SC20: International Conference for High Performance Computing, Networking, Storage and Analysis | 978-1-7281-9998-6/20/$31.00 ©2020 IEEE | DOI: 10.1109/SC41405.2020.00042

Multi-Node Multi-GPU Diffeomorphic Image Registration for Large-Scale Imaging Problems

Malte Brunn*, Naveen Himthani†, George Biros†, Miriam Mehl*, and Andreas Mang‡
*Computer Science, *University of Stuttgart*, Stuttgart, DE, Email: {malte.brunn,miriam.mehl}@ipvs.uni-stuttgart.de
†Oden Institute, *University of Texas*, Austin TX, US, Email: {naveen@oden.utexas.edu, gbiros@acm.org}
‡Mathematics, *University of Houston*, Houston TX, US, Email: andreas@math.uh.edu

***Abstract*—We present a Gauss-Newton-Krylov solver for large deformation diffeomorphic image registration. We extend the publicly available CLAIRE library to multi-node multi-graphics processing unit (GPUs) systems and introduce novel algorithmic modifications that significantly improve performance. Our contributions comprise (*i*) a new preconditioner for the reduced-space Gauss-Newton Hessian system, (*ii*) a highly-optimized multi-node multi-GPU implementation exploiting device direct communication for the main computational kernels (interpolation, high-order finite difference operators and Fast-Fourier-Transform), and (*iii*) a comparison with state-of-the-art CPU and GPU implementations. We solve a 256^3-resolution image registration problem in five seconds on a single NVIDIA Tesla V100, with a performance speedup of 70% compared to the state-of-the-art. In our largest run, we register 2048^3 resolution images (25 B unknowns; approximately 152× larger than the largest problem solved in state-of-the-art GPU implementations) on 64 nodes with 256 GPUs on TACC's Longhorn system.**

***Index Terms*—diffeomorphic image registration, multi-node multi-GPU, Gauss–Newton–Krylov solver, PDE-constrained optimization, preconditioning**

I. Introduction

3D diffeomorphic image registration is a critical task in biomedical imaging applications [1]–[3]. For example, it enables the analysis and study of morphological changes associated with the progression of neurodegenerative diseases in time series of medical images or in imaging studies of patient populations. The input to this inverse problem are two (or more) images $m_0(\boldsymbol{x})$ (the "*template image*") and $m_1(\boldsymbol{x})$ (the "*reference image*") of the same type of object, compactly supported on a domain $\Omega \subset \mathbb{R}^3$. The task of image registration is to compute a spatial transformation or mapping $\boldsymbol{y}(\boldsymbol{x})$ such that $m_0(\boldsymbol{y}(\boldsymbol{x})) \approx m_1(\boldsymbol{x})$ for all points $\boldsymbol{x} \in \Omega$ [2] (see Figure 1). Methods for the registration of images can be classified according to the parameterization for $\boldsymbol{y}$ [2]. We will consider maps $\boldsymbol{y}$ that are ***diffeomorphisms***, i.e., maps that are a differentiable bijection, and have a differentiable inverse. In the present work, we consider formulations that belong or are related to a class of methods referred to as *large-deformation diffeomorphic metric mapping* (**LDDMM**) [4]–[6]. These methods parameterize diffeomorphisms in terms of a smooth (time-dependent) velocity field. The associated mappings provide maximal flexibility [3] but are expensive to compute: the problem is infinite-dimensional, and upon discretization it becomes a nonlinear system with millions or even billions of unknowns. For example, registering two volumes of grid size 256^3 (a typical data size for clinical images) necessitates solving for approximately 50 M unknowns (three vector components per image grid point). This is further complicated by the fact that image registration is a highly non-linear, ill-posed inverse problem [1], resulting in ill-conditioned inversion operators. As a result, image registration can take several minutes on multi-core high-end CPUs. As clinical workflows for multi-center population-studies that require thousands of registrations become increasingly more common, execution time of a single registration becomes more and more critical; reducing the runtime to seconds corresponds to a reduction of clinical study time from weeks to a few days. GPUs with their inherent parallelism and low energy consumption are an attractive choice to achieve this goal. However, despite the need for high computational throughput and the existence of several software packages for LDDMM, there is little work on high-performance GPU implementations, and even less work on multi-node multi-GPU implementations for large-scale applications. One such application is the registration of CLARITY images [7]–[12] of resolution in the order of $20\,\mathrm{K} \times 20\,\mathrm{K} \times 1\,\mathrm{K}$, which corresponds to a problem with about 1.2 trillion unknowns (see Figure 2).

This work was partly supported by the National Science Foundation (DMS-1854853, DMS-2009923, DMS-2012825, CCF-1817048, CCF-1725743), the NVIDIA Corporation (NVIDIA GPU Grant Program), the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy-EXC 2075-390740016, by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Number DE-SC0019393; by the U.S. Air Force Office of Scientific Research award FA9550-17-1-0190; by the Portugal Foundation for Science and Technology and the UT Austin-Portugal program, and by NIH award 5R01NS042645-11A1. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the DFG, AFOSR, DOE, NIH, and NSF. Computing time on the Texas Advanced Computing Centers' (TACC) systems was provided by an allocation from TACC and the NSF. This work was completed in part with resources provided by the Research Computing Data Core at the University of Houston.

A. Formulation and Outline of the Method

We summarize our notation in Table I. `CLAIRE` uses an optimal control formulation. The deformation map $\boldsymbol{y}(\boldsymbol{x})$ is parameterized through a smooth, stationary velocity field $\boldsymbol{v}(\boldsymbol{x})$. The optimization problem is: given two images $m_0(\boldsymbol{x})$ (template image; image to be deformed) and $m_1(\boldsymbol{x})$ (reference

SC20, November 9-19, 2020, Is Everywhere We Are
978-1-7281-9998-6/20/$31.00 ©2020 IEEE

