registration time further. Moreover, given the Hephaestus, MI accelerator software-programmability and thanks to the automation framework testing different subvolume dimensions does not require resynthesizing the accelerator but only rerunning the registration process, opening the possibility of searching for the optimal trade-off in a reasonable time.

4.5 Related Work Comparison

Finally, we conclude Hephaestus evaluation considering the image registration literature. As highlighted in our previous work [7], researchers struggle to fairly compare their approaches when considering intensity-based heuristics in multi-modal image registration. Indeed, different algorithmic combinations of similarity metric, transformation, and optimizer components (as in Figure 1) lead to considerable differences in the final registration performance. Moreover, a registration algorithm must usually deal with the trade-off between execution time and accuracy that requires application-specific choices. Therefore, Shams et al. [26] try to normalize these differences with a

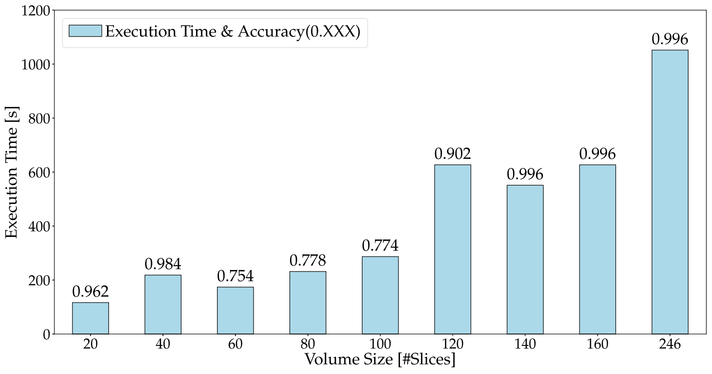


Fig. 9. Hephaestus speedup and accuracy variations while scaling the subvolume size analyzed from 20 slices to 160 with a 20-slice step compared to the entire input volume on the Alveo U200.

performance metric that accounts for the execution time, the number of processed voxels, and the optimizer iterations and is computed as:

$$Perf = \frac{Time[ms]}{MVoxel \times iterations} \tag{9}$$

Based on this definition, the optimizer iterations have an extremely high impact on the final performance without necessarily meaning a better execution time or accuracy (as shown in [7]). Moreover, we also adopt our previous enery efficiency evaluation metric [7] as:

$$EE = \frac{1}{Perf \times KWatt} = \frac{MVoxel \times iterations}{ms \times KWatt} \tag{10}$$

Looking for literature works, we have seen how they mainly deal with 2D registration, but comparing them with 3D solutions employing the performance metric is not an apple-to-apple comparison since it produces an entirely different order of magnitude in the obtained results. Therefore, trying to be as fair as possible, in Table 4, we only report works dealing with 3D image registration, along with the technology node, the proposed algorithm for the different blocks, the performance metric (Equation (9)), which is the lower, the better, and the energy efficiency (Equation (10)) which, conversely, is the higher, the better. Table 4 reports two works employing GPUs and SimpleITK as software counterparts of Hephaestus. Additionally, while Hephaestus and SimpleITK reach results in the same order of magnitude, the two works exploiting GPUs achieve way better performance and energy efficiency results. This difference is strictly connected to the heavy preprocessing done like considering only 15% of the total number of voxels, which, in some cases, as explained by the authors, also led to a failure of the registration process.

