SIAM J. SCI. COMPUT. © 2018 Society for Industrial and Applied Mathematics
Vol. 40, No. 3, pp. B858–B888

A MATRIX-FREE APPROACH TO PARALLEL AND MEMORY-EFFICIENT DEFORMABLE IMAGE REGISTRATION*

LARS KÖNIG†, JAN RÜHAAK†, ALEXANDER DERKSEN†, AND JAN LELLMANN‡

Abstract. We present a novel computational approach to fast and memory-efficient deformable image registration. In the variational registration model, the computation of the objective function derivatives is the computationally most expensive operation, both in terms of runtime and memory requirements. In order to target this bottleneck, we analyze the matrix structure of gradient and Hessian computations for the case of the normalized gradient fields distance measure and curvature regularization. Based on this analysis, we derive equivalent matrix-free closed-form expressions for derivative computations, eliminating the need for storing intermediate results and the costs of sparse matrix arithmetic. This has further benefits: (1) matrix computations can be fully parallelized, (2) memory complexity for derivative computation is reduced from linear to constant, and (3) overall computation times are substantially reduced. In comparison with an optimized matrix-based reference implementation, the CPU implementation achieves speedup factors between 3.1 and 9.7, and we are able to handle substantially higher resolutions. Using a GPU implementation, we achieve an additional speedup factor of up to 9.2. Furthermore, we evaluated the approach on real-world medical datasets. On 10 publicly available lung computed tomography (CT) images from the DIR-Lab 4DCT dataset, we achieve the best mean landmark error of 0.93mm compared to other submissions on the DIR-Lab website, with an average runtime of only 9.23s. Complete nonrigid registration of full-size three-dimensional thorax-abdomen CT volumes from oncological followup is achieved in 12.6s. The experimental results show that the proposed matrix-free algorithm enables the use of variational registration models also in applications which were previously impractical due to memory or runtime restrictions.

Key words. deformable image registration, computational efficiency, parallel algorithms

AMS subject classifications. 92C55, 65K10, 65Y05

DOI. 10.1137/17M1125522

1. Introduction.

Image registration denotes the process of aligning two or more images for analysis and comparison [30]. *Deformable* registration approaches, which allow for nonrigid, nonlinear deformations, are of particular interest in many areas of medical imaging and contribute to the development of new technologies for diagnosis and therapy. Applications range from motion correction in gated cardiac positron emission tomography (PET) [16] and biomarker computation in regional lung function analysis [15] to intersubject registration for automated labeling of brain data [1].

For successful clinical adoption of deformable image registration, moderate memory consumption and low runtimes are indispensable, which is made more difficult by the fact that data is usually three-dimensional (3D) and therefore large. For example, in order to fuse computed tomography (CT) images for oncological followup, volumes with typical sizes of around $512 \times 512 \times 900$ voxels need to undergo full 3D registration; see Figure 1 for an example of a thorax-abdomen registration. In a clinical setting, these registrations have to be performed for every acquired volume and must be avail-

*Submitted to the journal's Computational Methods in Science and Engineering section April 13, 2017; accepted for publication (in revised form) March 30, 2018; published electronically June 12, 2018. The first preliminary results of this work were announced in 4- and 5-page conference proceedings [28, 25].

http://www.siam.org/journals/sisc/40-3/M112552.html

†Fraunhofer MEVIS, 23562 Lübeck, Germany (lars.koenig@mevis.fraunhofer.de, jan.ruehaak@mevis.fraunhofer.de, alexander.derksen@mevis.fraunhofer.de).

‡Institute of Mathematics and Image Computing, University of Lübeck, 23562 Lübeck, Germany, and Fraunhofer MEVIS, 23562 Lübeck, Germany (jan.lellmann@mic.uni-luebeck.de).

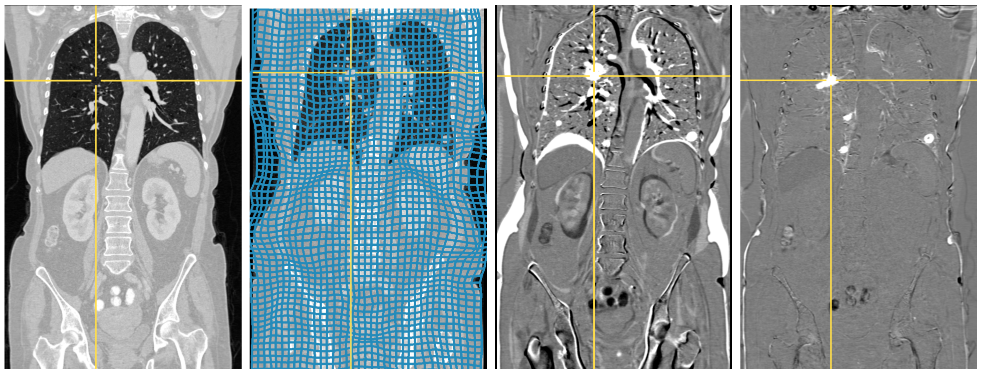


FIG. 1. *Registration of thorax-abdomen 3D CT scans for oncological followup.* (a) *coronal slice of prior scan,* (b) *coronal slice of current scan with deformation grid overlay,* (c) *subtraction image before registration,* (d) *subtraction image after registration. The current scan is deformed onto the prior scan using the proposed variational method. The subtraction image highlights the areas of change (white spots in the lung) corresponding to tumor growth. Image courtesy of Radboud University Medical Center, Nijmegen, The Netherlands. Used with permission.*

able quickly in order not to slow down the clinical workflow. In other situations, such as in population studies or during screening, the large number of required registrations demands efficient deformable image registration algorithms. In some areas, such as liver ultrasound tracking, real-time performance is even required [27, 11].

Consequently, a lot of research has been dedicated to increasing the efficiency of image registration algorithms; see [45, 47, 12] for an overview. The literature can be divided into approaches that aim at modifying the algorithmic structure of a registration method in order to increase efficiency [17, 18, 48] and approaches that specialize a given algorithm on a particular hardware platform—mostly on massively parallel architectures such as graphics processing units (GPUs) [24, 38, 4, 46], but also on digital signal processors (DSPs) [2] or field programmable gate arrays (FPGAs) [9].

Contributions and overview. In the following, we focus on variational approaches for image registration, which rely on numerical minimization of an energy function that depends on the input data. While such methods have been successfully used in various applications [43, 5, 16, 36], they involve expensive derivative computations, which are not straightforward to implement efficiently and hinder parallelizability.

In this work, we aim to work towards closing this gap:

- We analyze the derivative structure of the classical, widely known variational image registration model [35, 36] with the normalized gradient fields distance measure [19] and curvature regularization [14], which has been successfully used for various image registration problems [49, 26, 11] (sections 3 and 4).
- We study the use of efficient *matrix-free techniques* for derivative calculations in order to improve computational efficiency (section 4). In particular, we present fully matrix-free computation rules, based on the work on affine-linear image registration in [44] and deformable registration in [28, 25], for objective function gradient computations (section 4.1.1) and Gauss–Newton Hessian-vector multiplications (section 4.1.2).
- We perform a theoretical analysis of the proposed approach in terms of computational effort and memory usage (section 5).

- Finally, we quantitatively evaluate CPU and GPU implementations (section 6.3) of the new method with respect to speedup, parallel scalability, (section 6.1) and problem size and in comparison with two alternative methods (section 6.2). We demonstrate the real-world applicability on two medical applications (section 6.4).

Benefits of the proposed approach are full parallelization of objective function and derivative computations, reduction of derivative computation memory requirements from linear to constant, and a large reduction in overall runtime. The strategy can also be applied to other distance measures and regularizers, for which we discuss requirements and limitations.

In order to underline the merits of the approach in the clinic, we exemplarily consider two medical applications. First, the registration algorithm is employed for registering lung CT images in inhaled and exhaled positions (section 6.4.1). On the widely used DIR-Lab 4DCT database [8, 6], we achieve the best mean landmark error of 0.93 mm in comparison with the results reported at [7] with an average runtime of only 9.23 s, computed on a standard workstation. Second, we consider oncological follow-up studies in the thorax-abdomen region (section 6.4.2), where qualitatively convincing results are obtained at a clinically feasible runtime of 12.6 s.

2. Variational image registration framework.

2.1. Continuous model.

While there are numerous different applications of image registration, the goal is always the same: to obtain spatial correspondences between two or more images. Here we only consider the case of two images. The first image is denoted the *reference image* $\mathcal{R}$ (sometimes also called fixed image), and the second image is the *template image* $\mathcal{T}$ (or moving image) [36]. For 3D gray-scale images, the two images are given by functions $\mathcal{R} : \mathbb{R}^3 \to \mathbb{R}$ and $\mathcal{T} : \mathbb{R}^3 \to \mathbb{R}$ on domains $\Omega_{\mathcal{R}} \subset \mathbb{R}^3$ and $\Omega_{\mathcal{T}} \subset \mathbb{R}^3$, mapping each coordinate to a gray value. To establish correspondence between the images, a *deformation* function $\varphi : \Omega_{\mathcal{R}} \to \mathbb{R}^3$ is sought, which encodes the spatial alignment by mapping points from reference to template image domain. With the composition $\mathcal{T}(\varphi) : \Omega_{\mathcal{R}} \to \mathbb{R}$, $x \mapsto \mathcal{T}(\varphi(x))$, a *deformed template* image in the reference image domain can be obtained.

In order to find a suitable deformation φ, two competing criteria must be balanced. First, the deformed template image should be similar to the reference image, determined by some *distance measure* $\mathcal{D}(\mathcal{R}, \mathcal{T}(\varphi))$. Second, as the distance measure alone is generally not enough to make the problem well-posed and robust, a *regularizer* $\mathcal{S}(\varphi)$ is introduced, which requires the deformation to be "reasonable." An optimal deformation meeting both criteria is then found by solving the optimization problem

$$(1) \qquad \min_{\varphi:\Omega_{\mathcal{R}}\to\mathbb{R}^3} \mathcal{J}(\varphi), \quad \mathcal{J}(\varphi) := \mathcal{D}(\mathcal{R}, \mathcal{T}(\varphi)) + \alpha\mathcal{S}(\varphi),$$

where $\alpha > 0$ is a weighting parameter. Numerous different approaches and definitions for distance measures [10, 51, 35] and regularizers [14, 3, 5] have been presented, each with specific properties tailored to the application.

This work focuses on the normalized gradient fields (NGF) distance measure. It was first presented in [19] and has since been successfully used in a wide range of applications [43, 41, 26]. The NGF distance is particularly suited for multimodal registration problems, where image intensities of reference and template images are related in complex and usually nonlinear ways, and represents a fast and robust alternative [19] to the widely used mutual information [10, 51]. It is defined as

$$\mathcal{D}(\varphi) = \int_{\Omega_\mathcal{R}} 1 - \left(\frac{\langle \nabla \mathcal{T}(\varphi), \nabla \mathcal{R} \rangle + \tau \varrho}{\|\nabla \mathcal{T}(\varphi)\|_\tau \|\nabla \mathcal{R}\|_\varrho} \right)^2 \mathrm{d}x, \tag{2}$$

where $\|\cdot\|_\varepsilon := \sqrt{\langle \cdot, \cdot \rangle + \varepsilon^2}$, and $\tau, \varrho > 0$ are parameters that control filtering of noise in each image. The underlying assumption is that even in different imaging modalities, intensity *changes* still take place at corresponding locations. Therefore, the NGF distance advocates parallel alignment of *image gradient directions*, i.e., edge orientations.

For the regularizer $\mathcal{S}$, we use *curvature regularization* [14],

$$\mathcal{S}(\varphi) = \int_{\Omega_R} \sum_{j=1}^{3} (\Delta u_j)^2 \mathrm{d}x.$$

$\mathcal{S}$ is defined in terms of the *displacement* $u := (u_1, u_2, u_3)^\top := \varphi - \mathrm{id}$, where id is the identity transformation. The curvature regularizer penalizes second-order derivatives and thus favors "smooth" deformations, while not penalizing affine transformations such as translations or rotations. Similar to NGF, curvature regularization has been successfully used in multiple applications; see, e.g., [43, 26, 27].

2.2. Discretization. In order to solve the registration problem (1), we follow a *discretize-then-optimize* approach [36]: First, the deformation φ and objective function $\mathcal{J}$ are turned into a discretized energy. Second, this discretized energy is minimized using standard methods from numerical optimization (section 2.3). For reference, the notation introduced in the following is summarized in Table 1.

The *image domain* $\Omega_\mathcal{R}$ is discretized on a cuboid grid with m_x, m_y, m_z grid cells (voxels) in each dimension and $\bar{m} := m_x m_y m_z$ cells in total. The corresponding grid spacings are denoted by h_x, h_y, h_z and the cell volume by $\bar{h} := h_x h_y h_z$. The associated *image grid* is defined as the *cell-centers*

$$\mathcal{X} := \left\{ \left(\hat{\imath} h_x - \tfrac{h_x}{2}, \hat{\jmath} h_y - \tfrac{h_y}{2}, \hat{k} h_z - \tfrac{h_z}{2} \right) \middle| \hat{\imath} = 1, \ldots, m_x, \hat{\jmath} = 1, \ldots, m_y, \hat{k} = 1, \ldots, m_z \right\}.$$

Using a lexicographic ordering defined by the index mapping $i := \hat{\imath} + \hat{\jmath}\, m_x + \hat{k}\, m_x m_y$, the elements in $\mathcal{X}$ can be assembled into a single vector $\mathbf{x} := (\mathrm{x}_1, \ldots, \mathrm{x}_{3\bar{m}}) \in \mathbb{R}^{3\bar{m}}$, which is obtained by first storing all x-, then all y-, and finally all z-coordinates in the order prescribed by i. The ith grid point in $\mathcal{X}$ is then $\mathbf{x}_i := (\mathrm{x}_i, \mathrm{x}_{i+\bar{m}}, \mathrm{x}_{i+2\bar{m}}) \in \mathbb{R}^3$. Evaluating the reference image on all grid points can be compactly written as $R(\mathbf{x}) := \mathcal{R}\left(\mathbf{x}_i\right)_{i=1,\ldots,\bar{m}}$, resulting in a vector $R(\mathbf{x}) \in \mathbb{R}^{\bar{m}}$ of image intensities at the grid points.

In a similar way, we define the discretized *deformation* $\mathbf{y} := \varphi(\mathbf{x}^\mathrm{y})$ as the evaluation of φ on a *deformation grid* $\mathbf{x}^\mathrm{y}$. Again we order all components of $\mathbf{y}$ in a vector, $\mathbf{y} = (\mathrm{y}_1, \ldots, \mathrm{y}_{3\bar{m}^\mathrm{y}})$, so that $\varphi(\mathbf{x}_i^\mathrm{y}) = (\mathrm{y}_i, \mathrm{y}_{i+\bar{m}^\mathrm{y}}, \mathrm{y}_{i+2\bar{m}^\mathrm{y}})$, where the deformation grid has grid spacings $h_x^\mathrm{y}, h_y^\mathrm{y}, h_z^\mathrm{y}$, and $m_x^\mathrm{y}, m_y^\mathrm{y}, m_z^\mathrm{y}$ denote the number of grid points in each direction with $\bar{m}^\mathrm{y} := m_x^\mathrm{y} m_y^\mathrm{y} m_z^\mathrm{y}$.