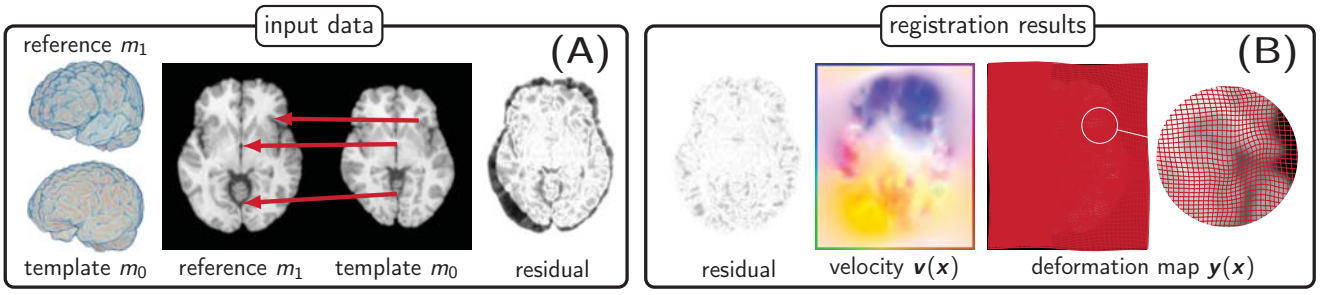

Fig. 1: *3D diffeomorphic image registration problem for human neuroimaging data. We illustrate the input data and the registration problem in panel (A) and the results in panel (B) for a multi-subject registration problem (NIREP dataset [13]; reference image:* `na01`*; template image:* `na10`*). Panel (A) [from left to right]: Volume rendering of the input images, axial view of the reference image, axial view of the template image, and the residual of these views before registration (white: small residual; black: large residual). The image registration problem is to identify spatial correspondences that map points from one image (the template image) to points in another image (the reference image); see red arrows. Panel (B) [from left to right]: residual after registration, computed velocity field $\boldsymbol{v}(\boldsymbol{x})$ that parameterizes the deformation map $\boldsymbol{y}(\boldsymbol{x})$ (color denotes orientation), and an illustration of $\boldsymbol{y}(\boldsymbol{x})$. Qualitatively, the computed map is a smooth diffeomorphism (confirmed numerically).*

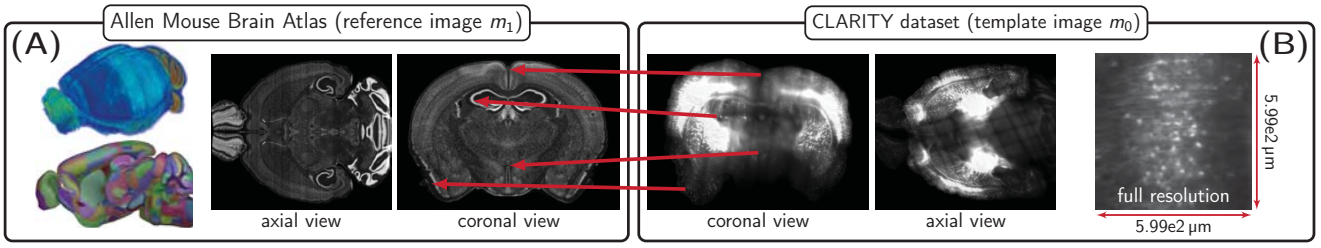

Fig. 2: *3D image registration problem for murine CLARITY imaging data. We illustrate a multi-subject registration problem. In panel (A), we show the Allen Mouse Brain Atlas [14] with a grid size of* $800 \times 1140 \times 1320$*. We show a volume rendering (top left), annotations of anatomical regions (bottom left), an axial view (middle) and a coronal view (right). In panel (B), we show a coronal and an axial view (after affine pre-registration to the atlas image), as well as a closeup of a subregion in full resolution. This CLARITY volume (Control 189) has a resolution of* $585\,\mathrm{nm} \times 585\,\mathrm{nm} \times 5\,\mu\mathrm{m}$ *with a grid size of* $20\,084 \times 24\,618 \times 1333$*. Once we have found the diffeomorphism, we can transfer the annotations of the anatomical regions identified in the atlas (see panel (A)) to the CLARITY dataset, and study anatomical subregions.*

Table I: *Notation and main symbols.*

Symbol	Description
Ω	spatial domain; $\Omega := [0, 2\pi)^3 \subset \mathbb{R}^3$ with boundary $\partial\Omega$
$\boldsymbol{x}$	spatial coordinate; $\boldsymbol{x} := (x_1, x_2, x_3)^\mathsf{T} \in \mathbb{R}^3$
t	pseudo-time variable; $t \in [0, 1]$
$m_1(\boldsymbol{x})$	reference image
$m_0(\boldsymbol{x})$	template image (image to be registered)
$\boldsymbol{v}(\boldsymbol{x})$	stationary velocity field
$\boldsymbol{y}(\boldsymbol{x})$	deformation map
$m(\boldsymbol{x}, t)$	state variable (transported intensities of m_0)
$\lambda(\boldsymbol{x}, t)$	adjoint variable
$\mathcal{A}$	regularization operator
$\beta > 0$	regularization parameter

image), we seek $\boldsymbol{v}(\boldsymbol{x})$ by solving

$$\underset{\boldsymbol{v},m}{\text{minimize}} \quad \frac{1}{2}\int_\Omega (m(\boldsymbol{x},1) - m_1(\boldsymbol{x}))^2 \mathrm{d}\boldsymbol{x} + \frac{\beta}{2}\,\mathrm{reg}(\boldsymbol{v}) \tag{1a}$$

subject to

$$\begin{aligned} \partial_t m(\boldsymbol{x},t) + \boldsymbol{v}(\boldsymbol{x}) \cdot \nabla m(\boldsymbol{x},t) &= 0 && \text{in } \Omega \times (0,1], \\ m(\boldsymbol{x},t) &= m_0(\boldsymbol{x}) && \text{in } \Omega \times \{0\} \end{aligned} \tag{1b}$$

on a three-dimensional rectangular domain Ω with periodic boundary conditions on $\partial\Omega$. The first term in (1a) is a similarity measure for the proximity between the deformed template image $m(\boldsymbol{x}, 1)$ and the reference image $m_1(\boldsymbol{x})$. Without loss of generality, we consider a squared L^2-distance. The second term in (1a) is a Tikhonov regularization functional with regularization parameter $\beta > 0$. This regularization operator is not only introduced to alleviate issues with the ill-posedness of the inverse problem but also prescribes sufficient regularity requirements for $\boldsymbol{v}(\boldsymbol{x})$ to ensure that the computed geometric transformation is a diffeomorphism [4], [6], [15]–[18]. The default configuration of `CLAIRE` is an H^1-Sobolev-seminorm; the regularization model is a standard L^2-inner product $\langle \mathcal{A}\boldsymbol{v}(\boldsymbol{x}), \boldsymbol{v}(\boldsymbol{x})\rangle_{L^2(\Omega)^3}$, where $\mathcal{A}$ is a vector Laplacian operator. The formulation is augmented with an additional penalty on the divergence of $\boldsymbol{v}$ (see [19], [20] for details). The transport equation (1b) describes the geometric transformation of the template image $m_0(\boldsymbol{x})$ by advecting the intensities forward in time. We use a reduced-space Gauss–Newton–Krylov method to solve (1). Details can be found in §II.