Based on these considerations, we discuss the results obtained by Hephaestus and SimpleITK regarding the performance metric and energy efficiency. As a first general trend, Table 4 shows, as expected, that the implementations exploiting 1+1 Evolutionary optimizer generally reach higher performance and energy efficiency results than the Powell-based counterparts, although, as we had seen, they are always less accurate. A second general trend that further supports the previously described results is that Hephaestus consistently achieves better performance and energy efficiency results than its SimpleITK counterparts with top improvements of 1.85× and 2.35×, respectively. Looking at the high-performance configurations, they are always faster than the embedded ones, and, therefore, they reach better performance results. Additionally, looking at the energy efficiency, they reach the same order of magnitude as the embedded ones, but such a similarity is given by

Table 4. Comparison with Related Work with Performance (Perf.) Measured as Proposed by Shams et al. [26] (the Lower the Better), and Energy Efficiency (Energy Eff.) as we Previously Proposed [7] (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	**1C-2HPE-16EPE**	3D Rigid	3D MI†	Powell	PYNQ ZU (16nm)	459.7509$^{\wedge\oslash}$	0.1954‡
	1C-2HPE-16EPE	3D Rigid	3D MI†	1+1	PYNQ ZU (16nm)	**207.8363**$^{\wr\otimes}$	**0.4216**‡
	1C-4HPE-16EPE	3D Rigid	3D MI†	Powell	ZCU104 (16nm)	470.3763$^{\wedge\oslash}$	0.1583‡
	1C-4HPE-16EPE	3D Rigid	3D MI†	1+1	ZCU104 (16nm)	213.4435$^{\wedge\otimes}$	0.348‡
	2C-8HPE-8EPE	3D Rigid	3D MI†	Powell	Alveo U200 (16nm)	56.1841$^{\circledast}$	0.1486‡
	2C-8HPE-8EPE	3D Rigid	3D MI†	Powell	Alveo U200 (16nm)	*13.0226*$^{\triangledown\circledast}$	*0.6413*‡
	1C-8HPE-8EPE	3D Rigid	3D MI†	Powell	Alveo U200 (16nm)	15.4949$^{\triangledown\circledast}$	0.6338‡
	1C-8HPE-8EPE	3D Rigid	3D MI†	Powell	Alveo U200 (16nm)	68.0001$^{\circledast}$	0.1444‡
	1C-8HPE-8EPE	3D Rigid	3D MI†	1+1	Alveo U200 (16nm)	34.6938$^{\otimes}$	0.2744‡
	2C-8HPE-16EPE	3D Rigid	3D MI†	Powell	Alveo U280 (16nm)	44.0455$^{\circledast}$	0.1959‡
	1C-16HPE-16EPE	3D Rigid	3D MI†	Powell	Alveo U280 (16nm)	54.9499$^{\circledast}$	0.1657‡
	1C-16HPE-16EPE	3D Rigid	3D MI†	1+1	Alveo U280 (16nm)	**28.4451**$^{\otimes}$	**0.3195**‡
GPU	[16]	3D Nonrigid	3D NMI†	Grad. Desc.†	GTX 580 (40nm)	0.1378$^{\spadesuit\clubsuit}$	31.52*
	[2]	3D Nonrigid†	3D NMI†	Stoch. Grad. Desc.†	Tesla K40c (28nm)	2.5750$^{\heartsuit}$	1.58*
CPU	SimpleITK [18]	3D Rigid	3D MI	Powell	ARM A53 (PZU) (16nm)	919.6886$^{\wedge\otimes}$	0.0857‡
	SimpleITK [18]	3D Rigid	3D MI	1+1	ARM A53 (PZU) (16nm)	282.2573$^{\wr\otimes}$	0.2800‡
	SimpleITK [18]	3D Rigid	3D MI	Powell	ARM A53 (ZCU) (16nm)	613.6293$^{\wedge\otimes}$	0.1529‡
	SimpleITK [18]	3D Rigid	3D MI	1+1	ARM A53 (ZCU) (16nm)	281.0516$^{\wedge\otimes}$	0.3335‡
	SimpleITK [18]	3D Rigid	3D MI	Powell	Intel i7-4770 (22nm)	121.8642$^{\otimes}$	0.0634‡
	SimpleITK [18]	3D Rigid	3D MI	1+1	Intel i7-4770 (22nm)	43.8520$^{\otimes}$	0.1796‡
	SimpleITK [18]	3D Rigid	3D MI	Powell	AMD R7-3700X (7nm)	64.8199$^{\otimes}$	0.0955‡
	SimpleITK [18]	3D Rigid	3D MI	1+1	AMD R7-3700X (7nm)	**23.9288**$^{\otimes}$	0.2592‡