Importantly, instead of using cell-center coordinates, the deformation is discretized on cell corners, i.e., using a *nodal* grid. This ensures that the deformation is discretized up to the boundary of the image domain, enabling an efficient grid conversion without requiring extrapolation or additional boundary conditions (section 4.3).

Table 1

Summary of notation introduced in section 2.2 for discretization of the continuous registration problem. All notation regarding the deformation grid is identical to the corresponding notation for the image grid, except for an additional superscript "y"; e.g., m denotes the number of grid cells in the image grid, and m^y denotes the number of grid cells in the deformation grid.

x, y, z	spatial coordinate axes
$\mathcal{R}, \mathcal{T}$	reference, template image functions, $\mathcal{R}, \mathcal{T} : \mathbb{R}^3 \to \mathbb{R}$
$\Omega_\mathcal{R}$	reference image domain, $\Omega_\mathcal{R} \subset \mathbb{R}^3$
φ	deformation function, $\varphi : \Omega_\mathcal{R} \to \mathbb{R}^3$
m, m^y	number of grid cells in the image/deformation grid, $m = (m_x, m_y, m_z)$
$\bar{m}, \bar{m}^\mathrm{y}$	total number of grid cells, $\bar{m} = m_x m_y m_z$
h, h^y	grid spacings for each coordinate direction, $h = (h_x, h_y, h_z)$
$\bar{h}, \bar{h}^\mathrm{y}$	volume of a single grid cell, $\bar{h} = h_x h_y h_z$
i	linear index with $i = \hat{\imath} + \hat{\jmath} m_x + \hat{k} m_x m_y$
$i - x, i + x$	indices of neighbors in x-direction, taking boundary conditions into account
$i - y, i + y$	indices of neighbors in y-direction, taking boundary conditions into account
$i - z, i + z$	indices of neighbors in z-direction, taking boundary conditions into account
$\mathbf{x}, \mathbf{x}^\mathrm{y}$	vector of lexicographically ordered grid points, $\mathbf{x} = (\mathrm{x}_1, \ldots, \mathrm{x}_{3\bar{m}}) \in \mathbb{R}^{3\bar{m}}$
$\mathbf{x}_i, \mathbf{x}_i^\mathrm{y}$	ith grid point, $\mathbf{x}_i = (\mathrm{x}_i, \mathrm{x}_{i+\bar{m}}, \mathrm{x}_{i+2\bar{m}}) \in \mathbb{R}^3$
$\mathbf{y}$	deformation discretized on deformation grid, $\mathbf{y} = (\mathrm{y}_1, \ldots, \mathrm{y}_{3\bar{m}^\mathrm{y}}) \in \mathbb{R}^{3\bar{m}^\mathrm{y}}$, $(\mathrm{y}_i, \mathrm{y}_{i+\bar{m}^\mathrm{y}}, \mathrm{y}_{i+2\bar{m}^\mathrm{y}}) = \varphi(\mathbf{x}_i^\mathrm{y})$
$\mathbf{u}$	displacement discretized on deformation grid, $\mathbf{u} = \mathbf{y} - \mathbf{x}^\mathrm{y} \in \mathbb{R}^{3\bar{m}^\mathrm{y}}$
P	grid conversion operator from deformation grid to image grid, $P : \mathbb{R}^{3\bar{m}^\mathrm{y}} \to \mathbb{R}^{3\bar{m}}$
$\hat{\mathbf{y}}$	deformation discretized on image grid points, $\hat{\mathbf{y}} = P(\mathbf{y}) \in \mathbb{R}^{3\bar{m}}$
$\hat{\mathbf{y}}_i$	ith deformation point, $\hat{\mathbf{y}}_i = (P(\mathbf{y})_i, P(\mathbf{y})_{i+\bar{m}}, P(\mathbf{y})_{i+2\bar{m}})$
$R(\mathbf{x})$	reference image evaluated on image grid, $R : \mathbb{R}^{3\bar{m}} \to \mathbb{R}^{\bar{m}}$
R_i	scalar value of reference image at ith grid point, $R_i = \mathcal{R}(\mathbf{x}_i) \in \mathbb{R}$
$T(P(\mathbf{y}))$	template evaluated on deformed image grid, $T : \mathbb{R}^{3\bar{m}} \to \mathbb{R}^{\bar{m}}$
T_i	scalar value of reference image at deformed ith grid point, $T_i = \mathcal{T}(\hat{\mathbf{y}}_i) \in \mathbb{R}$

This results in $m_x^\mathrm{y}, m_y^\mathrm{y}, m_z^\mathrm{y}$ coordinates in each direction and $\bar{m}^\mathrm{y} := m_x^\mathrm{y} m_y^\mathrm{y} m_z^\mathrm{y}$ grid points in total. The set of all nodal grid points is given by

$$\mathcal{X}^\mathrm{y} := \left\{ \left(\hat{\imath} h_x^\mathrm{y}, \hat{\jmath} h_y^\mathrm{y}, \hat{k} h_z^\mathrm{y} \right) \,\middle|\, \hat{\imath} = 0, \ldots, m_x^\mathrm{y} - 1, \hat{\jmath} = 0, \ldots, m_y^\mathrm{y} - 1, \hat{k} = 0, \ldots, m_z^\mathrm{y} - 1 \right\}.$$

Analogously to the image grid, using the linear index $i := \hat{\imath} + \hat{\jmath}\, m_x^\mathrm{y} + \hat{k}\, m_x^\mathrm{y} m_y^\mathrm{y}$, the points from $\mathcal{X}^\mathrm{y}$ can be ordered lexicographically in a vector $\mathbf{x}^\mathrm{y} := (\mathrm{x}_1^\mathrm{y}, \ldots, \mathrm{x}_{3\bar{m}^\mathrm{y}}^\mathrm{y}) \in \mathbb{R}^{3\bar{m}^\mathrm{y}}$, with a single point having the coordinates $\mathbf{x}_i^\mathrm{y} := (\mathrm{x}_i^\mathrm{y}, \mathrm{x}_{i+\bar{m}^\mathrm{y}}^\mathrm{y}, \mathrm{x}_{i+2\bar{m}^\mathrm{y}}^\mathrm{y})$.

We refer the reader to [36] for further discussion on different grids. In this work, the deformation grid size is always chosen with $m_k^\mathrm{y} - 1 \leq m_k,\ k = x, y, z$, so that the image grid is as least as fine as the deformation grid.

The separate choice of grids allows one to use a coarser grid for the deformation—which directly affects the number of unknowns—and a high-resolution grid for the input images. However, it adds an extra interpolation step, as in order to compute the distance measure, the deformed template image $\mathcal{T}(\varphi)$ needs to be evaluated on the same grid as the reference image $\mathcal{R}$.

We use a linear interpolation function $P : \mathbb{R}^{3\bar{m}^\mathrm{y}} \to \mathbb{R}^{3\bar{m}}$, mapping between image grid and deformation grid, and define $T(P(\mathbf{y})) := (\mathcal{T}(\hat{\mathbf{y}}_i))_{i=1,\ldots,\bar{m}} \in \mathbb{R}^{\bar{m}}$ with $\hat{\mathbf{y}}_i := (P(\mathbf{y})_i, P(\mathbf{y})_{i+\bar{m}}, P(\mathbf{y})_{i+2\bar{m}})$. For evaluating the template image $\mathcal{T}$, trilinear interpolation with Dirichlet boundary conditions is used. This is a reasonable assumption for medical images, which often exhibit a black background.

Discretizing the integral in (2) using the midpoint quadrature rule and denoting by T_i, R_i the ith component function, the compact expression

$$D(\mathbf{y}) = \bar{h} \sum_{i=1}^{\bar{m}} \left(1 - \left(\frac{\frac{1}{2}\langle \tilde{\nabla} T_i(P(\mathbf{y})), \tilde{\nabla} R_i(\mathbf{x}) \rangle + \tau\varrho}{\|\tilde{\nabla} T_i(P(\mathbf{y}))\|_\tau \|\tilde{\nabla} R_i(\mathbf{x})\|_\varrho} \right)^2 \right) \tag{3}$$

is obtained for the full discretized NGF term. The factor 1/2 is caused by the "square-then-average" scheme for discretizing the gradient $\tilde{\nabla}$: Given a discretized image $I \in \mathbb{R}^{\bar{m}}$, we define the *backward difference operator* at the ith grid point, $\tilde{\nabla}_- I_i : \mathbb{R}^{\bar{m}} \to \mathbb{R}^3$, as

$$\tilde{\nabla}_- I_i := \left(\frac{I_i - I_{i-x}}{h_x}, \frac{I_i - I_{i-y}}{h_y}, \frac{I_i - I_{i-z}}{h_z} \right),$$

where the neighborhood is defined via

(4)
$$\begin{aligned}
i - x &:= \max(\hat{\imath} - 1, 1) + \hat{\jmath}\, m_x + \hat{k} m_x m_y, & i + x &:= \min(\hat{\imath} + 1, m_x) + \hat{\jmath}\, m_x + \hat{k} m_x m_y,\\
i - y &:= \hat{\imath} + \max(\hat{\jmath} - 1, 1) m_x + \hat{k} m_x m_y, & i + y &:= \hat{\imath} + \min(\hat{\jmath} + 1, m_y)\, m_x + \hat{k} m_x m_y,\\
i - z &:= \hat{\imath} + \hat{\jmath} m_x + \max(\hat{k} - 1, 1) m_x m_y, & i + z &:= \hat{\imath} + \hat{\jmath} m_x + \min(\hat{k} + 1, m_z)\, m_x m_y
\end{aligned}$$

and $i_0 := i$. This allows us to implement Neumann boundary conditions without special treatment of the boundary cells. By $\tilde{\nabla}_+ I_i$ we denote the corresponding forward finite differences operator. We combine both operators into one,

$$\tilde{\nabla} I_i := \left(\tilde{\nabla}_- I_i, \tilde{\nabla}_+ I_i \right), \tag{5}$$

with $\tilde{\nabla} I_i : \mathbb{R}^{\bar{m}} \to \mathbb{R}^6$, and define $\|\tilde{\nabla} I_i\|_\varepsilon := \sqrt{\frac{1}{2}\langle \tilde{\nabla} I_i, \tilde{\nabla} I_i \rangle + \varepsilon^2}$. This "square-then-average" scheme for approximating the terms in (2) allows us to use "short" forward and backward differences, which better preserve high frequency gradients than "long" central differences [21]. The factor 1/2 is required in order to faithfully discretize (2) and can be interpreted as an averaging of the squared norms of the forward and backward operators.

Similarly to the NGF, the curvature regularizer is discretized as

$$S(\mathbf{y}) = \bar{h}^{\mathrm{y}} \sum_{i=1}^{\bar{m}^{\mathrm{y}}} \sum_{d=0}^{2} \left(\tilde{\Delta} \mathbf{u}_{i + d\bar{m}^{\mathrm{y}}} \right)^2, \tag{6}$$

with the discretized *displacement* $\mathbf{u} := (\mathrm{u}_1, \dots, \mathrm{u}_{3\bar{m}^{\mathrm{y}}}) := \mathbf{y} - \mathbf{x}^{\mathrm{y}}$ and the discretized Laplace operator

$$\tilde{\Delta} \mathbf{u}_{i + d\bar{m}^{\mathrm{y}}} := \sum_{k \in \{x,y,z\}} \frac{1}{(h_k^{\mathrm{y}})^2} \left(\mathrm{u}_{i-k+d\bar{m}^{\mathrm{y}}} - 2\, \mathrm{u}_{i+d\bar{m}^{\mathrm{y}}} + \mathrm{u}_{i+k+d\bar{m}^{\mathrm{y}}} \right), \tag{7}$$

with homogeneous Neumann boundary conditions. Overall, we obtain the discretized version

$$\min_{\mathbf{y} \in \mathbb{R}^{3\bar{m}^{\mathrm{y}}}} J(\mathbf{y}), \quad J(\mathbf{y}) := D(\mathbf{y}) + \alpha S(\mathbf{y}), \tag{8}$$

of the minimization problem (1), which can then be solved using quasi-Newton methods, as summarized in the following section.

2.3. Numerical optimization. In order to find a minimizer of the discretized objective function (8), we use iterative Newton-like optimization schemes. In each step of the iteration, an equation of the form

$$\hat{\nabla}^2 J(\mathbf{y}^k)\mathbf{s}^k = -\nabla J(\mathbf{y}^k) \tag{9}$$

is solved for a descent direction $\mathbf{s}^k$. Then $\mathbf{y}^k$ is updated via

$$\mathbf{y}^{k+1} = \mathbf{y}^k + \eta \mathbf{s}^k,$$

where the step length η is determined by Armijo line search [39, 36]. The matrix $\hat{\nabla}^2 J$ should approximate the Hessian $\nabla^2 J(\mathbf{y}^k)$. Here we consider the *Gauss–Newton* scheme and the *L-BFGS* scheme, which have both been used in different image registration applications [50, 25, 43]. The minimization is embedded in a coarse-to-fine multilevel scheme, where the problem is solved on consecutively finer deformation and image grids.

2.3.1. Gauss–Newton. The Gauss–Newton scheme uses a quadratic approximation of the Hessian and is suitable for least-squares-type objective functions of the form

$$\hat{J}(\mathbf{y}) = \hat{r}(\mathbf{y})^\top \hat{r}(\mathbf{y}) = \|\hat{r}(\mathbf{y})\|_2^2, \tag{10}$$

where $\hat{r}(\mathbf{y})$ is a residual function, which can depend in a nonlinear way on the unknown $\mathbf{y}$. The gradient in the Newton equation (9) can be written as $\nabla \hat{J} = 2\,\hat{r}^\top d\hat{r}$, where $d\hat{r}$ is the Jacobian of $\hat{r}$, and the Hessian is approximated by $H := \hat{\nabla}^2 \hat{J} = 2 d\hat{r}^\top d\hat{r}$ [39]. This approximation discards second-order derivative parts of the Hessian and guarantees a symmetric positive semidefinite Hessian approximation H, so that $\mathbf{s}^k$ in (9) is always a descent direction.

We apply this approximation only to the Hessian of the distance measure D, $\nabla^2 D \approx H$. For the regularizer S, we use the exact Hessian, which is readily available, as S is a quadratic function. The (quasi-)Newton equation (9) can be written as

$$(H + \alpha \nabla^2 S)\mathbf{s}^k = -(\nabla D + \alpha \nabla S),$$

which is then approximately solved in each step using a conjugate-gradient (CG) iterative solver [39].

2.3.2. L-BFGS. Instead of directly determining a Hessian approximation at each step, the L-BFGS scheme iteratively updates the approximation using previous and current gradient information.