B. Contributions & Challenges

We extend the open source diffeomorphic image registration framework termed CLAIRE [19]–[25]. CLAIRE uses an optimal control formulation with partial differential equations (**PDE**s; e.g., a pure advection equation for the image intensities) as constraints. The overall mathematical formulation and solution strategy has not been altered from [20]. CLAIRE has been developed to scale on standard x86 CPU clusters using the Message Passing Interface (**MPI**) for parallelism [20], [22], [24], [25] and has recently been ported to GPU architectures (***single-node single-GPU*** implementation) [21].

In the present work, we propose a new, highly optimized multi-node multi-GPU implementation of CLAIRE. The main ***challenges*** are (*i*) eliminating costly host-to-device copies, (*ii*) addressing significant communication costs between devices, (*iii*) reducing memory pressure to enable large-scale runs on limited resources, and (*iv*) identifying an adequate balance between parallelism and local computational throughput. (Per GPU we need to hold enough data to locally perform a sufficient amount of computations, since the computational kernels are extremely fast. Too few data to process per GPU deteriorates scalability. This effect is much more pronounced on GPUs compared to CPUs.) Our main ***contributions*** are:

1) We propose an efficient GPU-only single- and multi-node multi-GPU implementation of CLAIRE. The proposed multi-GPU implementation is available for download at https://github.com/andreasmang/claire [25].

2) We minimize communication between host and device through CUDA-aware MPI, and increase the computational throughput in the most important computational kernels of the solver, scattered-data interpolation (**IP**) and differentiation.

3) We propose several improvements to reduce memory pressure and, thus, further increase the computational throughput. With the proposed implementation, it is possible to solve problems with datasets of grid sizes of 512^3 on a single node using four NVIDIA Tesla V100 GPUs in under $30\,\mathrm{s}$.

4) We propose a completely new preconditioner for the reduced-space Hessian based on a zero-velocity approximation, which we term "*Inv*$\mathcal{H}_0$". This allows us to eliminate expensive incremental forward and adjoint PDE solves (hyperbolic transport equations) in the evaluation of the preconditioner. Our method is matrix-free (we do not store or assemble the preconditioner or the Hessian). To further amortize computational costs, we propose a two-level coarse grid approximation.

5) We report results for synthetic and real data, which includes results for CLARITY imaging data for a grid size of $1024\times768\times768$. Overall, we achieve a speedup of up to about 70% on a single GPU compared to the state-of-the-art [21]. This makes the proposed solver $34\times$ faster than the CPU version [20], [22], [24] and $50\times$ faster than other, exemplary GPU-accelerated implementations for LDDMM (c.f., benchmark study in [21]). Moreover, our multi-GPU implementation allows us to solve problems that are approximately $152\times$ larger ($N = 2048^3$, 25 B unknowns) compared to [21].

C. Limitations

We have optimized memory allocation for the core components of CLAIRE. Additional optimizations by sharing memory across external libraries and parallel-in-time integration methods to further reduce the memory pressure remain subject to future work. Moreover, CLAIRE uses stationary velocities. This drastically improves efficiency, but results in theoretical limitations.

D. Related Work

The present work builds upon the open source framework termed CLAIRE [19]–[25]. Related LDDMM software packages include Demons [26], ANTs [27]–[29], DARTEL [30], deformetrica [31]–[34], FLASH [35], LDDMM [4], [36], ARDENT [37], ITKNDReg [38], and PyCA [39]. Literature surveys of image registration can be found in [2], [3]. We refer to [20] for a recent overview of existing LDDMM methods. Surveys of GPU-accelerated solvers for image registration are [40]–[42]; particular examples for various formulations are [32], [43]–[58]. Multi-GPU implementations for LDDMM in the context of atlas construction are described in [49], [50], [57], [58]. None of the hardware-accelerated LDDMM methods cited above, except for CLAIRE [19]–[25], use second-order information for numerical optimization. Many of the available methods reduce the number of unknowns by using coarser resolutions either through parameterization or by solving the problem on coarser grids; they use simplified algorithms and deliver subpar registration quality.

The work most pertinent to ours is [21], [49], [50]. In [49], [50], a multi-node multi-GPU implementation of the algorithm in [51] is presented. The considered application is atlas construction from multiple image volumes. While computational throughput on a single GPU is optimized, the focus is on data-parallelism: Multiple input images are loaded and synchronously processed on distinct GPUs. We propose a multi-node multi-GPU framework with high computational throughput for single (large-scale) registration problems. This problem is no longer embarrassingly parallel. The computational bottlenecks in [49]–[51] are the repeated solution of a Helmholtz-type PDE and trilinear scattered data interpolation to apply the deformation map. The PDE is solved via an implicit successive over-relaxation method. The trilinear interpolation kernel is hardware accelerated with 3D texture volume support. The runtime for a single dataset of size $160 \times 192 \times 160$ is $20\,\mathrm{s}$ on an NVIDIA Quadro FX 5600. The work in [21] presents a single-node single-GPU implementation of CLAIRE. The present work ports CLAIRE to a heterogeneous multi-node multi-GPU environment by exploiting CUDA-aware MPI. We present several improvements over the computational kernels described in [21] (see contributions above).

II. Discretization and Numerical Algorithms

To solve (1), we apply the method of Lagrange multipliers to obtain the Lagrangian functional,

$$\begin{aligned}\mathcal{L}(\boldsymbol{\phi}) :=& \frac{1}{2}\int_\Omega (m(\boldsymbol{x},1)-m_1(\boldsymbol{x}))^2\,\mathrm{d}\boldsymbol{x} + \frac{\beta}{2}\,\mathrm{reg}(\boldsymbol{v}) \\ &+ \int_0^1\int_\Omega \lambda(\boldsymbol{x},t)(\partial_t m + \boldsymbol{v}\cdot\nabla m)\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}t \\ &+ \int_\Omega \lambda(\boldsymbol{x},0)(m(\boldsymbol{x},0)-m_0(\boldsymbol{x}))\,\mathrm{d}\boldsymbol{x},\end{aligned}$$

with state, adjoint, and control variables $(m,\lambda,\boldsymbol{v}) := \boldsymbol{\phi}$, respectively.