In bold the best 3D metrics per device class, while in italics the best overall performing configuration (high-performance exploiting embedded tailoring). †Implemented in hardware. *Computed with Thermal Design Power (TDP) as power. ‡Computed with whole system power consumption. $^{\wr}$Registration on 100 slices. $^{\wedge}$Registration on 60 slices. $^{\triangledown}$Registration on 40 slices. $^{\otimes}$200 iterations. $^{\circledast}$234 iterations. $^{\oslash}$156 iterations. $^{\spadesuit}$Exploits the binning to reduce joint histogram sizes $^{\clubsuit}$This value is the result of several dataset-specific approximations and preprocessing that reduces the computation to 1/6 and lead to misregistrations [16]. $^{\heartsuit}$Working on 15% of image pixels.

the fact that the energy efficiency metric considers both the power consumption and the execution time. Therefore the similarity in the attained values resides in the fact that, while the embedded devices require less power than the high-performance ones, they are also slower.

Finally, a last consideration should be done regarding the two high-performance configurations on the Alveo U200 exploiting the tailoring made for the embedded (1C-8HPE-8EPE and 2C-8HPE-8EPE); indeed, they exploit a 40-slice subvolume to find the optimal transformation parameters reaching an accuracy above 0.98 while highly reducing the overall registration time. Therefore, Table 4 shows how they attain, although exploiting Powell's optimizer, the best performance and energy efficiency results with an improvement over SimpleITK run on the same CPU host (Intel i7-4770). In particular, Hephaestus achieves improvement in performance of 9.35× and 7.86×, and in energy efficiency of 10, 11× and 9.99×, on dual-core and single-core versions of AU200 respectively, along with higher accuracy (0.984 versus 0.816).

5 CONCLUSIONS AND FUTURE WORK

This paper presented Hephaestus, a 3D-based framework for creating and developing software programmable FPGA-based accelerators for the MI similarity metric integrated into affine image registration pipelines. Indeed, the software programmability easily adapts to different input volume dimensions and, thanks to the proposed hardware abstraction layer, transparently allows the end users to benefit from hardware acceleration without needing specific knowledge in hardware development. Finally, from Hephaestus validation, we have seen how it outperforms the other solutions in terms of accuracy, execution time (up to 1.85× faster), and energy efficiency (up to 2.35× more efficient).

Future Work – To improve Hephaestus, we aim to explore the use of pyramid levels for high-resolution images. Indeed, at the first pyramid level, a low-resolution version of the volumes should be registered to fix large misalignment, until, in the last pyramid level, the original high-resolution volumes are used, and the registration aims at registering eventual finer residual misalignment. In this second scenario, the proposed automation could enable the creation of a database of 3D MI accelerators with different image dimensions, where the FPGA reconfigurability is exploited to adapt the accelerator to the different pyramid levels.

Finally, exploring the URAM usage would open further improvements of Hephaestus. Thanks to this, we could cache the whole reference volume throughout the registration procedure and scale the number of PEs better, gaining more bandwidth, higher running frequencies, data locality, and power consumption improvements.

ACKNOWLEDGMENTS

We would like to thank the AMD University Program for providing the boards used in this paper.

REFERENCES

[1] AMD-Xilinx. 2021. PYNQ: Python for Productivity v2.7. https://github.com/Xilinx/PYNQ/tree/image_v2.7

[2] Parag Bhosale, Marius Staring, Zaid Al-Ars, and Floris F Berendsen. 2018. 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, 105740R.

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

[4] Anirban Chakraborty and Ayan Banerjee. 2020. A novel VLSI architecture of CORDIC based image registration. In *2020 Sixth International Conference on Bio Signals, Images, and Instrumentation (ICBSII)*. IEEE, 1–6.