L-BFGS requires an initial approximation H_0 of the Hessian, which is often chosen to be a multiple of the identity. However, in our case we can incorporate the known Hessian of the regularizer S by choosing $H_0 = \nabla^2 S + \gamma I$, where I is the identity matrix and $\gamma > 0$ is a parameter, in order to make H_0 positive definite; further details can be found in [36, 39].

3. Analytical derivatives.

As can be seen from section 2.3, computing derivatives of distance measure and regularizer is a critically important task for the minimization of the discretized objective function. In this section, algebraic formulations of the required derivatives are presented which allow for a closer analysis and construction of specialized computational schemes.

3.1. Curvature. The discretized curvature regularizer (6) is a quadratic function involving the (linear) Laplace operator $\tilde{\Delta}\mathbf{u}_i$. Thus, using the chain rule the ith element of the gradient can explicitly be computed by

$$(\nabla S(\mathbf{y}))_{i+d\bar{m}^y} = 2\,\bar{h}^y \tilde{\Delta}\left(\tilde{\Delta}\mathbf{u}\right)_{i+d\bar{m}^y}, \tag{11}$$

with $d \in \{0, 1, 2\}$ for the directional derivatives. The Hessian is constant and can be immediately seen from (7).

3.2. NGF. The inner computations of the NGF in (3) can be rewritten more compactly by introducing a residual function $r : \mathbb{R}^{\bar{m}} \to \mathbb{R}^{\bar{m}}$ with components

$$r_i(T) := \frac{\frac{1}{2}\langle \tilde{\nabla} T_i, \tilde{\nabla} R_i\rangle + \tau\varrho}{\|\tilde{\nabla} T_i\|_\tau \|\tilde{\nabla} R_i\|_\varrho}, \tag{12}$$

so that the NGF term in (3) becomes

$$D(\mathbf{y}) = \bar{h}\sum_{i=1}^{\bar{m}} \left(1 - r_i(T(P(\mathbf{y})))^2\right). \tag{13}$$

Introducing a reduction function $\psi : \mathbb{R}^{\bar{m}} \to \mathbb{R}, (r_1, \ldots, r_{\bar{m}})^\top \mapsto \bar{h}\sum_{i=1}^{\bar{m}}(1 - r_i^2)$, it holds that

$$D(\mathbf{y}) = \psi(r(T(P(\mathbf{y})))). \tag{14}$$

This composite function maps the deformation $\mathbf{y} \in \mathbb{R}^{3\bar{m}^y}$ onto a scalar image similarity in four steps,

$$\mathbb{R}^{3\bar{m}^y} \xrightarrow{P} \mathbb{R}^{3\bar{m}} \xrightarrow{T} \mathbb{R}^{\bar{m}} \xrightarrow{r} \mathbb{R}^{\bar{m}} \xrightarrow{\psi} \mathbb{R}. \tag{15}$$

Using the chain rule, the gradient of D can now be obtained as a product of (sparse) Jacobian matrices

$$\nabla D = \left(\frac{\partial\psi}{\partial r}\frac{\partial r}{\partial T}\frac{\partial T}{\partial P}\frac{\partial P}{\partial \mathbf{y}}\right)^\top. \tag{16}$$

The Gauss–Newton approximation of the Hessian (section 2.3.1) becomes

$$H = 2\,\bar{h}\,\frac{\partial P}{\partial \mathbf{y}}^\top \frac{\partial T}{\partial P}^\top \frac{\partial r}{\partial T}^\top \frac{\partial r}{\partial T}\frac{\partial T}{\partial P}\frac{\partial P}{\partial \mathbf{y}}. \tag{17}$$

Note that, in contrast to the classical Gauss–Newton method (10), the residual in (13) has a different sign. Therefore, in order to compute a descent direction in (9), the sign of the Hessian approximation has been inverted (see also [36]).

Using these formulations, the evaluation of the gradient and Hessian can be implemented based on sparse matrix representations [36]. However, while insightful for educational and analytic purposes, these schemes have important shortcomings: storing the matrix elements requires large amounts of memory, while assembling the sparse matrices and sparse matrix multiplications impedes efficient parallelization. In the following sections, we will therefore focus on strategies for direct evaluation that do not require intermediate storage and are highly parallel.

4. Derivative analysis and matrix-free computation schemes.

From a computational perspective, the NGF (13) and curvature formulations (6) are very amenable to parallelization, as all summands are independent of each other, and there are no inherent intermediate matrix structures required. However, this advantage is lost when the evaluation of the derivatives is implemented using individual matrices for the factors in (16) and (17).

In order to exploit the structure of the partial derivatives, in the following sections we will analyze the matrix structure of all derivatives and derive equivalent closed-form expressions that do not rely on intermediate storage.

We will make substantial use of the fact that the matrix structure of the partial derivatives is independent of the data. Therefore, the locations of nonzero elements are known a priori, which allows us to derive efficient sparse schemes.

4.1. Derivative computations for NGF.

4.1.1. Gradient.

The NGF gradient computation in (16) involves four different Jacobian matrices. The derivative of the vector reduction $\psi(r)$ is straightforward:

$$\frac{\partial \psi}{\partial r} = -2\bar{h}r = -2\bar{h}\,(r_1, \ldots, r_{\bar{m}}) \in \mathbb{R}^{1\times\bar{m}}. \tag{18}$$

In contrast, the computation of $\frac{\partial r}{\partial T} \in \mathbb{R}^{\bar{m}\times\bar{m}}$ is considerably more involved. In order to calculate a single row of this Jacobian $\frac{\partial r_i}{\partial T}$, the definition (13) can be used and interpreted as a quotient

$$r_i = \frac{r_{1,i}}{r_{2,i}} = \frac{\frac{1}{2}\langle \tilde{\nabla} T_i, \tilde{\nabla} R_i\rangle + \tau\varrho}{\|\tilde{\nabla} T_i\|_\tau \|\tilde{\nabla} R_i\|_\varrho}. \tag{19}$$

Separately differentiating numerator and denominator, this yields

$$\frac{\partial r_{1,i}}{\partial T} = \frac{1}{2}(\tilde{\nabla} R_i)^\top \frac{\partial \tilde{\nabla}_i}{\partial T} \in \mathbb{R}^{1\times\bar{m}} \quad \text{and} \quad \frac{\partial r_{2,i}}{\partial T} = \|\tilde{\nabla} R_i\|_\varrho \frac{(\tilde{\nabla} T_i)^\top}{2\|\tilde{\nabla} T_i\|_\tau} \frac{\partial \tilde{\nabla}_i}{\partial T} \in \mathbb{R}^{1\times\bar{m}}.$$

Included in both of these terms is the derivative of the finite differences gradient computation $\tilde{\nabla} T_i : \mathbb{R}^{\bar{m}} \to \mathbb{R}^6$ as defined in (5), i.e., mapping an image to the forward and backward difference gradients. As the gradient at a single point only depends on the values at the point itself and at neighboring points, the Jacobian $\frac{\partial \tilde{\nabla}_i}{\partial T} \in \mathbb{R}^{6\times\bar{m}}$ is of the form

$$\frac{\partial \tilde{\nabla}_i}{\partial T} = \begin{array}{c} \begin{array}{ccccccc} i-z & i-y & i-x & i & i+x & i+y & i+z \end{array} \\ \left(\begin{array}{ccccccc} & & \frac{-1}{h_x} & \frac{1}{h_x} & & & \\ & \frac{-1}{h_y} & & \frac{1}{h_y} & & & \\ \frac{-1}{h_z} & & & \frac{1}{h_z} & & & \\ & & & \frac{-1}{h_x} & \frac{1}{h_x} & & \\ & & & \frac{-1}{h_y} & & \frac{1}{h_y} & \\ & & & \frac{-1}{h_z} & & & \frac{1}{h_z} \end{array}\right) \end{array},$$

where only nonzero elements are shown and the nonzero column indices are displayed above the matrix. The quotient rule, applied to (19), leads to

$$\frac{\partial r_i}{\partial T} = \frac{1}{r_{2,i}^2}\left(\frac{\partial r_{1,i}}{\partial T} r_{2,i} - r_{1,i}\frac{\partial r_{2,i}}{\partial T}\right) = \frac{\partial r_{1,i}}{\partial T}\frac{1}{r_{2,i}} - \frac{r_{1,i}}{r_{2,i}^2}\frac{\partial r_{2,i}}{\partial T} \tag{20}$$

$$= \frac{(\tilde\nabla R_i)^\top}{2\|\tilde\nabla T_i\|_\tau \|\tilde\nabla R_i\|_\varrho}\frac{\partial \tilde\nabla_i}{\partial T} - \frac{\frac{1}{2}\langle \tilde\nabla T_i, \tilde\nabla R_i\rangle + \tau\varrho}{\|\tilde\nabla T_i\|_\tau^2 \|\tilde\nabla R_i\|_\varrho^2}\|\tilde\nabla R_i\|_\varrho \frac{(\tilde\nabla T_i)^\top}{2\|\tilde\nabla T_i\|_\tau}\frac{\partial \tilde\nabla_i}{\partial T} \tag{21}$$

$$= \frac{1}{2}\left(\frac{(\tilde\nabla R_i)^\top}{\|\tilde\nabla T_i\|_\tau \|\tilde\nabla R_i\|_\varrho}\frac{\partial \tilde\nabla_i}{\partial T} - \frac{\frac{1}{2}\langle \tilde\nabla T_i, \tilde\nabla R_i\rangle + \tau\varrho}{\|\tilde\nabla T_i\|_\tau^3 \|\tilde\nabla R_i\|_\varrho}(\tilde\nabla T_i)^\top \frac{\partial \tilde\nabla_i}{\partial T}\right). \tag{22}$$

We introduce the abbreviation

$$\rho_i(k) := \frac{-R_i + R_{i+k}}{\|\tilde\nabla T_i\|_\tau \|\tilde\nabla R_i\|_\varrho} - \frac{\left(\frac{1}{2}\langle \tilde\nabla T_i, \tilde\nabla R_i\rangle + \tau\varrho\right)(-T_i + T_{i+k})}{\|\tilde\nabla T_i\|_\tau^3 \|\tilde\nabla R_i\|_\varrho} \tag{23}$$

and the set of indices of nonzero elements

$$\mathcal{K} := \{-z, -y, -x,\ 0, x, y, z\}. \tag{24}$$

Furthermore, define $\hat h_k := \frac{1}{2\left(h_{|k|}\right)^2}$ and

$$\hat\rho_i(k) := \begin{cases} \sum_{j\in\mathcal{K}\setminus\{0\}} -\hat h_j \rho_i(j) & \text{if } k = 0,\\ \hat h_k \rho_i(k) & \text{otherwise.}\end{cases} \tag{25}$$

Then, (22) can be compactly written as

$$\frac{\partial r_i}{\partial T} = \left(\begin{array}{ccccccc} \scriptstyle i-z & \scriptstyle i-y & \scriptstyle i-x & \scriptstyle i & \scriptstyle i+x & \scriptstyle i+y & \scriptstyle i+z \\ \hat\rho_i(-z) & \hat\rho_i(-y) & \hat\rho_i(-x) & \hat\rho_i(0) & \hat\rho_i(x) & \hat\rho_i(y) & \hat\rho_i(z)\end{array}\right) \in \mathbb{R}^{1\times\bar m}, \tag{26}$$

where only nonzero elements are shown and single element locations are denoted above the vector. It can be seen that (26) exhibits a very sparse pattern with only seven nonzero elements.

For a full gradient computation as in (16), it remains to consider the terms $\frac{\partial T}{\partial P}$ and $\frac{\partial P}{\partial \mathbf{y}}$. The first term represents the derivative of the template image interpolation function with respect to the image grid coordinates. Given the lexicographical ordering of the grid points, the image derivative matrix is composed of three diagonal matrices,

$$\frac{\partial T}{\partial P} = \left(\operatorname{diag}\left(\tfrac{\partial T_1}{\partial P_1}\cdots\tfrac{\partial T_{\bar m}}{\partial P_{\bar m}}\right), \operatorname{diag}\left(\tfrac{\partial T_1}{\partial P_{\bar m+1}}\cdots\tfrac{\partial T_{\bar m}}{\partial P_{2\bar m}}\right), \operatorname{diag}\left(\tfrac{\partial T_1}{\partial P_{2\bar m+1}}\cdots\tfrac{\partial T_{\bar m}}{\partial P_{3\bar m}}\right)\right), \tag{27}$$

where each nonzero element represents the derivative at a single point with respect to a single coordinate.

The last remaining term $\frac{\partial P}{\partial \mathbf{y}}$ is the Jacobian of the grid conversion function. This is a linear operator, which will be analyzed in detail in section 4.3.

From the derivations in (18), (26), and (27), it can be seen that the main components of the NGF gradient computation exhibit a sparse structure with fixed patterns. The main effort comes from computing the derivative of r, which has a seven-banded diagonal structure.

Algorithm 1 Pseudocode for matrix-free computation of the elements of the NGF gradient as in (28). The algorithm can be fully parallelized over all loop iterations.

```
for $\hat{k}$ in $[0, m_z - 1]$ do
    for $\hat{j}$ in $[0, m_y - 1]$ do
        for $\hat{i}$ in $[0, m_x - 1]$ do
            $i \leftarrow \hat{i} + m_x\hat{j} + m_x m_y \hat{k}$                ▷ Compute linear index
            $i \pm x, i \pm y, i \pm z \leftarrow$ as in (4)            ▷ Compute neighbor indices
            [dTx, dTy, dTz] $\leftarrow$ imageDerivative($T_i$)   ▷ Compute image derivative

            r $\leftarrow [r_{i-z}, r_{i-y}, r_{i-x}, r_i, r_{i+x}, r_{i+y}, r_{i+z}]$        ▷ $r_i$ as defined in (19)
            dr $\leftarrow [\hat{\rho}_{i-z}(z), \hat{\rho}_{i-y}(y), \hat{\rho}_{i-x}(x), \hat{\rho}_i(0), \hat{\rho}_{i+x}(-x), \hat{\rho}_{i+y}(-y), \hat{\rho}_{i+z}(-z)]$
                                                                   ▷ $\hat{\rho}_i$ as defined in (25)
            drSum $\leftarrow -2\bar{h}$ (r[0]dr[0] + r[1]dr[1] + r[2]dr[2] + r[3]dr[3]
                         + r[4]dr[4] + r[5]dr[5] + r[6]dr[6])
                                                                   ▷ Compute sum of (28)
            grad[$i$      ] $\leftarrow$ drSum $\cdot$ dTx
            grad[$i + \bar{m}$] $\leftarrow$ drSum $\cdot$ dTy
            grad[$i + 2\bar{m}$] $\leftarrow$ drSum $\cdot$ dTz
        end for
    end for
end for
Output: grad[$i + d\bar{m}$] = $\left(\frac{\partial D}{\partial P}\right)_{i+d\bar{m}}$
```

Exploiting this pattern, a single element of the partial gradient $\frac{\partial D}{\partial P} := \frac{\partial \psi}{\partial r}\frac{\partial r}{\partial T}\frac{\partial T}{\partial P} \in \mathbb{R}^{1\times 3\bar{m}}$ can be explicitly computed as

$$\left(\frac{\partial D}{\partial P}\right)_{i+d\bar{m}} = -2\,\bar{h}\left(\sum_{k\in\mathcal{K}} r_{i+k}\hat{\rho}_{i+k}(-k)\right)\frac{\partial T_i}{\partial P_{i+d\bar{m}}}, \tag{28}$$

where $d \in \{0,1,2\}$ and $\mathcal{K} = \{-z,-y,-x,\ 0,x,y,z\}$ as in (24). Algorithm 1 shows a pseudocode implementation for computing the individual elements of the gradient as in (28).