Optimality Conditions & Reduced Space Approach: We derive first-order optimality conditions by taking variations with respect to the state variable m, the adjoint variable λ, and the control variable $\boldsymbol{v}$. This results in a set of coupled, hyperbolic-elliptic PDEs in *4D (space-time)*, consisting of three equations. At optimality, we require that the gradient of our problem vanishes. CLAIRE uses a ***reduced-space approach***, in which one iterates only on the reduced-space of $\boldsymbol{v}$. We require that $\boldsymbol{g}(\boldsymbol{v}) = \boldsymbol{0}$, where

$$\boldsymbol{g}(\boldsymbol{v}) := \beta\mathcal{A}\boldsymbol{v}(\boldsymbol{x}) + \int_0^1 \lambda(\boldsymbol{x},t)\nabla m(\boldsymbol{x},t)\,\mathrm{d}t. \tag{2}$$

is the so-called ***reduced gradient*** system (variation of $\mathcal{L}$ with respect to $\boldsymbol{v}$). To evaluate (2), we first solve the forward problem (1b) (variation of $\mathcal{L}$ with respect to λ) and then the *adjoint problem* (variation of $\mathcal{L}$ with respect to m)

$$-\partial_t\lambda(\boldsymbol{x},t) - \nabla\cdot\lambda(\boldsymbol{x},t)\boldsymbol{v}(\boldsymbol{x}) = 0 \qquad \text{in } \Omega\times[0,1) \tag{3}$$

with final condition $\lambda(\boldsymbol{x},t) = m_1(\boldsymbol{x}) - m(\boldsymbol{x},t)$ in $\Omega\times\{1\}$ and periodic boundary conditions on $\partial\Omega$. CLAIRE uses a Newton–Krylov method to solve the non-linear problem $\boldsymbol{g}(\boldsymbol{v}) = \boldsymbol{0}$ as described below.

Discretization: The forward and adjoint PDEs in the space-time interval $\Omega\times[0,1]$, $\Omega := [0,2\pi)^3 \subset \mathbb{R}^3$, with periodic boundary conditions on $\partial\Omega$, are discretized on a regular grid with $N = N_1N_2N_3$ grid points $\boldsymbol{x}_{ijk} \in \mathbb{R}^3$ in space and N_t+1 grid points in time. A semi-Lagrangian scheme is used to solve the transport equations that appear in the optimality system [24], [59]. That is, the advection term is discretized in space and time based on backward trajectories of grid points. The total time derivative is evaluated by means of the difference of the current value of the transported variable at a grid point and the previous time step's value at the end point of a backward trajectory in time. An interpolation in space is needed at the end points of the backward trajectories that are, in general, off-grid points. The backward trajectories themselves are calculated by solving an ODE of the form $\partial_t\boldsymbol{y}(t) = \boldsymbol{v}(\boldsymbol{y}(t))$ in $[t,t+\delta t)$ with final condition $\boldsymbol{y}(t+\delta t) = \boldsymbol{x}$ using a second-order Runge–Kutta scheme.

Aside from integrating the PDEs in time, we need to apply gradient and divergence operators to evaluate $\boldsymbol{g}$ in (2) and to solve (3) for λ. CLAIRE uses finite difference (**FD**) operators for these differential operators [21]. The reduced gradient (2) also involves the vector-Laplacian $\mathcal{A}$ and a Leray(-type) projection (see [19]). These operators are implemented in the spectral domain since (*i*) as we will see below, the solver requires the application of the inverse of $\mathcal{A}$ and (*ii*) the Leray projection also involves the inverse of a Laplacian operator. In spectral methods, inverting higher order differential operators can be done at the cost of two FFTs and one a Hadamard product. Using a different scheme would introduce significant complications.

Gauss–Newton–Krylov Solver: CLAIRE uses a Gauss–Newton–Krylov method globalized with an Armijo line search. The iterative scheme is given by

$$\mathbf{v}_{k+1} = \mathbf{v}_k + \alpha_k\tilde{\mathbf{v}}_k, \quad \mathbf{H}\tilde{\mathbf{v}}_k = -\mathbf{g}_k, \quad k = 0,1,2,\ldots \tag{4}$$

where $\mathbf{H}\in\mathbb{R}^{3N,3N}$ is the discretized reduced-space Hessian operator, $\tilde{\mathbf{v}}_k \in \mathbb{R}^{3N}$ the search direction, $\mathbf{g}_k \in \mathbb{R}^{3N}$ a discrete version of the gradient in (2), $\alpha_k > 0$ a line search parameter, and $k\in\mathbb{N}$ the Gauss–Newton iteration index. We have to solve the linear system in (4) at each Gauss–Newton step. We do not form or assemble $\mathbf{H}$; we use a matrix-free preconditioned conjugate gradient (**PCG**) method. This only requires an expression for applying the Hessian matrix to a vector (***Hessian matvec***). In the continuum, the Gauss–Newton approximation of this matvec is given by

$$\mathcal{H}\tilde{\boldsymbol{v}} = \beta\mathcal{A}\tilde{\boldsymbol{v}}(\boldsymbol{x}) + \int_0^1 \tilde{\lambda}(\boldsymbol{x},t)\nabla m(\boldsymbol{x},t)\,\mathrm{d}t. \tag{5}$$

To evaluate this matvec we have to find $\tilde{\lambda}$. Likewise to evaluating the gradient in (2), this necessitates the solution of two PDEs backward and forward in time, namely

$$\partial_t\tilde{m}(\boldsymbol{x},t) + \boldsymbol{v}(\boldsymbol{x})\cdot\nabla\tilde{m}(\boldsymbol{x},t) + \tilde{\boldsymbol{v}}(\boldsymbol{x})\cdot\nabla m(\boldsymbol{x},t) = 0 \tag{6}$$

in $\Omega\times(0,1]$ and

$$-\partial_t\tilde{\lambda}(\boldsymbol{x},t) - \nabla\cdot\tilde{\lambda}(\boldsymbol{x},t)\boldsymbol{v}(\boldsymbol{x}) = 0 \quad \text{in } \Omega\times[0,1) \tag{7}$$

with initial and final conditions $\tilde{m}(\boldsymbol{x},t) = 0$ in $\Omega\times\{0\}$ and $\tilde{\lambda}(\boldsymbol{x},t) = -\tilde{m}(\boldsymbol{x},t)$ in $\Omega\times\{1\}$, respectively. Inverting $\mathbf{H}$ in (4) is the most expensive part of CLAIRE. We propose a new preconditioner for the reduced-space system in (4) to amortize the computational costs.