[5] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, et al. 2018. Serving dnns in real time at datacenter scale with project brainwave. *IEEE Micro* 38, 2 (2018), 8–20.

[6] 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.

[7] Davide Conficconi, Eleonora D'Arnese, Emanuele Del Sozzo, Donatella Sciuto, and Marco D Santambrogio. 2021. A framework for customizable FPGA-based image registration accelerators. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 251–261.

[8] 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.

[9] Eleonora D'Arnese, Davide Conficconi, Emanuele Del Sozzo, Luigi Fusco, Donatella Sciuto, and Marco Domenico Santambrogio. 2022. Faber: A hardware/software toolchain for image registration. *IEEE Transactions on Parallel and Distributed Systems* 34, 1 (2022), 291–303.

[10] Eleonora D'Arnese, Emanuele Del Sozzo, Davide Conficconi, and Marco D Santambrogio. 2021. Exploiting heterogeneous architectures for rigid image registration. In *2021 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 1–5.

[11] Eleonora D'Arnese, Davide Conficconi, Marco D Santambrogio, and Donatella Sciuto. 2022. Reconfigurable architectures: The shift from general systems to domain specific solutions. In *Emerging Computing: From Devices to Systems: Looking Beyond Moore and Von Neumann*. Springer, 435–456.

[12] Jan Flusser, Tomas Suk, and Barbara Zitová. 2016. *2D and 3D Image Analysis by Moments*. John Wiley & Sons.

[13] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, et al. 2018. A configurable cloud-scale DNN processor for real-time AI. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1–14.

[14] Ali Hatamizadeh, Yucheng Tang, Vishwesh Nath, Dong Yang, Andriy Myronenko, Bennett Landman, Holger R Roth, and Daguang Xu. 2022. Unetr: Transformers for 3d medical image segmentation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 574–584.

[15] Hati International. 2019. The Massive Interest of Non-Healthcare Companies in the Healthcare Industry. https://hatiintl.com/blog/the-massive-interest-of-non-healthcare-companies-in-healthcare-industry

[16] 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.

[17] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 1–12.

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

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

[20] Meenal V Narkhede, Prashant P Bartakke, and Mukul S Sutaone. 2022. A review on weight initialization strategies for neural networks. *Artificial Intelligence Review* 55, 1 (2022), 291–322.

[21] 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

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems* 32 (2019).

[23] Edgar Riba, Dmytro Mishkin, Daniel Ponsa, Ethan Rublee, and Gary Bradski. 2020. Kornia: An open source differentiable computer vision library for pytorch. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 3674–3683.

[24] Septimiu E Salcudean, Hamid Moradi, David G Black, and Nassir Navab. 2022. Robot-assisted medical imaging: A review. *Proc. IEEE* (2022).

[25] 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.

[26] 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.

[27] Emanuele Del Sozzo, Davide Conficconi, Alberto Zeni, Mirko Salaris, Donatella Sciuto, and Marco D Santambrogio. 2022. Pushing the level of abstraction of digital system design: A survey on how to program FPGAs. *Comput. Surveys* 55, 5 (2022), 1–48.

[28] 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.

[29] Yucheng Tang, Dong Yang, Wenqi Li, Holger R Roth, Bennett Landman, Daguang Xu, Vishwesh Nath, and Ali Hatamizadeh. 2022. Self-supervised pre-training of swin transformers for 3d medical image analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 20730–20740.

[30] Russell Tessier, Kenneth Pocek, and Andre DeHon. 2015. Reconfigurable computing architectures. *Proc. IEEE* 103, 3 (2015), 332–354.

[31] Barbara Zitova and Jan Flusser. 2003. Image registration methods: A survey. *Image and Vision Computing* 21, 11 (2003), 977–1000.

Received 23 March 2023; revised 2 June 2023; accepted 1 July 2023