This formulation has multiple benefits: Any element of the gradient can be computed directly from the input data, without the need for (sparse) matrices and without having to store intermediate results. Furthermore, gradient elements can be computed independently, which allows for a fully parallel implementation. As will be discussed in section 5, both of these properties substantially decrease memory usage and computation time.

4.1.2. Hessian-vector multiplication. As noted in section 2.3.1, when using the Gauss–Newton scheme for optimization, linear systems involving the quadratic approximation $H \in \mathbb{R}^{3\bar{m}^{y}\times 3\bar{m}^{y}}$ of the Hessian (17) need to be solved in each iteration. Although H is sparse, the memory requirements for storing the final as well as intermediate matrices can still be considerable. Therefore, in the following we will consider an efficient scheme for evaluating the matrix-vector product $H\mathbf{p} = \mathbf{q}$ with $\mathbf{p}, \mathbf{q} \in \mathbb{R}^{3\bar{m}^{y}}$, which is the foundation for using iterative methods, such as conjugate gradients, in order to solve the (Gauss–)Newton equation (9).

Figure 2 shows a schematic of the computations involved. Again, we consider the Jacobian $\frac{\partial P}{\partial \mathbf{y}}$ and its transpose in (17) as separate grid conversion steps, which will be discussed in detail in section 4.3. The main computations consist of computing the matrix product

$$\hat{H} := \frac{\partial T}{\partial P}^{\top} \frac{\partial r}{\partial T}^{\top} \frac{\partial r}{\partial T} \frac{\partial T}{\partial P} \in \mathbb{R}^{3\bar{m} \times 3\bar{m}}, \tag{29}$$

with the components $\frac{\partial T}{\partial P} \in \mathbb{R}^{\bar{m} \times 3\bar{m}}$, $\frac{\partial r}{\partial T} \in \mathbb{R}^{\bar{m} \times \bar{m}}$, which is equivalent to computing the approximate Hessian H in (17) with the exception of the grid conversion steps.

We first analyze the matrix-vector product $\hat{H}\hat{\mathbf{p}} = \hat{\mathbf{q}}$, with $\hat{\mathbf{p}}, \hat{\mathbf{q}} \in \mathbb{R}^{3\bar{m}}$. Abbreviating $dr := \frac{\partial r}{\partial T}$, the main challenge is efficiently computing the matrix product $dr^{\top} dr$.

Using the definition and notation of a single *row* of dr given in (26), a single *column* of the matrix can be written as

(30)

$$\frac{\partial r}{\partial T_i} = \begin{array}{c} \\ \Big(\end{array} \begin{array}{ccccccc} \scriptstyle i-z & \scriptstyle i-y & \scriptstyle i-x & \scriptstyle i & \scriptstyle i+x & \scriptstyle i+y & \scriptstyle i+z \\ \hat{\rho}_{i-z}(z) & \hat{\rho}_{i-y}(y) & \hat{\rho}_{i-x}(x) & \hat{\rho}_i(0) & \hat{\rho}_{i+x}(-x) & \hat{\rho}_{i+y}(-y) & \hat{\rho}_{i+z}(-z) \end{array} \begin{array}{c} \\ \Big)^{\top}, \end{array}$$

again only showing nonzero elements at the indices denoted above the vector. As in (26), a single column contains only seven nonzero elements.

Abbreviating $dr_i := \frac{\partial r}{\partial T_i} \in \mathbb{R}^{\bar{m} \times 1}$, each element of the matrix $dr^{\top} dr$ is a scalar product of columns $\langle dr_i, dr_j \rangle$ for $i, j = 1, \ldots, \bar{m}$. However, due to the sparsity of the columns of dr in (30), there are only a few nonzero scalar products. As a basic example, consider the diagonal elements of $dr^{\top} dr$. In column dr_i, nonzero elements are located at $(dr_i)_{i+k}$, $k \in \mathcal{K}$, with $\mathcal{K}$ as in (24). This yields the scalar products $\langle dr_i, dr_i \rangle = \sum_{k \in \mathcal{K}} (dr_i)_{i+k}^2 = \sum_{k \in \mathcal{K}} \hat{\rho}_{i+k}(-k)^2$, a sum of seven terms.

More generally, for arbitrary i, j, define $\kappa := j - i$, so that $\langle dr_i, dr_j \rangle = \langle dr_i, dr_{i+\kappa} \rangle$. In order to characterize the elements where $\langle dr_i, dr_{i+\kappa} \rangle$ can be nonzero, observe that the nonzero elements of dr_i are located at indices $i + \mathcal{M}$,

$$\mathcal{M} := \{-m_x m_y, -m_x, -1,\ 0, 1, m_x, m_x m_y\}. \tag{31}$$

Consequently, $\langle dr_i, dr_{i+\kappa} \rangle$ can only be nonzero if $(i + \mathcal{M}) \cap (i + \kappa + \mathcal{M}) \neq \emptyset$, i.e., if the scalar product involves at least two nonzero elements. Equivalently, $\kappa \in \mathcal{N} := \mathcal{M} - \mathcal{M}$, and therefore the number of nonzero inner products is bounded from above by the number of elements $|\mathcal{N}|$ in $\mathcal{N}$. Naturally, $|\mathcal{N}| \leq |\mathcal{M}| \cdot |\mathcal{M}| = 49$; however, it turns out that this bound can be substantially lowered.

In order to do so, define the mapping

$$N(i, j) := j - i, \qquad N : \mathcal{M} \times \mathcal{M} \to \mathbb{Z},$$

so that

$$\mathcal{N} = N(\mathcal{M}, \mathcal{M}) = \Big\{ \kappa \in \mathbb{Z} \ \Big|\ \kappa = N(i, j),\ i, j \in \mathcal{M} \Big\}.$$

Inspecting all possible combinations for i and j, it turns out that $|\mathcal{N}| = 25$ (Table 2), which implies that each column of $dr^{\top} dr$ has at most 25 nonzero elements.

For a given offset $\kappa = j - i$, the preimage $N^{-1}(\kappa)$ indicates the indices of nonzero elements contributing to the inner product $\langle dr_i, dr_j \rangle$. As shown in the rightmost column in Table 2, for the main diagonal of $dr^{\top} dr$, seven elements are involved in the

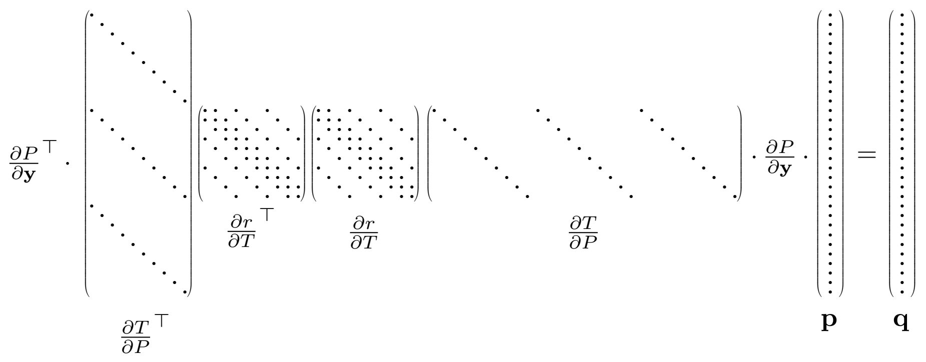

$\frac{\partial P}{\partial \mathbf{y}}^\top \cdot$
$\frac{\partial T}{\partial P}^\top$
$\frac{\partial r}{\partial T}^\top$
$\frac{\partial r}{\partial T}$
$\frac{\partial T}{\partial P}$
$\cdot \frac{\partial P}{\partial \mathbf{y}} \cdot$
$=$
$\mathbf{p}$
$\mathbf{q}$

FIG. 2. *Schematic view of Hessian-vector multiplication $H\mathbf{p} = \mathbf{q}$. The computation involves highly sparse matrices with a fixed pattern of nonzeros. The main computational effort results from the matrix multiplication $\frac{\partial r}{\partial T}^\top \frac{\partial r}{\partial T}$, where $\frac{\partial r}{\partial T}$ has seven nonzero diagonals.*

scalar product, while for all other elements only either one or two products need to be computed. Using $\kappa = j - i$ and $(\hat{i}, \hat{j}) \in N^{-1}(\kappa)$, it holds that

$$\begin{aligned} \left(dr^\top dr\right)_{i,j} &= \begin{cases} \sum_{(\hat{i},\hat{j}) \in N^{-1}(j-i)} \left(dr_i\right)_{i+\hat{i}} \left(dr_j\right)_{j+\hat{j}} & \text{if } j - i \in \mathcal{N}, \\ 0 & \text{otherwise,} \end{cases} \\ &= \begin{cases} \sum_{(\hat{i},\hat{j}) \in N^{-1}(j-i)} \hat{\rho}_{i+\hat{i}}(-\hat{i}) \; \hat{\rho}_{j+\hat{j}}(-\hat{j}) & \text{if } j - i \in \mathcal{N}, \\ 0 & \text{otherwise,} \end{cases} \end{aligned} \tag{32}$$

where, in a slight abuse of notation, $\hat{\rho}(-m_1 m_2)$ should be interpreted as $\hat{\rho}(-z)$, etc.

With this and the explicit formulation of $\hat{\rho}_i(k)$ in (23) and (25), the computational cost of calculating $dr^\top dr$ can be reduced substantially: multiplications involving zero elements are no longer considered, administrative costs for locating and handling sparse matrix elements have been eliminated, and only essential calculations remain.

Substituting (32) into the expression (29) for evaluating the matrix-vector product $\hat{\mathbf{q}} = \hat{H}\hat{\mathbf{p}}$, we obtain

$$\hat{\mathrm{q}}_{d\bar{m}+i} = \sum_{\kappa \in \mathcal{N}} \sum_{l \in \{0,1,2\}} \frac{\partial T_i}{\partial P_{d\bar{m}+i}} \left(dr^\top dr\right)_{i,i+\kappa} \frac{\partial T_\kappa}{\partial P_{l\bar{m}+i+\kappa}} \; \hat{\mathrm{p}}_{l\bar{m}+i+\kappa} \tag{33}$$

for $d \in \{0, 1, 2\}$, using the definition of $\left(dr^\top dr\right)_{i,i+\kappa}$ from (32).

Similar to the matrix-free gradient formulation in (28), this allows a direct computation of each element of the result vector $\hat{\mathbf{q}} = \hat{H}\hat{\mathbf{p}}$ fully in parallel and directly from the input data. An implementation in pseudocode is shown in Algorithm 2.

4.2. Derivative computations for curvature regularization. In (11), a matrix-free version of the curvature gradient computation was already given. As the curvature computation is a quadratic function, the Hessian-vector multiplication

$$\mathrm{q}_{i+d\bar{m}^{\mathrm{y}}} = \left(\nabla^2 S\mathbf{p}\right)_{i+d\bar{m}^{\mathrm{y}}} = 2\bar{h}^{\mathrm{y}} \tilde{\Delta} \left(\tilde{\Delta}\mathbf{p}\right)_{i+d\bar{m}^{\mathrm{y}}}, \tag{34}$$

for $d \in \{0, 1, 2\}$, is closely related, with $\mathbf{p}$ replacing $\mathbf{u}$ in (11). In contrast to NGF, this uses the exact Hessian rather than a Gauss–Newton approximation. As the regularizer operates on the deformation grid, no grid conversion is required.

TABLE 2

Offsets $\kappa := j - i$, for which $\langle dr_i, dr_j \rangle \neq 0$, corresponding to nonzero elements $(dr^\top dr)_{i,j}$ in the matrix-product $dr^\top dr$. The preimage sets $N^{-1}(\kappa)$ characterize the locations of the nonzero element in dr_i and dr_j. The 11 omitted cases are identical to the ones shown except for opposite sign. The rightmost column shows the number of products involving nonzero coefficients in the evaluation of $\langle dr_i, dr_j \rangle$ and sums to $|\mathcal{M}| \cdot |\mathcal{M}| = 49$.

$\kappa \in \mathcal{N}$	$N^{-1}(\kappa)$		$\lvert N^{-1}(\kappa)\rvert$
$-2m_1m_2$	$\{(m_1m_2, -m_1m_2)\}$		1
$-m_1m_2 - m_1$	$\{(m_1, -m_1m_2),$	$(m_1m_2, -m_1)\}$	2
$-m_1m_2 - 1$	$\{(1, -m_1m_2),$	$(m_1m_2, -1)\}$	2
$-m_1m_2$	$\{(0, -m_1m_2),$	$(m_1m_2, 0)\}$	2
$-m_1m_2 + 1$	$\{(-1, -m_1m_2),$	$(m_1m_2, 1)\}$	2
$-m_1m_2 + m_1$	$\{(-m_1, -m_1m_2),$	$(m_1m_2, m_1)\}$	2
$-2m_1$	$\{(m_1, -m_1)$		1
$-m_1 - 1$	$\{(1, -m_1),$	$(m_1, -1)\}$	2
$-m_1$	$\{(0, -m_1),$	$(m_1, 0)\}$	2
$-m_1 + 1$	$\{(-1, -m_1),$	$(m_1, 1)\}$	2
-2	$\{(1, -1)\}$		1
-1	$\{(0, -1),$	$(1, 0)\}$	2
0	$\{(\hat{i}, \hat{i}) \mid \hat{i} \in \mathcal{M}\}$		7
1	$\{(0, 1),$	$(-1, 0)\}$	2
$\vdots$	$\vdots$	$\vdots$	$\vdots$

4.3. Grid conversion. We now consider the grid conversion steps omitted in the previous sections. The grid conversion maps deformation values $\mathbf{y} \in \mathbb{R}^{3\bar{m}^y}$ from the deformation grid to the image grid using the function $P : \mathbb{R}^{3\bar{m}^y} \to \mathbb{R}^{3\bar{m}}$ (section 2.2). In this section, we separately analyze P and its transpose $P^\top$.