Preconditioning: As we can see in (5), the Hessian operator consists of two terms. In a discrete setting, we have $\mathbf{H} = \mathbf{A} + \tilde{\mathbf{H}}$. Here, $\mathbf{A}\in\mathbb{R}^{3N,3N}$ corresponds to the regularization operator $\mathcal{A}$ and forming $\tilde{\mathbf{H}}\in\mathbb{R}^{3N,3N}$ involves $3N$ solutions of (6) and (7). Given that each Hessian matvec involves two PDE solves, we have to keep the number of PCG iterations as small as possible. With this in mind, we propose a new preconditioner.

As a benchmark, we consider a spectral preconditioner Inv$\mathcal{A}$ based on the inverse of $\mathbf{A}$—a common choice in PDE-constrained optimization [60]–[62] and the default option in CLAIRE [22]–[24]. This preconditioner is given by

$$\mathbf{s} = (\beta\mathbf{A})^{-1}\mathbf{r}, \tag{8}$$

where $\mathbf{r}$ is the residual of the Krylov solver. The cost of applying $(\beta\mathbf{A})^{-1}$ to a vector is two FFTs and a Hadamard product in spectral space.

The proposed preconditioner is based on a zero-velocity approximation of $\mathbf{H}$. This allows us to evaluate the Hessian matvec without having to solve (6) and/or (7). We term this preconditioner Inv$\mathcal{H}_0$. For $\mathbf{v} = \mathbf{0}$, the reduced-space Hessian system in (4) becomes $\mathbf{H}_0\tilde{\mathbf{v}} = -\mathbf{g}$, where $\mathbf{H}_0 := (\beta\mathbf{A} + \nabla\mathbf{m}_0 \otimes \nabla\mathbf{m}_0)$. Here, $\mathbf{m}_0$ is a discrete representation of the template image and $\otimes$ denotes the outer product. It is important to notice that $\mathbf{m}_0$ does not change during the course of the iterations. We use an (approximate) inverse of $\mathbf{H}_0$ as a preconditioner. To compute the action of $\mathbf{H}_0^{-1}$ we iteratively solve the linear system

$$(\beta\mathbf{A} + \nabla\mathbf{m}_0 \otimes \nabla\mathbf{m}_0)\mathbf{s} = \mathbf{r} \tag{9}$$

using a matrix-free PCG method with a relative tolerance $\epsilon_{\mathcal{H}_0}\epsilon_K$. Here, $\epsilon_K > 0$ is the tolerance for the outer PCG and $\epsilon_{\mathcal{H}_0} \in (0, 1)$. (We need to use a smaller tolerance in the inner PCG since the preconditioner would not act as a linear operator otherwise. We set $\epsilon_{\mathcal{H}_0}$ to $1\text{e}{-3}$ for the NIREP data and to $1\text{e}{-2}$ for the CLARITY data for our runs (see Table VI). These values were determined by experimentation in an attempt to obtain optimal runtimes per type of dataset.)

To compute the inverse of $\mathbf{H}_0$ efficiently, we propose several twists. First, we left-precondition $\mathbf{H}_0$ in (9) with $(\beta\mathbf{A})^{-1}$ (this adds vanishing computational costs; see above). Second, since $\mathbf{H}_0$ represents a zero-velocity approximation to $\mathbf{H}$, we expect the performance of the preconditioner to deteriorate as we iterate. As a remedy, we replace $\mathbf{m}_0$ in (9) with the deformed template image obtained for the current iterate $\mathbf{v}_k$ at the beginning of each Gauss-Newton iteration. Third, to further amortize the computational costs, we consider a second variant of Inv$\mathcal{H}_0$ that exploits a coarse grid discretization. We term this variant 2LInv$\mathcal{H}_0$. Here, we invert $\mathbf{H}_0$ on a coarse grid with half the resolution of the fine grid. We restrict the residual $\mathbf{r}$ and $\nabla\mathbf{m}_0$ in (9). The restriction and prolongation operators are implemented in the spectral domain. 2LInv$\mathcal{H}_0$ operates only on the low frequency components of $\mathbf{r}$. The solution of the iterative solver, $\mathbf{s}_c$, found on the coarse grid is prolonged to the fine grid and added to the filtered high frequency part of the original residual on the fine grid. In this context, the left-preconditioner $(\beta\mathbf{A})^{-1}$ can be viewed as a (poor) approximation of a multi-grid smoother. Algorithm 1 gives an overview of the two proposed preconditioner variants.

We observed that the performance of Inv$\mathcal{H}_0$ deteriorates for vanishing β. We found by experimentation that, if we use a lower bound of $5\text{e}{-2}$ for β in (9), the preconditioner remains effective even for vanishing βs for the overall problem. That is, if $\beta < 5\text{e}{-2}$, we set β in (9) to $5\text{e}{-2}$.

Finally, the suggested setting for `CLAIRE` is to use a β-continuation scheme for the solution of the inverse problem (1) [20], [21], [23]. That is, `CLAIRE` solves the registration problem for a vanishing sequence of values for β. For each new value, the velocity obtained at the former step is used as an initial guess for the Gauss-Newton-Krylov solver. For large β, the problem is dominated by the regularization operator $\mathbf{A}$. As a consequence, the problem is not only easy to solve but the spectral preconditioner is also quite effective. Therefore, if `CLAIRE` is executed using a β-continuation scheme we use Inv$\mathcal{A}$ for $\beta > 5\text{e}{-1}$ and switch to either variant, Inv$\mathcal{H}_0$ or 2LInv$\mathcal{H}_0$, for $\beta \leq 5\text{e}{-1}$ (this bound has been determined by experimentation).