The conversion, i.e., interpolation, of $\mathbf{y}$ from deformation grid to image grid is performed using trilinear interpolation and thus can be written as a matrix-vector product

$$\hat{\mathbf{y}} = P\mathbf{y}, \quad \text{with } P \in \mathbb{R}^{3\bar{m} \times 3\bar{m}^y} \quad \text{and} \quad \mathbf{y} \in \mathbb{R}^{3\bar{m}^y},\ \hat{\mathbf{y}} \in \mathbb{R}^{3\bar{m}}.$$

The transformation matrix P has a block-diagonal structure with three identical blocks, each converting one of the three coordinate components of $\mathbf{y}$. As the deformation grid is a nodal grid, no boundary handling is required, as each (cell-centered) image grid point is surrounded by deformation grid points.

In order to get closed-form expressions for the nonzero entries of P, define the function $c_l(k) := \frac{(k+0.5)m_l^y}{m_l}$, which converts a coordinate index from deformation to image grid, and the remainder function $r_l(k) := c_l(k) - \lfloor c_l(k) \rfloor$. Then, for the point at coordinates $(\hat{i}, \hat{j}, \hat{k})$ with $i = \hat{i} + \hat{j}m_x + \hat{k}m_xm_y$ and $d \in \{0, 1, 2\}$, the interpolated deformation value can be written as

$$(P\mathbf{y})_{\hat{i}+\hat{j}m_x+\hat{k}m_xm_y+d\bar{m}} = \sum_{\alpha=0}^{1}\sum_{\beta=0}^{1}\sum_{\gamma=0}^{1} w^{\alpha,\beta,\gamma}(\hat{i}, \hat{j}, \hat{k})\ \mathrm{y}_{\xi(\alpha,\beta,\gamma,\hat{i},\hat{j},\hat{k},d)}, \tag{35}$$

where the indices of the neighboring deformation grid points are

(36)
$$\xi(\alpha, \beta, \gamma, \hat{i}, \hat{j}, \hat{k}, d) := \Big(\lfloor c_x(\hat{i}) \rfloor + \alpha\Big) + m_x \Big(\lfloor c_y(\hat{j}) \rfloor + \beta\Big) + m_xm_y \Big(\lfloor c_z(\hat{k}) \rfloor + \gamma\Big) + d\bar{m}$$

Algorithm 2 Pseudocode for matrix-free computation of the elements of the result vector $\hat{\mathbf{q}}$ in the NGF Hessian-vector multiplication $\hat{H}\hat{\mathbf{p}} = \hat{\mathbf{q}}$ as in (29). The algorithm can be fully parallelized over all loop iterations of $\hat{i}, \hat{j}, \hat{k}$, computing three elements of the result per thread.

```
N ← $[-2m_1m_2, -m_1m_2-m_1, -m_1m_2-1, -m_1m_2, -m_1m_2+1, -m_1m_2+m_1, -2m_1, -m_1-1, -m_1,$
        $-m_1+1, -2, -1, 0, 1, 2, m_1-1, m_1, m_1+1, 2m_1, m_1m_2-m_1, m_1m_2-1, m_1m_2, m_1m_2+1$
        $m_1m_2+m_1, 2m_1m_2]$                        ▷ Initialize 25 indices in N; see Table 2
q ← 0
for $\hat{k}$ in $[0, m_z-1]$ do
    for $\hat{j}$ in $[0, m_y-1]$ do
        for $\hat{i}$ in $[0, m_x-1]$ do
            $i \leftarrow \hat{i} + m_x\hat{j} + m_xm_y\hat{k}$                    ▷ Compute linear index
            $i \pm x, i \pm y, i \pm z \leftarrow$ as in (4)                 ▷ Compute neighbor indices
            [dTx, dTy, dTz] ← imageDerivative($T_{i+\mathtt{N}}$)
                                            ▷ Compute image derivatives at 25 points
            for k in $[0, 24]$ do                   ▷ Compute 25 values of $dr^\top dr$; see (33)
                drdr ← 0

                for $(\tilde{i}, \tilde{j})$ in $N^{-1}$(N[k]) do
                                         ▷ Compute 49 values of $\hat{\rho}_i$; see (32) and Table 2
                    drdr ← drdr + $\hat{\rho}_{i+\tilde{i}}(-\tilde{i}) \cdot \hat{\rho}_{i+\mathtt{N[k]}+\tilde{j}}(-\tilde{j})$
                end for

                q[$i$       ] ← q[$i$       ] + dTx[12] · drdr · dTx[k] · $\hat{\mathrm{p}}_{i+\mathtt{N[k]}}$
                q[$i + \bar{m}$] ← q[$i + \bar{m}$] + dTy[12] · drdr · dTy[k] · $\hat{\mathrm{p}}_{i+\mathtt{N[k]}+\bar{m}}$
                q[$i + 2\bar{m}$] ← q[$i + 2\bar{m}$] + dTz[12] · drdr · dTz[k] · $\hat{\mathrm{p}}_{i+\mathtt{N[k]}+2\bar{m}}$
                                ▷ The 12th element corresponds to the derivative at index $i$
            end for
        end for
    end for
end for
Output: q[$i + d\bar{m}$] = $\hat{\mathrm{q}}_{d\bar{m}+i}$
```

and the interpolation weights are given by

$$w^{\alpha,\beta,\gamma}(\hat{i},\hat{j},\hat{k}) := \hat{w}_x^\alpha(\hat{i})\hat{w}_y^\beta(\hat{j})\hat{w}_z^\gamma(\hat{k}), \quad \hat{w}_l^s(k) := \begin{cases} 1 - r_l(k) & \text{if } s = 0, \\ r_l(k) & \text{otherwise.} \end{cases}$$

As can be seen in (35), a single interpolated deformation value on the image grid is simply a weighted sum of the values of its eight neighbors on the deformation grid; see also Figure 3(a).

For the NGF derivative computations in (16) and (17), the left-sided multiplication $\hat{\mathbf{y}}^\top P = (P^\top \hat{\mathbf{y}})^\top$ is also required in order to evaluate the transposed operator. For each (nodal) deformation grid point, this amounts to a weighted sum of values from all image grid points in the adjacent deformation grid cells, which can be many more than eight (Figure 3(b)).

By singling out the contribution of all image grid points within a single deformation grid cell as in Figure 3(c), the weights $\hat{w}_l^s(k)$ can be reused and only need to be computed once for each deformation grid cell. This suggests parallelizing the

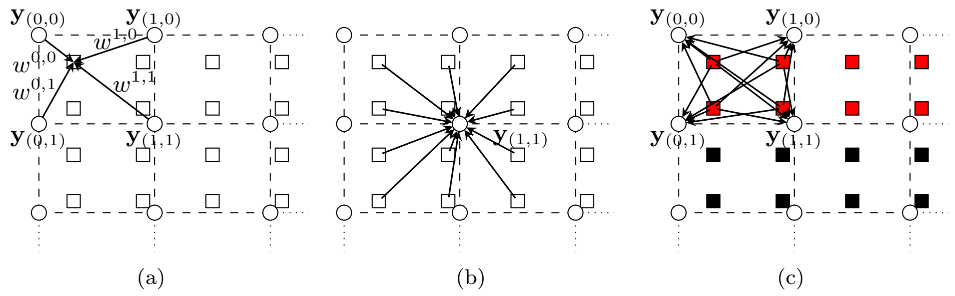


FIG. 3. *Grid conversion operations (2D example).* Circles: *nodal deformation grid;* squares: *cell-centered image grid.* (a) *Deformation grid to image grid: each value on the image grid is computed by a weighted sum of the values of its deformation grid neighbors.* (b) *Transposed operation: each value on the deformation grid is a weighted sum of all image grid values from neighboring deformation grid cells.* (c) *Transposed operation using the proposed red-black scheme. Weighted values on the image grid are accumulated into all surrounding deformation grid points using multiple write accesses. Weights need to be computed only once per cell. Parallelization is performed per row (2D) or slice (3D) using a red-black scheme to avoid write conflicts.*

computations on a per-cell basis. However, as a deformation grid point has multiple neighboring grid cells, this would lead to write conflicts.

Therefore, in order to achieve both efficient reuse of weights and parallelizability, we propose using a red-black scheme as shown in Figure 3(c). Each parallel unit sequentially processes a single two-dimensional (2D) slice of deformation grid cells, first all odd (red) and finally all even (black) slices.

4.4. Extension to related models. By carefully examining the derivative structure of the NGF distance measure and curvature regularizer, we have derived closed-form expressions for computing individual elements of the gradients as well as Hessian matrix-vector products. The new formulations operate directly on the input data, avoid storage of intermediate results (and thus costly memory accesses), and allow for fully parallel execution.

It is important to note that the proposed approach is by no means conceptually restricted to the NGF distance measure or curvature regularization. The central idea of replacing sparse matrix construction by on-the-fly coefficient calculation for derivative computation is in principle applicable to all distance measures and regularization schemes. The effectiveness, however, strongly depends on the derivative structure of the considered objective function terms.

In more detail, consider an arbitrary distance measure $D = \psi(r(T(P)))$. We first observe that the proposed reformulation for the grid conversion operations related to P as well as the image interpolation calculations for T are independent of the choice of the distance metric. Hence, any matrix-free reformulation for these functions can be applied to every choice of distance measure D. Since the derivative of ψ is just a vector, the most critical part for a matrix-free computation is the residual function derivative $\frac{\partial r}{\partial T}$. For NGF, the nonzero elements are distributed to just seven diagonals, thus allowing us to derive a closed expression for $\frac{\partial r}{\partial T}$. In the case of the well-known sum-of-squared-differences (SSD) distance measure, the derivative $\frac{\partial r}{\partial T}$ is even the identity and can thus be omitted from the computations [44].

More generally, matrix-free methods may be considered practically applicable as long as the sparsity pattern of $\frac{\partial r}{\partial T}$ is known and fixed. Examples of suitable matrix types are band matrices, block structures, and similar configurations. Unfortunately, the widely used mutual information distance measure [10, 51] does not allow for such application of the proposed matrix-free concept: here, the sparsity pattern of $\frac{\partial r}{\partial T}$ depends on the image intensity distribution of the deformed template image and thus on both T and the current deformation $\mathbf{y}$ [36]. As the deformation changes at each iteration step, the sparsity pattern of $\frac{\partial r}{\partial T}$ may do so as well, prohibiting the derivation of static matrix-free calculation rules. Again, however, the matrix-free grid change schemes and the discussed template image derivative computations can directly be reused also for mutual information.

For the regularization term, we note that widely used energies such as the diffusive regularizer [13] or the linear-elastic potential [35] are essentially calculated using finite differences schemes on the deformation $\mathbf{y}$. Hence, their matrix-free computation is far less complex than for the distance term with its function chain structure, and similar techniques may directly be used, as done for the curvature regularization in this work. Also, more complex regularization schemes such as hyperelastic regularization [5], which penalizes local volume change and generates diffeomorphic transformations, may be similarly implemented in a matrix-free manner.

In summary, the underlying idea of our work is widely applicable and not restricted to the exemplary use case of NGF with curvature regularization. An interesting question is what performance improvements can be expected for conceptually different methods such as LDDMM [32]; we refer the reader to [33] for some notes on parallelizing such methods.

5. Algorithm analysis.

In this section, we will perform a theoretical analysis and quantification of the expected speedup obtained by using the proposed matrix-free approach, compared to a matrix-based approach as used in [36].

5.1. Matrix element computations and memory stores.

While the exact cost of sparse matrix multiplications is hard to quantify and highly depends on the implementation and chosen sparse matrix format, meaningful estimates can be made for the cost of matrix coefficient computations and memory stores. As in this section we are primarily concerned with the *time* consumed by memory write accesses, by "memory store" we refer to a single memory write operation. The *amount* of memory required by each scheme is discussed in section 5.2.

5.1.1. Matrix-based approach.

Data term. For the NGF gradient, analyzing the structure of $\frac{\partial \psi}{\partial r}$, $\frac{\partial r}{\partial T}$, and $\frac{\partial T}{\partial P}$ in (18), (26), and (27), an upper bound for the required number of nonzero element computations can be computed.

The vector $\frac{\partial \psi}{\partial r}$ requires the computation of $\bar{m}$ residual elements, where $\bar{m}$ is the number of cells in the image grid (section 2.2). For the matrix $\frac{\partial r}{\partial T}$, with a seven-banded structure, at most $7\bar{m}$ elements need to be computed, while $\frac{\partial T}{\partial P}$ contains at most $3\bar{m}$ nonzero matrix elements.

The grid conversion, discussed in section 4.3, involves a matrix with eight coefficients in each row, one per contributing deformation grid point (35), resulting in the computation of $8\bar{m}$ coefficients. In total, this gives $19\bar{m}$ coefficient computations for the NGF term in the matrix-based scheme. As the matrix elements only depend on the size of the image and deformation grids, they can be computed once and stored for each resolution.

Regularizer. The curvature regularizer is a quadratic function and can be implemented using a matrix with 25 diagonals, resulting in $25\bar{m}^{\mathrm{y}}$ memory stores, where $\bar{m}^{\mathrm{y}}$

is the number of cells in the deformation grid.

Hessian. For the Gauss–Newton Hessian-vector multiplication, no additional coefficient computations are needed, as the Hessian is approximated from first-order derivatives.

Total number of operations. After computing all matrix elements, these have to be stored in memory for later use, resulting in $19\bar{m} + 25\bar{m}^{\mathrm{y}}$ memory store operations for one evaluation of the full gradient and Hessian approximation. This assumes that the latter will never be explicitly stored, as discussed in the following section. If multiple matrix-vector multiplications with the same gradient are required, as in the inner CG iterations, the factor matrices can be reused and no new memory stores are necessary.

5.1.2. Matrix-free approach.

Data term. For the matrix-free formulation of the NGF gradient in (28), excluding grid conversion, the same $11\bar{m}$ coefficient computations as in the matrix-based case have to be performed for one evaluation of the gradient. However, no intermediate storage of elements is needed.

For the grid conversion, the coefficients are recomputed during each gradient evaluation and Hessian-vector multiplication. Furthermore, they are recomputed during the application of the transposed operator. With $8\bar{m}$ computations in each grid conversion, this adds up to $16\bar{m}$ additional coefficient computations for application of the forward and transposed operator compared to the matrix-based case.

In the proposed approach, the grid conversions are evaluated separately from the remaining derivatives, instead of fully integrating them into the matrix-free computations. While the latter further lowers memory requirements, it requires recomputations of weights even within a single conversion, which in our experience resulted in slower overall computation times.

Regularizer. For the curvature regularizer, the coefficients are highly redundant, resulting from the finite differences stencil. Therefore, there is no computational cost for coefficients in the matrix-free scheme.

Hessian. For the matrix-free Gauss–Newton Hessian-vector multiplication, a number of elements are recomputed. As shown in (33) and (32), coefficients $\hat{\rho}_i(k)$ are computed $2 \cdot 49\bar{m}$ times for one Hessian-vector multiplication. With 25 diagonals in $dr^\top dr$, image derivatives from $\frac{\partial T}{\partial P}$ are required for 25 points, resulting in additional $25\bar{m}$ coefficient computations. In total, the Hessian-vector multiplication requires $123\bar{m}$ additional coefficient computations compared to the matrix-based method.

Note that, in the matrix-based case, we have assumed that the full Hessian is never stored but rather evaluated using its factor matrices (Figure 2). If the final Hessian is also saved, an additional $9 \cdot 25\bar{m}$ memory stores are required, as $dr^\top dr$ exhibits 25 diagonals.

While in both schemes the gradient is fully stored once computed and can be reused, for the matrix-free Hessian-vector multiplication, all elements need to be recomputed for every single matrix-vector multiplication. Therefore, the exact number of additional arithmetic operations depends on other variables, such as the number of CG iterations and line-search steps, and is specific to the input data.

Total number of operations. In summary, for matrix-free NGF gradient evaluations, there are no additional coefficient computations compared to the matrix-based scheme and all intermediate memory stores are completely eliminated, making it highly amenable to massive parallelization.

For the Gauss–Newton Hessian-vector multiplications, the matrix-free scheme adds a considerable number of recomputations, but again no stores are required. However, memory store operations are generally orders of magnitude slower than arithmetic operations. Therefore, as will be shown in the next section, in practice this trade-off between computations and memory stores results in faster overall runtimes. Moreover, it significantly reduces the amount of required memory, which is a severely limiting factor for Hessian-based methods.

5.2. Memory requirements. As discussed in the previous section, the memory required for the derivative matrices in a classical matrix-based scheme depends on the image grid size $\bar{m}$. For the gradient computations, when storing $\frac{\partial \psi}{\partial P}$, $\frac{\partial r}{\partial T}$, and $\frac{\partial T}{\partial P}$, $\bar{m}$, $7\bar{m}$, and $3\bar{m}$ values need to be saved. For the grid conversion, $8\bar{m}$ values need to be stored, while the curvature regularizer requires $25\bar{m}^{\mathrm{y}}$ values. In terms of complexity, this leads to memory requirements in the order of $\mathcal{O}(\bar{m})$.

In comparison, the memory required by the matrix-free schemes does not depend on the image grid size. When integrating the grid conversion steps directly into the computation, as described in the previous section, only constant auxiliary space is required. The matrix-free approach therefore reduces the required memory size for intermediate storage from $\mathcal{O}(\bar{m})$ to $\mathcal{O}(1)$.

6. Experiments and results. In order to characterize the performance at the implementation level, we performed a range of experiments comparing four approaches for derivative computation. Three approaches are implemented on the CPU: an open-source, mixed C and MATLAB implementation in the FAIR toolbox [36] ("FAIR"), an in-house C++ implementation of the matrix-based computations using sparse matrices ("MBC"), and a C++ implementation of the proposed matrix-free computations ("MFC"). Additionally, we evaluated a GPU implementation of the matrix-free computations using NVIDIA CUDA ("MFC$^{\text{GPU}}$").

All CPU implementations use OpenMP for parallelization of critical components. In FAIR and MBC, these include distance measure computations, linear interpolation, and sparse-matrix multiplications. FAIR provides a "matrix-free" mode, which uses matrix-free computations for the curvature regularizer. In contrast to our approach, all other derivative matrices are still built explicitly and require additional storage and memory accesses. This mode was used for all comparisons. Furthermore, MATLAB-MEX versions of FAIR components were used where available. In the matrix-free method MFC, full elementwise parallelization was used for function value and derivative calculations.

Both the classical MBC and the proposed MFC implementations have undergone thorough optimization. In addition, due to the favorable structure, the NGF gradient computations for MFC on the CPU were manually vectorized using the Intel Advanced Vector Instructions (AVX). As stated in section 4.3, the grid conversions were computed as separate steps. Additionally, the deformed template was temporarily stored in order to reduce interpolation calls.

The experiments on the CPU were performed on a dual processor 12-core Intel Xeon E5-2620 workstation with 2.0 GHz and 32 GB RAM. The GPU computations were performed on a NVIDIA GeForce GTX980; see section 6.3 for more detailed specifications. All experiments were averaged over 30 runs with identical input in order to reduce measurement noise.

6.1. Scalability. The proposed matrix-free derivative calculations permit a fully parallel execution with only a single reduction for the function value, leading to high arithmetic intensity, i.e., floating point operations per byte of memory transfer. Thus,

TABLE 3

Scaling of the proposed parallel implementation on a 12-core workstation compared to serial computation. Runtimes and speedups are itemized for gradient evaluations, Hessian-vector multiplications, and grid conversions. All operations scale almost linearly, with the exception of the transposed grid conversion operator $P^\top \hat{\mathbf{y}}$ on small images, where the number of available parallel tasks is too low, and the curvature regularizer, which is primarily memory-bound. Small images: 64^3 image resolution, 17^3 deformation grid size; large images: 512^3 image resolution, 129^3 deformation grid size.

	Small images			Large images		
Method	Serial (ms)	Par. (ms)	Speedup	Serial (s)	Par. (s)	Speedup
D	65.6	5.94	**11.0**	44.6	3.31	**13.5**
$\frac{\partial D}{\partial P}$	131	10.6	**12.4**	72.9	5.40	**13.5**
$P\mathbf{y}$	8.95	0.743	**12.1**	4.51	0.355	**12.7**
$P^\top \hat{\mathbf{y}}$	9.91	2.27	4.34	7.14	0.58	**12.4**
$\hat{H}\hat{\mathbf{p}}$	1450	117	**12.4**	913	61.1	**15.0**
S	0.101	0.0655	1.54	0.0553	0.0167	3.31
∇S	0.226	0.0411	5.51	0.110	0.0428	2.58
$\nabla^2 S\mathbf{p}$	0.224	0.0450	4.98	0.109	0.0371	2.94

excellent scalability with respect to the number of computational cores can be expected.

In order to experimentally support this claim, we computed objective function derivatives using a varying number of computational cores on CPU. Two cases were considered: small images (64^3 voxels), typically occurring in multilevel computations of medical images, and larger images (512^3 voxels), representing, e.g., state-of-the-art thoracic CT scans.

As can be seen from the results in Table 3, for both image sizes, all NGF evaluations—function value, gradient, and Hessian-vector multiplication—exhibit the expected speedup factors ranging from 11.0 to 15.0 on a 12-core system. Grid conversions from deformation grid to image grid also show speedups of approximately 12. For the transposed operator, on small images, a speedup of only 4.34 is achieved: due to the small number of parallel tasks available, not all computational cores can be fully utilized. On the large images, the expected speedup of 12.4 is achieved.

In comparison, the curvature regularizer scales much less favorably, with speedups ranging from 1.54 to 4.90. As the implementation of the curvature regularizer derivatives is fully parallelized and matrix-free, this indicates that the curvature computation is limited by memory bandwidth rather than arithmetic throughput. However, this has little effect on the overall performance, as the evaluation of the curvature regularizer is multiple orders of magnitude faster than the NGF evaluation.

6.2. Runtime comparison. As discussed in section 5, the matrix-free approach reduces the memory consumption and improves parallelization; however, it introduces additional cost due to multiple recalculations. Therefore, it is interesting to compare in detail the runtime of the matrix-free implementation MFC to the matrix-based implementations MBC and FAIR.

6.2.1. Derivative and Hessian evaluation. We separately evaluated runtimes for computation of the derivatives and for matrix-vector multiplication with the Gauss–Newton Hessian, including grid conversions.

Image sizes ranged from 64^3 to 512^3 voxels. To account for various use cases with different requirements for accuracy, we tested three different resolutions for the deformation grid: full resolution, 1/4 of the image resolution, and 1/16 of the image

Table 4

Runtimes (left) and speedup factors (right) for objective function gradient computations (top) and Hessian matrix-vector multiplications (bottom) on CPU. Three implementations were compared: a classical matrix-based implementation from the FAIR toolbox (FAIR), an optimized matrix-based C++ implementation (MBC), and the proposed matrix-free approach (MFC). For each image grid size m_i, three different deformation grid sizes m_i^{y} were tested. The proposed matrix-free approach MFC achieves speedups of 2.34–11.4 *(derivatives) and* 17.9–159 *(Hessian multiplications) compared to the optimized C++ implementation MBC (rightmost column). It can also handle high resolutions for which FAIR and MFC run out of memory (third and fourth columns, marked* $*$*).*

		Runtime			Speedup		
					MBC	**MFC**	**MFC**
	m_i	m_i^{y}	FAIR (s)	MBC (s)	**MFC** (s) (Proposed)	vs. FAIR	vs. FAIR	vs. MBC

	m_i	m_i^{y}	FAIR (s)	MBC (s)	**MFC** (s) (Proposed)	MBC vs. FAIR	**MFC** vs. FAIR	**MFC** vs. MBC
Gradient	512	513	*	*	15.39	–	–	–
		129	*	*	4.61	–	–	–
		33	*	*	4.48	–	–	–
	256	257	*	*	1.37	–	–	–
		65	155.62	5.42	0.57	28.7	**275.0**	**9.6**
		17	129.90	5.37	0.60	24.2	**214.8**	**8.9**
	128	129	19.57	1.70	0.19	11.5	**104.5**	**9.1**
		33	17.25	0.78	0.07	22.1	**252.4**	**11.4**
		9	14.45	0.80	0.12	18.1	**125.5**	**6.9**
	64	65	2.31	0.26	0.05	8.8	**44.8**	**5.1**
		17	1.94	0.11	0.01	17.7	**196.8**	**11.1**
		5	1.58	0.11	0.05	14.4	**33.7**	**2.3**
Hessian-vector mult.	512	513	*	*	83.14	–	–	–
		129	*	*	64.46	–	–	–
		33	*	*	64.28	–	–	–
	256	257	*	*	8.18	–	–	–
		65	*	*	7.51	–	–	–
		17	230.75	*	7.58	–	**30.4**	–
	128	129	*	*	1.06	–	–	–
		33	36.28	27.84	1.01	1.3	**35.9**	**27.5**
		9	24.44	21.37	1.01	1.1	**24.1**	**21.1**
	64	65	25.20	20.55	0.13	1.2	**195.5**	**159.5**
		17	4.05	3.38	0.12	1.2	**32.5**	**27.1**
		5	2.56	2.52	0.14	1.0	**18.2**	**17.9**

resolution. As can be seen from the results in Table 4, for the gradient computation, we achieve speedups from 33.7 to 275 relative to FAIR. Furthermore, FAIR runs out of memory at larger image and deformation grid sizes. Here, the matrix-free scheme allows for computations at much higher resolutions.

Relative to the matrix-based C++ implementation MBC, we achieve speedups between 2.34 and 11.4. These values are particularly remarkable if one considers that MBC is already highly optimized and parallelized, and that MFC includes a large amount of redundant computations. Again, MFC allows us to handle the highest resolution of 512^3, while MBC runs out of memory at 256^3 voxels.

Comparing the runtimes of the Hessian-vector multiplication in Table 4, the benefits of the low memory usage of MFC become even more obvious. While MBC and FAIR cannot be used for more than half of the resolutions tested, MFC has no such issues and scales approximately linear in the number of voxels. MFC achieves particularly high speedups for large deformation grids compared to the optimized C++ implementation MBC, ranging from 17.9 to 159.

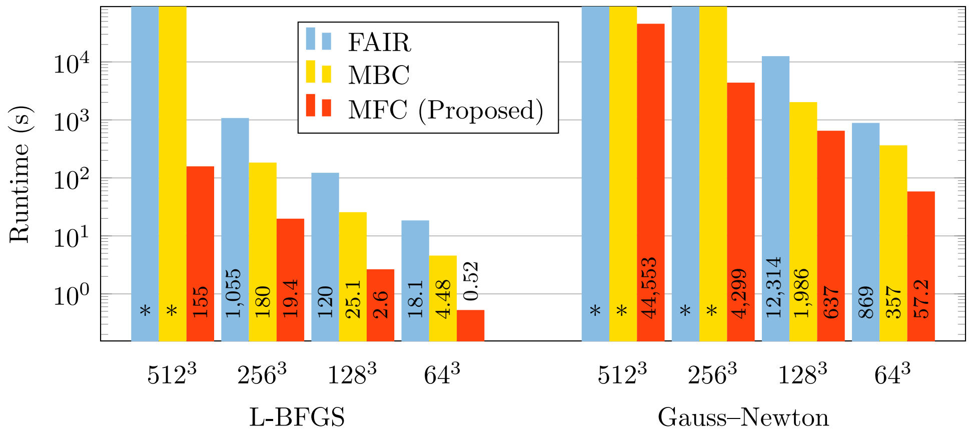


FIG. 4. *Runtime comparison for a full multilevel registration on CPU using L-BFGS and Gauss–Newton optimization schemes for different image sizes (logarithmic scale). FAIR: matrix-based, MATLAB, using the FAIR toolbox* [36]*; MBC: matrix-based, C++, using sparse matrices; MFC: proposed matrix-free algorithm in C++;* $*$*: computation ran out of memory. Deformation grid sizes were set to one quarter of the image size in each dimension. The proposed algorithm (MFC) enables faster computations at higher resolutions for both optimization schemes and routinely handles resolutions of up to* 512^3 *voxels.*

TABLE 5

Speedup factors of the proposed matrix-free approach on CPU (MFC is n times faster) relative to the matrix-based MBC and MATLAB-based FAIR methods for the data in Figure 4. $*$*: FAIR or MBC ran out of memory.*

	L-BFGS				Gauss–Newton			
	512^3	256^3	128^3	64^3	512^3	256^3	128^3	64^3
MFC vs. FAIR	*	54.4	46.1	34.8	*	*	19.4	15.2
MFC vs. MBC	*	9.27	9.67	8.62	*	*	3.12	6.25

6.2.2. Full registration. The ultimate goal of the presented approach is to improve runtime and memory usage of the full image registration system in real-world applications. Therefore, we performed a range of full multilevel image registrations on three levels for different image and deformation grid sizes with all three implementations. The input data consisted of thorax-abdomen images, as described in section 6.4.2 below, cropped to a size of 512^3 voxels. In order to avoid potentially large runtime differences caused by the effect of small numerical differences on the stopping criteria, we set a fixed number of 20 iterations on each resolution level.

For L-BFGS, the proposed method MFC is between 8.62 and 9.27 times faster than MBC (Figure 4 and Table 5). Computation on the highest resolution with an image size of 512^3 and deformation grid size of 129^3 is only possible using MFC due to memory limitations of the other methods.

For Gauss–Newton, overall speedups range from 3.12 to 6.25. Additionally, due to the more memory-intense Hessian computations, even a registration with an image size of 256^3 and a modest deformation grid size of 65^3 could only be performed with the proposed MFC method.