Algorithm 1: Algorithmic overview of the two variants of the Inv$\mathcal{H}_0$ preconditioner.

```
func INVH0PC(r)
    s_f ← (βA)^{-1} r,   tol ← ε_{H_0} ε_K
    s_f ← run CG(H_0, s_f, (βA)^{-1}, tol)        ▷ solve (9)
    return s_f

func TWOLVLINVH0PC(r)
    s_f ← (βA)^{-1} r,   tol ← ε_{H_0} ε_K
    s_c ← RESTRICT(s_f)
    s_c ← run CG(H_{0,c}, s_c, (βA)^{-1}, tol)    ▷ solve (9) on coarse grid
    s_f ← PROLONG(s_c) + HIGHPASS(s_f)
    return s_f
```

III. Computational Kernels

In this section, we describe the multi-node multi-GPU implementation of our computational kernels. In Algorithm 2, we summarize the overall algorithm. We identify the three most important kernels and their overall contribution to the computational cost: interpolation (**IP**), finite differences (**FD**), and fast-Fourier transforms (**FFT**s). The costs of solving $\mathbf{g}(\mathbf{v}) = \mathbf{0}$ (first-order optimality conditions, where $\mathbf{g}$ is a discrete version of (2)) for $\mathbf{v}(\boldsymbol{x})$ are

$$c_{\text{total}} \approx n_{\text{GN}}\left(n_{\text{CG}}\left(2c_{\text{PDE}} + c_{\mathbf{H}} + c_{\text{PC}}\right) + 2c_{\text{PDE}}\right), \tag{10}$$

where n_{GN} is the number of Gauss–Newton iterations, $n_{\text{CG}}(2c_{\text{PDE}} + c_{\mathbf{H}} + c_{\text{PC}})$ summarizes the cost of computing the Gauss-Newton step in (4), n_{CG} is the number of PCG iterations per Gauss-Newton step (assuming that it is constant to simplify the analysis). The cost for evaluating (5) is denoted by $c_{\mathbf{H}}$. The cost c_{PC} is for the application of the preconditioner (e.g., iteratively solving (9)). c_{PDE} is a prototypical cost for solving the forward or adjoint equations; in particular, (6) and (7). Let c_{FD} denote the cost for the FD gradient and c_{IP} the cost for evaluating the IP kernel for a scalar field, then c_{PDE} for the RK2 implementation of the semi-Lagrangian scheme is $\mathcal{O}(N_t(c_{\text{FD}} + 4c_{\text{IP}}))$ for (6) (if we choose to not store the gradient of the state variable during the solution of (1b)) and $\mathcal{O}(N_t c_{\text{IP}})$ for (7). The remaining $2c_{\text{PDE}}$ in (10) are for evaluating the objective functional (1) (which involves the solution of (1b)) and the solution of the adjoint problem in (3). The cost $c_{\mathbf{H}}$ for evaluating (5) is dominated by $2c_{\text{FFT}}$ for applying the regularization operator in the spectral domain (or its inverse) and $N_t c_{\text{FD}}$ (if we choose to not store the gradient of the state variable). The cost for the preconditioner c_{PC} depends on the choice of the preconditioner. That is, c_{PC} is $\mathcal{O}(2c_{\text{FFT}})$ for Inv$\mathcal{A}$, $\mathcal{O}(2c_{\text{FFT}}n_{\text{CG,PC}})$ for Inv$\mathcal{H}_0$, and $\mathcal{O}\left(2c_{\text{FFT}}\frac{1}{8}\left(2c_{\text{FFT}}n_{\text{CG,PC}}\right)\right)$ for

2LInv$\mathcal{H}_0$, where $n_{\text{CG, PC}}$ is the number of PCG iterations to compute the action of the inverse of $\mathbf{H}_0$. (We kept some of the constant factors to explicitly document the computational steps.). The computational and communication components of c_{IP}, c_{FD} and c_{FFT} are reported in §III-A, §III-B and §III-C, respectively. We refer to [21], where a DRAM based (ignoring cache heirarchy) roofline analysis is performed for the IP and FD kernels (on a single GPU). DRAM memory accesses for each kernel are modelled analytically assuming full reuse. The number of floating point operations are also estimated analytically. The arithmetic intensity, which is defined as the ratio of total number of floating point operations to number of bytes accessed, is assessed based on this model. The analytical value is compared with the experimental value obtained by the NVIDIA profiler. It is found that both kernels are bound by the GPU DRAM bandwidth.

The work in [21] discusses several technical optimizations beyond a pure transition to GPUs, in particular, several options for the IPs as the most important kernel in the semi-Lagrangian solver. In addition, [21] suggests to replace FFTs used in [20] for first order derivatives by FD approximations. In [21], it is shown empirically that this does not deteriorate the accuracy if FD kernels of high enough order are used. In the following, we describe the implementation of different variants of these kernels, which includes optimizations compared to the work in [21] for efficient execution on a multi-node multi-GPU architecture.

Algorithm 2: Overview of the Gauss–Newton–Krylov solver implemented in `CLAIRE`.

$\mathbf{v} \leftarrow \mathbf{v}_{init}, \quad \epsilon_N \leftarrow 5\text{e}{-2}$
run NEWTONSOLVER$(\mathbf{v}, \epsilon_N)$
  $\mathbf{m} \leftarrow$ SOLSTATEEQ$(\mathbf{v}, \mathbf{m}_0)$ ▷ solve (1b)
  $\lambda \leftarrow$ SOLADJOINTEQ$(\mathbf{v}, \mathbf{m}, \mathbf{m}_1)$ ▷ solve (3)
  $\mathbf{g} \leftarrow$ EVALGRAD$(\mathbf{v}, \mathbf{m}, \lambda)$ ▷ evaluate (2)
  $\epsilon_K \leftarrow \min(\sqrt{\|\mathbf{g}\|_{rel}}, 0.5)$
  run PCG(MATVEC$, -\mathbf{g}, \epsilon_K)$ ▷ solve (4)
    MATVEC$(\tilde{\mathbf{v}})$
      $\tilde{\mathbf{m}} \leftarrow$ SOLINCSTATEEQ$(\mathbf{v}, \tilde{\mathbf{v}}, \mathbf{m})$ ▷ solve (6)
      $\tilde{\lambda} \leftarrow$ SOLINCADJOINTEQ$(\mathbf{v}, \tilde{m})$ ▷ solve (7)
      $\mathbf{H}\tilde{\mathbf{v}} \leftarrow$ EVALMATVEC$(\tilde{\mathbf{v}}, \mathbf{m}, \tilde{\lambda})$ ▷ eval (5)
      $\mathbf{r} \leftarrow -\mathbf{g} - \mathbf{H}\tilde{\mathbf{v}}$
    APPLYPRECOND$(\mathbf{r})$
      see Algorithm 1
  run LINESEARCH(α)
    $m \leftarrow$ SOLSTATEEQ$(\mathbf{v} + \alpha\tilde{\mathbf{v}}, \mathbf{m}_0)$ ▷ solve (1b)
    EVALOBJECTIVE$(\mathbf{v} + \alpha\tilde{\mathbf{v}}, \mathbf{m})$ ▷ eval (1a)
  $\mathbf{v} \leftarrow \mathbf{v} + \alpha\tilde{\mathbf{v}}$ ▷ Newton step

The total memory consumption mostly depends on the domain size $N = N_1N_2N_3$. The state variable $m(\boldsymbol{x}, t)$ has to be stored for all time steps to avoid additional PDE solves. The memory footprint for the proposed method is

$$\begin{aligned}\mu_{\text{total}} &\approx \mu_{\text{PDE}} + \mu_{\text{FFT}} + \mu_{\text{FD}} + \mu_{\text{SL}} + \mu_{\text{GN/CG}} + \mu_{\text{IP}} + \mu_{\text{API}} \\ &= ((24 + N_t) + 7 + 2 + 11 + 30)^{N\mu_0}/_p + \mu_{\text{IP}} + \mu_{\text{API}} \\ &= (74 + N_t)^{N\mu_0}/_p + \mu_{\text{IP}} + \mu_{\text{API}},\end{aligned}$$

where μ_0 is word size of the datatype (i.e. 4 byte for single precision floating point values). The memory required for the ghost layer communication in the IP model is $\mu_{\text{IP}} \approx 30dN_2N_3\mu_0$ with polynomial degree d. Note that the runtime API overhead, μ_{API}, depends on N (especially for `cuFFT` [63] and `PETSc` [64], [65]), but is not further estimated.

A. Interpolation

The semi-Lagrangian scheme requires IP of vector and scalar fields along backward characteristics. We use Lagrange polynomial-based cubic IP but also consider first-order trilinear IP since GPUs offer hardware acceleration through texture units (not fully single-precision). The formula for interpolating at an off-grid query point $\boldsymbol{x} = (x_1, x_2, x_3)$ is given by

$$f(\boldsymbol{x}) = \textstyle\sum_{i,j,k=0}^{d} f_{ijk}\phi_i(x_1)\phi_j(x_2)\phi_k(x_3),$$

where f_{ijk} is the function value at a grid point, d is the polynomial order and ϕ_l, $l = 0, \ldots, d$, are the Lagrange polynomial basis functions. The numerical accuracy and compute performance of variants of the IP kernel on a single GPU have been discussed in [21]. We focus on optimizations for the multi-GPU implementation. We follow the workflow described in [22], [24] with the following major modifications:
1) We use CUDA-aware MPI to reduce or eliminate expensive on-node host-device transfers.
2) We use the `thrust` library [66] to efficiently determine, which query points need to be processed by which GPU, thereby completely eliminating host-side computation.
3) We use a sparse point-to-point communication to send points on the backward characteristics to other processors, as proposed in [22]. We adaptively allocate memory for the respective MPI send and receive buffers using an estimate of the maximal displacement of grid points along backward trajectories based on the CFL number of the velocity field.
4) Following [21], we perform local IP on a single GPU using GPU-TXTLAG or GPU-TXTLIN (for high-resolution images). Although GPU-TXTSPL in [21] is much faster than GPU-TXTLAG on a single GPU, for the distributed memory implementation it requires ghost layer communication for the pre-filtering step, which makes it slower than GPU-TXTLAG.

The computational cost c_{IP} of applying the IP kernel GPU-TXTLAG is $\mathcal{O}(^{482N}/_p)$ (see [21]), where p is the number of processors and $N = N_1N_2N_3$. For GPU-TXTLIN, it is $\mathcal{O}(^{30N}/_p)$. The total cost of communicating ghost points, query points and interpolated values is $\mathcal{O}(u_{\max}N_2N_3)$ where $u_{\max} \in \mathbb{R}$ is an estimate of the maximum displacement of a voxel from a regular grid point along the coordinate directions. For the IP kernel we do not consider overlapping communication and computation because of the data dependencies in the semi-Lagrangian scheme.

We perform a weak scaling experiment for an isolated semi-Lagrangian solve on a real dataset and present the runtime breakdown in Table II. We use a realistic velocity field for this experiment (obtained by registration of two brain images) to ensure a representative scenario for the communication of query points between MPI ranks. The major **observations** are:

1) Since we use slab decomposition in x_1-dimension, the message size for `ghost_comm` is $\mathcal{O}(N_2N_3)$. Hence, it roughly doubles every time N_2 or N_3 is doubled.
2) We see a similar increase for `interp_comm` and `scatter_comm`. Due to the non-uniformity in space of the query points, communication time does not double exactly and we observe an imbalance in the communication for different MPI ranks.
3) The time spent in `interp_kernel` is almost the same across all cases and takes up the majority of the time for up to 16 GPUs. Beyond 16 GPUs, communication dominates the overall runtime.
4) Since we are performing scattered IP, determining which and how many query points need to be processed locally or sent to other MPI ranks in `scatter_mpi_buffer` leads to expensive scattered memory accesses.[1] This explains why `scatter_mpi_buffer` requires almost one third of `interp_kernel` runtime.

B. Finite Differences

The CPU version of `CLAIRE` uses FFTs for spatial derivatives [20], [22], [24]. Since our functions are periodic, these spectral operators are diagonal. [21] proposes a mixed-accuracy implementation that replaces the spectral discretization of the divergence and gradient operators with a FD scheme. This mixed scheme is more accurate (for the considered grid sizes—not asymptotically) and faster than differentiation via FFTs. In particular, an 8^{th} order central difference scheme is used. We extend the single-GPU FD kernel described in [21] to a multi-node multi-GPU environment. The computational cost c_{FD} of applying the FD kernel is $\mathcal{O}(^{20N}/_p)$, where p is the number of processors and $N = N_1N_2N_3$. To compute derivatives at the boundary of our 2D slab decomposition, we communicate a ghost layer of size $\mathcal{O}(N_2N_3)$ to neighboring MPI ranks. We perform strong and weak scaling experiments for computing the gradient of a synthetic scalar field; see Table III. For a single GPU, no communication is involved. It is much faster than using multiple GPUs (for small problem sizes). In the weak scaling setup, the runtime increases when we switch from one to eight to 64 GPUs because the size of the ghost layer increases (N_2 and N_3 increase), while the kernel execution time itself remains constant. In the strong scaling setting, the kernel scales well for up to 8 GPUs. Beyond 8 GPUs, the kernel execution time becomes much smaller than the communication time (which is constant); this negatively impacts the scalability. Since the FD kernel is not a bottleneck—as seen in Table VII—we did not explore the idea of overlapping communication and computation when evaluating the kernel.