While for L-BFGS the overall speedups are in a range similar to the raw speedups in Table 4, they are slightly reduced for the Gauss–Newton method. This is due to the fact that in the matrix-based MBC method, the stored NGF Hessian is reused multiple

Table 6

Total peak memory usage in megabytes for the full multilevel registration on CPU, corresponding to the data in Figure 4. *Reported sizes include objective function derivatives, multilevel representations of the images, memory required for the optimization algorithm, CG solver, and further auxiliary space. Compared to MBC and FAIR, the proposed algorithm (MFC) enables computations at high resolutions with moderate memory use.* $*$: *FAIR or MBC ran out of memory (more than* 32 GB *required).*

	L-BFGS				Gauss–Newton			
	512^3	256^3	128^3	64^3	512^3	256^3	128^3	64^3
MFC	**4914**	**618**	**78**	**10**	**4296**	**539**	**68**	**9**
MBC	*	14 673	1837	230	*	*	5658	725
FAIR	*	17 422	2403	543	*	*	4375	800

times in the CG solver, while it is recomputed for each evaluation in the matrix-free MFC approach. Solving a linear system with the CG solver using the NGF Hessian is only needed for Gauss–Newton optimization, which could additionally impose a performance bottleneck. In our evaluations, however, we found that, while the CG solver consumes 97% to 99% of the total registration runtime, the matrix-free NGF Hessian-vector multiplications again consume more than 99% of the overall CG solver runtime, which can thus fully benefit from our parallelized computations.

The memory usage of the full multilevel registration for all approaches is shown in Table 6. Besides the objective function derivatives, the measured memory usage includes memory required for multilevel representations of the images, the optimization algorithm, the CG solver, and further auxiliary space. The matrix-free approach reduces memory requirements by one to two orders of magnitude. As the L-BFGS method uses additional buffers for storing previously computed gradients, the Gauss–Newton method has a lower memory usage with the matrix-free approach.

Overall, the proposed efficient matrix-free method enables both computation of higher resolutions and shorter runtimes.

6.3. GPU implementation. The presented matrix-free algorithm is not limited to a specific platform or processor type. Due to its parallelizable formulation and low memory requirements, it is also well-suited for implementation on specialized hardware such as GPUs. In comparison to CPUs, GPUs feature a massively parallel architecture and excellent computational performance. Thus, GPUs have become increasingly popular for general purpose computing applications [47]. However, in order to utilize these benefits, specialized implementations are required, which exploit platform-specific features such as GPU topology and different memory classes.

In this section, we present a GPU implementation of the matrix-free registration algorithm using NVIDIA CUDA C/C++ [40]. The CUDA framework allows for a comparatively easy implementation and direct access to hardware-related features such as different memory and caching models and has been widely adopted by the scientific community [47]. We implemented optimized versions of all parts of the registration algorithm in CUDA, allowing the full registration algorithm to run on the GPU without intermediate transfers to the host. In the following, we present details on how the implementation makes use of the specialized platform features in order to improve performance.

6.3.1. Implementation details. The CUDA programming model organizes the GPU code in *kernels*, which are launched from the host to execute on the GPU. The kernels are executed in parallel in different *threads*, which are grouped into *thread* *blocks.*

Memory areas. CUDA threads have access to different memory areas: global memory, constant memory, and shared memory. All data has to be transferred from the CPU main memory to the GPU global memory before it can be used for GPU computations. As frequent transfers can limit computational performance, the GPU implementation transfers the initial images R, T exactly once at the beginning of the registration. All computations, including the creation of coarser images for the multilevel scheme, are then performed entirely in GPU memory.

While the global memory is comparatively large (up to 16 GB on recent Tesla devices) and can be accessed from all threads, access is slow because caching is limited. Constant and shared memory, in contrast, are very fast. However, constant memory is read-only, while shared memory can only be accessed within the same thread block. Both are currently limited to 48 to 96 kB, depending on the architecture.

We therefore store frequently used parameters such as $m, m^{\mathrm{y}}, \bar{m}, h, h^{\mathrm{y}}, \bar{h}$ in constant memory to minimize access times. Shared memory (private per block) is used for reductions in L-BFGS and function value, with an efficient scheme from [20].

Image interpolation. Current GPUs include an additionally specialized memory area for textures. Texture memory stores the data optimized for localized access in 2D or 3D coordinates and provides hardware-based interpolation and boundary condition handling. Therefore, we initially employed texture memory for the image interpolation of the deformed template image $T(\hat{\mathbf{y}})$. However, we found that the hardware interpolation causes errors on the order of 10^{-3} in comparison to the CPU implementation, as the GPU only uses 9-bit fixed-point arithmetic for these computations. The errors resulted in erroneous derivative calculations, causing the optimization to fail. Despite an interpolation speedup of approximately 35% with texture memory, we therefore performed the interpolation in software within the kernels rather than in hardware.

Grid conversion. In section 4.3, a red-black scheme for the transposed operator $P^{\top}\mathbf{y}$ was proposed in order to avoid write conflicts in parallel execution. While on the CPU this results in fast calculations, on the GPU there are specialized *atomics* available to allow for fast simultaneous write accesses without conflicts. This enables a per-point parallel implementation without the red-black scheme, which also improves the limited speedup for small images shown in Table 3. We took care that all GPU computations return identical results to the CPU code with the exception of numerical and rounding errors, in order to allow for a meaningful comparison.

6.3.2. Evaluation. Experiments were performed on a GeForce GTX980 graphics card with 4 GB of memory. The GPU features 16 streaming multiprocessors with 128 CUDA cores each, resulting in 2048 CUDA cores in total for parallelization, and a theoretical peak performance of 4.6 TFLOPs for single-precision computations. The double-precision peak performance is much lower at 144 GFLOPs, which is less than the 192 GFLOPs achieved by the dual-CPU Xeon E5-2620 that was used for the CPU computations. Additionally, the previously mentioned *atomics* are only available for the single-precision *float* datatype. Therefore, all computations were performed in single precision. In order to minimize the effect of rounding errors, an exception was made for reductions, which were computed in double precision.

Comparing the peak performances of CPU and GPU, the theoretical maximum speedup is a factor of 20. While this maximum value is hard to obtain in practice, we will see that using the GPU can still lead to a substantial acceleration of the registration algorithm. To this end, we implemented a 2D image registration with the

Table 7

Runtimes (left) and speedup factors (right) for objective function gradient computations using the proposed matrix-free method, implemented on CPU (MFC) and GPU (MFC^{GPU}) in two dimensions. For each image grid size m_i, three different deformation grid sizes m_i^{y} were tested. The GPU-based implementation achieves additional speedup factors of up to 10.1 *in comparison to the matrix-free CPU implementation. The speedup increases with larger image and deformation grid sizes.*

		Runtime		Speedup
m_i	m_i^{y}	MFC (ms)	**MFC$^{\mathbf{GPU}}$** (ms)	**MFC$^{\mathbf{GPU}}$** vs. MFC
4096	2049	357	35.3	**10.1**
	1025	289	33.5	**8.63**
	513	286	32.3	**8.85**
2048	1025	96.7	10.0	**9.67**
	513	72.3	9.38	**7.71**
	257	74.3	9.32	**7.97**
1024	513	19.7	3.66	**5.38**
	257	16.2	3.53	**4.59**
	129	16.0	2.93	**5.46**
512	257	4.67	2.29	**2.04**
	129	4.02	1.67	**2.41**
	65	3.96	1.67	**2.37**
256	129	1.27	1.04	**1.22**
	65	1.16	1.03	**1.13**
	33	2.68	1.07	**2.50**

L-BFGS optimizer. As GPU implementations of FAIR and MBC are not available, we compare the results to the matrix-free CPU implementation MFC.

Derivative. Similar to section 6.2.1 and Table 4, we measured the runtime for the evaluation of the objective function derivative. The results are compared with the matrix-free algorithm on CPU in Table 7 for different image and deformation grid resolutions. Realistic speedups are in the range between 1.13 and 10.1. Relative GPU performance benefits from higher resolutions, with a maximum speedup at the full image size of 4096^2 pixels and a deformation grid size of 2049^2.

Full registration. We performed a full multilevel registration with three levels for different image and deformation grid sizes analogous to how it was done in section 6.2.2. The input consisted of patches of 4096^2 pixels in size from histological serial sections of tumor tissue, which were differently stained and scanned in a high resolution [31], as shown in Figure 5. Here, image registration allows us to align adjacent slices, evaluate different stains on the same slice, and compensate for deformation from the slicing process. Before registration, the images were rigidly prealigned and converted to gray-scale.

The matrix-free GPU implementation is compared to the corresponding matrix-free CPU implementation in Figure 6. We used a maximum of 20 iterations per level to ensure comparable runtimes. Depending on the image size, speedups from 1.04 to 9.21 are achieved. As before, compared to the CPU code, the highest speedup of 9.21 is achieved at the finest image resolution of 4096^2 pixels, since more parallel threads can be distributed on the GPU CUDA cores, resulting in a better GPU utilization.

Summary. While the proposed approach is well-suited for GPUs due to its high parallelizability, implementation requires careful consideration of the architecture's specifics. In particular, available memory types need to be fully utilized and unnecessary transfers avoided. While the slightly reduced accuracy has to be accounted for,

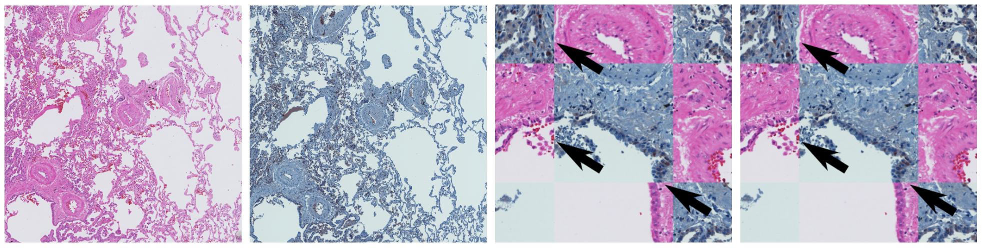
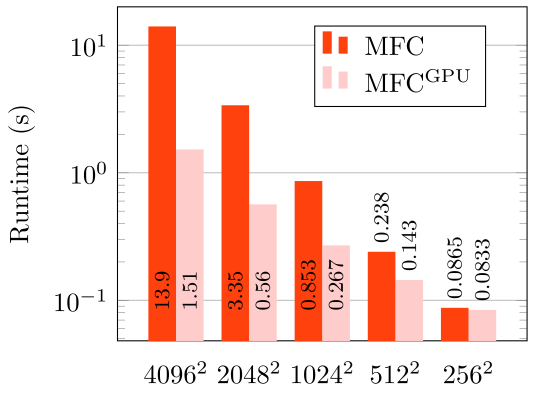


Fig. 5. *Registration of patches from histological serial sections with different stains on GPU.* (a) *reference image,* (b) *template image,* (c) *checkerboard overlay of details from both images before registration, and* (d) *from both images after registration. While discontinuities can be seen at the borders of the checker pattern before registration, smooth transitions are achieved after registration. Areas of large differences are indicated by arrows. The images were obtained from the publicly available dataset used in* [31].



(a)

	Speedup
m_i	**MFC$^{\mathbf{GPU}}$** vs. MFC
4096	9.21
2048	5.98
1024	3.19
512	1.66
256	1.04

(b)

Fig. 6. *Runtime comparison for a full multilevel registration on CPU and GPU.* (a) *runtimes for CPU-based (MFC) and GPU-based (MFC^{GPU}) implementations of the proposed matrix-free algorithm for different image sizes (logarithmic scale),* (b) *corresponding speedup factors. Deformation grid sizes were set to one quarter of the image size in each dimension. In comparison to the CPU implementation, the GPU implementation achieves an additional speedup of up to* 9.21*. The largest runtime improvements are achieved for high image resolutions.*

other features such as atomics can be used to reduce the code complexity without sacrificing performance. Overall, we achieved realistic speedups of about one order of magnitude compared to a CPU implementation.

6.4. Medical applications. For the CPU implementation of the matrix-free algorithm, we additionally studied two medical applications of the proposed approach in order to demonstrate the suitability for real-world tasks and to show its potential for clinical use. We used the L-BFGS method for these experiments—while in practice it required more iterations than the Gauss–Newton scheme, overall runtime was faster.

6.4.1. Pulmonary image registration. This medical application requires registration of maximum inhale and exhale images of four-dimensional (4D) thorax CT (4DCT) scans. The results can be used to assess local lung ventilation and aid the planning of radiation therapy of lung cancer [23].

Table 8

Average landmark errors in millimeters and execution times on CPU in seconds for the DIR-Lab 4*DCT datasets after affine-linear prealignment (Initial) and after deformable registration (Proposed). Compared to other results reported on the benchmark page* [7]*, the proposed algorithm achieves the lowest mean distance at the time of this writing, with an average runtime of* 9.23 s*; see also section* 6.4.1 *for a comparison.*

Dataset	Initial (mm)	Proposed (mm)	Runtime (s)
4DCT1	3.89 ± 2.78	0.81 ± 0.89	6.68
4DCT2	4.34 ± 3.90	0.75 ± 0.85	8.56
4DCT3	6.94 ± 4.05	0.92 ± 1.05	6.78
4DCT4	9.83 ± 4.86	1.34 ± 1.28	7.08
4DCT5	7.48 ± 5.51	1.06 ± 1.44	7.74
4DCT6	10.89 ± 6.97	0.86 ± 0.96	11.64
4DCT7	11.03 ± 7.43	0.83 ± 1.01	12.30
4DCT8	14.99 ± 9.01	1.02 ± 1.32	13.75
4DCT9	7.92 ± 3.98	0.88 ± 0.94	7.85
4DCT10	7.30 ± 6.35	0.84 ± 1.00	9.93
Mean	8.46 ± 5.48	$\mathbf{0.93 \pm 1.07}$	**9.23**

For benchmarking, we used the publicly available DIR-Lab database [8, 6], which consists of 10 4DCT scans. The maximum inhale and exhale images come with 300 expert-annotated landmarks each for evaluation of the registration accuracy. All images have voxel sizes of approximately $1\,\mathrm{mm} \times 1\,\mathrm{mm} \times 2.5\,\mathrm{mm}$.

As is common in pulmonary image registration, prior to registration the lungs were segmented and the images were cropped to the lung region [37]. The segmentation masks were generated with the fully automatic algorithm proposed in [29]. The cropped images were first coarsely registered with an affine-linear transformation model, the final result serving as an initial guess for the main deformable registration. Prior to deformable registration, the images were isotropically resampled in the z-direction to match the x-y plane resolution.

A multiresolution scheme with four levels was used, with the finest level being the image at full resolution. The deformation resolution was chosen four times coarser than the image resolution on the finest level. For each coarser level, the number of cells per dimension was halved. The regularizer weight was set to $\alpha = 1$ and the NGF filter parameters to $\varrho, \tau = 10$. The parameters were determined manually by evaluating several different parameter sets, and we also found them to be suitable for lung registration in other applications.

Table 8 summarizes the results and measured runtimes using the L-BFGS optimization method. At an average runtime of 9.23 s, the average landmark error is below the voxel diameter at 0.93 mm. The average landmark error is also the lowest reported in the public benchmark [7] at the time of this writing. In comparison, the most recent submission to the benchmark in [52] reports an average landmark error of 0.95 ± 1.15mm, with a much longer average runtime of 180 s. Two other methods [42, 22] achieve errors of 0.95 ± 1.07mm. Only does the author of [22] report a runtime, stating 98 s on average.

6.4.2. Oncological followup. As a second medical application of the proposed registration scheme, we considered oncological followup in the thorax-abdomen region. Here, CT scans are acquired at different time points—typically a few months apart—with the goal of assessing the development of tumors, e.g., during chemotherapy. Deformable registration is then employed to propagate prior findings to the current scan, facilitate side-by-side comparison of the same structures at multiple time points via cursor or slice synchronization, and visualize and highlight change by image subtraction.

On state-of-the-art scanners, the resolution of such CT scans is approximately 0.7 mm isotropic, leading to a challenging image size of $\approx 512 \times 512 \times 900$, depending on the size of the scanned region. To the best of our knowledge, no public data with expert annotations is available for evaluating registration accuracy. We therefore restricted ourselves to visual assessment and runtime comparison on images from clinical routine (first scan: $512 \times 512 \times 848$ voxels; second scan: $512 \times 512 \times 833$ voxels), acquired approximately nine months apart.

A multiresolution scheme with three levels was used, with one quarter of the original image resolution at the finest level. On each level, the deformation resolution was half the current image resolution. Further increasing the deformation resolution did not improve results, which is also supported by the findings in [24, 41]. The model parameters were manually chosen as $\alpha = 10$ and $\varrho, \tau = 5$.

Using the proposed method, we obtain qualitatively satisfying results (Figure 1) in a clinically acceptable runtime of 12.6 s, enabling online registration of thorax-abdomen CT scans for follow-up examinations.

7. Summary and conclusions.

We presented a novel computational approach for deformable image registration based on the widely used normalized gradient fields distance measure and curvature regularization. Through detailed mathematical analysis of the building blocks, we derived a new matrix-free formulation, which tackles two main bottlenecks of the employed registration algorithm.

First, it enables *effective parallelization.* The most-used components exhibit virtually linear scalability (section 6.1), efficiently utilizing multiple computational cores on modern CPUs. This leads to an average reduction in overall runtime by a factor of 45.1 for L-BFGS optimization (Gauss–Newton: 17.3) compared to a MATLAB implementation, and 9.19 for L-BFGS optimization (Gauss–Newton: 4.69) compared to a sparse matrix-based, optimized C++ implementation, despite necessary recomputations of intermediate values. Our GPU-based implementation of the matrix-free registration approach, exploiting the massively parallel architecture, achieves a further speedup of 9.21 compared to an optimized CPU-based implementation.

Second, the presented method largely reduces the *memory requirements* of the registration algorithm. As shown in Table 4, even medium resolutions cannot be handled using matrix-based approaches, while the matrix-free scheme is ultimately only limited by runtime.

Experiments showed that the presented algorithm is able to solve real-world clinical registration tasks with high accuracy at fast runtimes. In the DIR-Lab 4DCT benchmark for lung registration, we achieve the best mean distance among all submissions at an average runtime of 9.23 s. In thorax-abdomen registration for oncology screening, large images could be processed within 12.6 s, potentially allowing for online assessment by clinicians.

The presented scheme allows for efficient image registration on much higher resolutions than previously possible and shows a substantial speedup over existing implementations. It is not tied to a specific platform or CPU type and is very suitable for implementation on GPUs, enabling a wider use of rapid deformable image registration in various applications.

Acknowledgment. We thank Martin Meike for providing the GPU implementation developed in his thesis [34].

REFERENCES

[1] B. B. Avants, C. L. Epstein, M. Grossman, and J. C. Gee, *Symmetric diffeomorphic image registration with cross-correlation: Evaluating automated labeling of elderly and neurodegenerative brain*, Med. Image. Anal., 12 (2008), pp. 26–41.

[2] R. Berg, L. König, J. Rühaak, R. Lausen, and B. Fischer, *Highly efficient image registration for embedded systems using a distributed multicore DSP architecture*, J. Real-Time Image Process., 14 (2018), pp. 341–361.

[3] C. Broit, *Optimal Registration of Deformed Images*, Ph.D. thesis, University of Pennsylvania, Philadelphia, 1981.

[4] P. Bui and J. Brockman, *Performance analysis of accelerated image registration using GPGPU*, in Proceedings of 2nd ACM Workshop on General Purpose Processing on Graphics Processing Units, 2009, pp. 38–45.

[5] M. Burger, J. Modersitzki, and L. Ruthotto, *A hyperelastic regularization energy for image registration*, SIAM J. Sci. Comput., 35 (2013), pp. B132–B148, https://doi.org/10.1137/110835955.

[6] E. Castillo, R. Castillo, J. Martinez, M. Shenoy, and T. Guerrero, *Four-dimensional deformable image registration using trajectory modeling*, Phys. Med. Biol., 55 (2010), pp. 305–327.

[7] R. Castillo, *DIR-Lab Results*. https://www.dir-lab.com/Results.html (accessed: 2017-04-10).

[8] R. Castillo, E. Castillo, R. Guerra, V. E. Johnson, T. McPhail, A. K. Garg, and T. Guerrero, *A framework for evaluation of deformable image registration spatial accuracy using large landmark point sets*, Phys. Med. Biol., 54 (2009), pp. 1849–1870.

[9] C. R. Castro-Pareja, J. M. Jagadeesh, and R. Shekhar, *FAIR: A hardware architecture for real-time 3-D image registration*, IEEE Trans. Inf. Technol. Biomed., 7 (2003), pp. 426–434.

[10] A. Collignon, F. Maes, D. Delaere, D. Vandermeulen, P. Suetens, and G. Marchal, *Automated multi-modality image registration based on information theory*, in Information Processing in Medical Imaging, Vol. 3, Springer, Dordrecht, The Netherlands, 1995, pp. 263–274.

[11] V. De Luca, T. Benz, S. Kondo, L. König, D. Lübke, S. Rothlübbers, O. Somphone, S. Allaire, M. L. Bell, D. Chung, et al., *The 2014 liver ultrasound tracking benchmark*, Phys. Med. Biol., 60 (2015), pp. 5571–5599.

[12] A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte, *Medical image processing on the GPU – Past, present and future*, Med. Image. Anal., 17 (2013), pp. 1073–1094.

[13] B. Fischer and J. Modersitzki, *Fast diffusion registration*, in Inverse Problems, Image Analysis, and Medical Imaging, Contemp. Math. 313, AMS, Providence, RI, 2002, pp. 117–128.

[14] B. Fischer and J. Modersitzki, *Curvature based image registration*, J. Math. Imaging Vision, 18 (2003), pp. 81–85.

[15] C. J. Galbán, M. K. Han, J. L. Boes, K. A. Chughtai, C. R. Meyer, T. D. Johnson, S. Galbán, A. Rehemtulla, E. A. Kazerooni, F. J. Martinez, et al., *Computed tomography-based biomarker provides unique signature for diagnosis of COPD phenotypes and disease progression*, Nat. Med., 18 (2012), pp. 1711–1715.

[16] F. Gigengack, L. Ruthotto, M. Burger, C. H. Wolters, X. Jiang, and K. P. Schafers, *Motion correction in dual gated cardiac PET using mass-preserving image registration*, IEEE Trans. Med. Imag., 31 (2012), pp. 698–712.

[17] E. Haber, S. Heldmann, and J. Modersitzki, *An octree method for parametric image registration*, SIAM J. Sci. Comput., 29 (2007), pp. 2008–2023, https://doi.org/10.1137/060662605.

[18] E. Haber, S. Heldmann, and J. Modersitzki, *Adaptive mesh refinement for nonparametric image registration*, SIAM J. Sci. Comput., 30 (2008), pp. 3012–3027, https://doi.org/10.1137/070687724.

[19] E. Haber and J. Modersitzki, *Beyond mutual information: A simple and robust alternative*, in Bildverarbeitung für die Medizin, Springer, Berlin, Heidelberg, 2005, pp. 350–354.

[20] M. Harris, *Optimizing Parallel Reduction in CUDA*, NVIDIA Developer Technology, 2007.

[21] S. Heldmann, *Non-linear Registration Based on Mutual Information: Theory, Numerics, and Application*, Logos-Verlag, Berlin, 2006.

[22] S. Hermann, *Evaluation of scan-line optimization for 3D medical image registration*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 3073–3080.

[23] S. Kabus, J. von Berg, T. Yamamoto, R. Opfer, and P. J. Keall, *Lung ventilation estimation based on 4D-CT imaging*, in First International Workshop on Pulmonary Image

Analysis, 2008, pp. 73–81.

[24] A. Köhn, J. Drexl, F. Ritter, M. König, and H.-O. Peitgen, *GPU accelerated image registration in two and three dimensions*, in Bildverarbeitung für die Medizin, Springer, Berlin, Heidelberg, 2006, pp. 261–265.

[25] L. König, A. Derksen, M. Hallmann, and N. Papenberg, *Parallel and memory efficient multimodal image registration for radiotherapy using normalized gradient fields*, in Proceedings of the IEEE International Symposium on Biomedical Imaging: From Nano to Macro, 2015, pp. 734–738.

[26] L. König, A. Derksen, N. Papenberg, and B. Haas, *Deformable image registration for adaptive radiotherapy with guaranteed local rigidity constraints*, Radiat. Oncol., 11 (2016), 122.

[27] L. König, T. Kipshagen, and J. Rühaak, *A non-linear image registration scheme for real-time liver ultrasound tracking using normalized gradient fields*, in Proceedings of the MICCAI CLUST14 Workshop, Boston, MA, 2014, pp. 29–36.

[28] L. König and J. Rühaak, *A fast and accurate parallel algorithm for non-linear image registration using normalized gradient fields*, in Proceedings of the IEEE International Symposium on Biomedical Imaging: From Nano to Macro, 2014, pp. 580–583.

[29] B. Lassen, J.-M. Kuhnigk, M. Schmidt, S. Krass, and H.-O. Peitgen, *Lung and lung lobe segmentation methods at Fraunhofer MEVIS*, in Fourth International Workshop on Pulmonary Image Analysis, 2011, pp. 185–200.

[30] J. Le Moigne, N. S. Netanyahu, and R. D. Eastman, *Image Registration for Remote Sensing*, Cambridge University Press, Cambridge, UK, 2011.

[31] J. Lotz, J. Olesch, B. Muller, T. Polzin, P. Galuschka, J. M. Lotz, S. Heldmann, H. Laue, M. Gonzalez-Vallinas, A. Warth, et al., *Patch-based nonlinear image registration for gigapixel whole slide images*, IEEE Trans. Biomed. Eng, 63 (2016), pp. 1812–1819.

[32] A. Mang and G. Biros, *A semi-Lagrangian two-level preconditioned Newton–Krylov solver for constrained diffeomorphic image registration*, SIAM J. Sci. Comput., 39 (2017), pp. B1064–B1101, https://doi.org/10.1137/16M1070475.

[33] A. Mang, A. Gholami, and G. Biros, *Distributed-memory large deformation diffeomorphic 3D image registration*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2016, pp. 842–853.

[34] M. Meike, *GPU-basierte nichtlineare Bildregistrierung*, Master's thesis, University of Lübeck, Lübeck, Germany, 2016.

[35] J. Modersitzki, *Numerical Methods for Image Registration*, Oxford University Press, Oxford, UK, 2004.

[36] J. Modersitzki, *FAIR: Flexible Algorithms for Image Registration*, Fundam. Algorithms 6, SIAM, Philadelphia, 2009, https://doi.org/10.1137/1.9780898718843.

[37] K. Murphy, B. Van Ginneken, J. M. Reinhardt, S. Kabus, K. Ding, X. Deng, K. Cao, K. Du, G. E. Christensen, V. Garcia, et al., *Evaluation of registration methods on thoracic CT: The EMPIRE10 challenge*, IEEE Trans. Med. Imag., 30 (2011), pp. 1901–1920.

[38] P. Muyan-Ozcelik, J. D. Owens, J. Xia, and S. S. Samant, *Fast deformable registration on the GPU: A CUDA implementation of demons*, in Proceedings of the IEEE International Conference on Computational Sciences and Its Applications (ICCSA), 2008, pp. 223–233.

[39] J. Nocedal and S. Wright, *Numerical Optimization*, Springer, New York, 1999.

[40] NVIDIA Corporation, *CUDA C Programming Guide*, no. PG-02829-001_v8.0, 2017, http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf.

[41] T. Polzin, M. Niethammer, M. P. Heinrich, H. Handels, and J. Modersitzki, *Memory efficient LDDMM for lung CT*, in Proceedings of the 19th International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI 2016), 2016, pp. 28–36.

[42] T. Polzin, J. Rühaak, R. Werner, J. Strehlow, S. Heldmann, H. Handels, and J. Modersitzki, *Combining automatic landmark detection and variational methods for lung CT registration*, in Fifth International Workshop on Pulmonary Image Analysis, 2013, pp. 85–96.

[43] J. Rühaak, S. Heldmann, T. Kipshagen, and B. Fischer, *Highly accurate fast lung CT registration*, in SPIE Medical Imaging, Image Processing, SPIE 86690, International Society for Optics and Photonics, Bellingham, WA, 2013.

[44] J. Rühaak, L. König, F. Tramnitzke, H. Köstler, and J. Modersitzki, *A matrix-free approach to efficient affine-linear image registration on CPU and GPU*, J. Real-Time Image Process., 13 (2017), pp. 205–225.

[45] J. Shackleford, N. Kandasamy, and G. Sharp, *High Performance Deformable Image Registration Algorithms for Manycore Processors*, Morgan Kaufmann, Waltham, MA, 2013.

[46] D. P. Shamonin, E. E. Bron, B. P. Lelieveldt, M. Smits, S. Klein, and M. Staring, *Fast parallel image registration on CPU and GPU for diagnostic classification of Alzheimer's disease*, Front. Neuroinform., 7 (2013), pp. 1–15.

[47] R. Shams, P. Sadeghi, R. Kennedy, and R. Hartley, *A survey of medical image registration on multicore and the GPU*, IEEE Signal Process. Mag., 27 (2010), pp. 50–60.

[48] M. Stürmer, H. Köstler, and U. Rüde, *A fast full multigrid solver for applications in image processing*, Numer. Linear Algebra Appl., 15 (2008), pp. 187–200.

[49] K. Stützer, R. Haase, F. Lohaus, S. Barczyk, F. Exner, S. Löck, J. Rühaak, et al., *Evaluation of a deformable registration algorithm for subsequent lung computed tomography imaging during radiochemotherapy*, Med. Phys., 43 (2016), pp. 5028–5039.

[50] T. Vercauteren, X. Pennec, A. Perchant, and N. Ayache, *Diffeomorphic demons: Efficient non-parametric image registration*, NeuroImage, 45 (2009), pp. S61–S72.

[51] P. Viola and W. Wells III, *Alignment by maximization of mutual information*, Int. J. Comput. Vis., 24 (1997), pp. 137–154.

[52] V. Vishnevskiy, T. Gass, G. Szekely, C. Tanner, and O. Goksel, *Isotropic total variation regularization of displacements in parametric image registration*, IEEE Trans. Med. Imag., 36 (2017), pp. 385–395.