C. FFT

The distributed memory implementation of `CLAIRE` [20], [22], [24] uses `AccFFT` [67], [68], which supports MPI for CPUs and GPUs. In [21], `cuFFT` [63] is used, as they focus on a single-GPU implementation. Higher order derivatives and their inverses require 3D FFTs. `AccFFT` uses a pencil decomposition (see, e.g., [24]), which is efficient for 1D FFTs (needed for divergence and gradient operators). In [21], 1^{st} order derivatives have been replaced by FD kernels. For the proposed multi-GPU implementation, we use a combination of `cuFFT` and a new 2D slab decomposition, which allows us to use the highly optimized 2D `cuFFT` on each GPU. We decompose the spatial domain in the outer-most dimension (i.e., x_1) and in the spectral domain in x_2 dimension. Thus, the inner-most x_3 dimension is always continuous in memory. This reduces misaligned memory accesses for communication and transpose operations. The real-to-complex transformation is divided into three steps. (*i*) We use `cuFFT`'s batched 2D FFTs in the x_2–x_3 plane. (*ii*) The complex data are transposed to a decomposition in x_2 dimension. (*iii*) We apply `cuFFT`'s batched 1D FFTs to the x_1 dimension, which is non-continuous in memory. For the inverse complex-to-real transformation, these three steps are executed in reverse order, using the respective inverse transformations. The complexity for communication of the 2D slab decomposition is $\mathcal{O}(^N/_P - {}^N/_{P^2})$ per process. If the FFT is executed on a single rank, we still use `cuFFT`'s 3D FFT to avoid additional operations, in particular an explicit transpose operation on the data and misaligned memory accesses. Also, it reduces the number of memory accesses of the spectral data from device memory.

For communication between GPUs, we use CUDA-aware MPI. We found that `MPI_Alltoallv` (IBM Spectrum MPI 10.3 [69]) is not optimized for direct GPU communication. For communication volumes larger than $\sim 500\,\text{kB}$, all-to-all communication using direct GPU-optimized peer-to-peer routines is faster on our test system (see Table IV). We implement a threshold of $512\,\text{kB}$ to switch between an asynchronous peer-to-peer communication scheme or `MPI_Alltoallv`. For FFTs on a single node (four GPUs), we always use the peer-to-peer scheme to utilize the *NVLink* inter-GPU bus. The communication is only overlapped with the process-local transpose operation due to data dependencies.

In addition to the memory footprint of `cuFFT` our 2D slab decomposition needs twice the local domain size to execute an out-of-place transformation. The temporary memory consumtion of `cuFFT` is between $^{2N}/_p$ and $^{16N}/_p$ real valued elements [63]. Table V shows that our 3D FFT with 2D slab decomposition is almost as fast as `cuFFT` 3D-FFT, but can be accelerated and scaled to data sizes beyond the memory capacity of a single GPU. Given the $\mathcal{O}(N \log N)$ computational complexity of the FFT (with data size N) and the huge amount of data communication inherent to FFTs, we observe good scalability up to 128 GPUs, for the large problem sizes—even in strong scaling.

IV. Results

We (*i*) analyze the numerical and runtime efficiency of our new preconditioner and (*ii*) assess the overall scalability and efficiency of our multi-GPU multi-node implementation.

[1] We rely on the `thrust::copy_if` algorithm for this purpose.

Table II: *Weak scaling study for the IP kernel. We report runtimes for the semi-Lagrangian scheme. We advect a real brain MRI (`na10` of the NIREP data; see §IV) with a velocity field obtained from the registration of `na10` to `na01`. We use cubic IP GPU-TXTLAG and $N_t = 4$ time steps. We report the runtime (in seconds) of the major components in the algorithm and their percentages with respect to the total runtime. These components are `ghost_comm` (communication of ghost points), `interp_comm` (communication of interpolated values), `scatter_comm` (communication of query points), `interp_kernel` (IP kernel), `scatter_mpi_buffer` (creation of MPI buffer for sending query points to other ranks). The experiments were performed on TACC's Longhorn system with four Nvidia V100 GPUs per node and a single GPU per MPI rank. We scale from a single GPU to 64 GPUs for grid resolutions ranging from 256^3 to 1024^3.*

size	256×256×256		512×256×256		512×512×256		512×512×512		1024×512×512		1024×1024×512		1024×1024×1024	
#GPUs	1	%	2	%	4	%	8	%	16	%	32	%	64	%
`ghost_comm`	0.00	0.0	2.48e−3	7.6	3.49e−3	9.9	7.51e−3	18.0	8.66e−3	19.1	1.31e−2	24.0	2.23e−2	31.3
`interp_comm`	0.00	0.0	1.71e−3	5.2	1.80e−3	5.1	3.62e−3	8.7	4.17e−3	9.2	5.92e−3	10.9	9.73e−3	13.6
`scatter_comm`	0.00	0.0	2.65e−4	0.8	7.81e−4	2.2	2.02e−3	4.8	2.85e−3	6.3	5.42e−3	10.0	8.72e−3	12.2
`interp_kernel`	1.77e−2	93.3	1.79e−2	54.8	1.76e−2	49.8	1.76e−2	42.0	1.83e−2	40.2	1.84e−2	33.9	1.87e−2	26.2
`scatter_mpi_buffer`	0.00	0.0	5.88e−3	18.0	7.16e−3	20.3	6.63e−3	15.9	6.98e−3	15.4	7.00e−3	12.9	7.30e−3	10.2
total	1.90e−2	100.0	3.28e−2	100.0	3.53e−2	100.0	4.18e−2	100.0	4.54e−2	100.0	5.44e−2	100.0	7.13e−2	100.0

Table III: *Scalability for our finite difference (FD) scheme for first order derivatives. We show strong scaling for 512^3 from one to eight MPI ranks, and weak scaling for 256^3 to 1024^3 from one to 64 MPI ranks. We report the breakdown of runtime (in seconds) into `comm` (communication of ghost points) and `kernel` (FD kernel) and show percentages with respect to total runtime.*

#GPUs	size	comm	%	kernel	%	total
1	256^3	0.0	0.0	6.32e−4	100.0	6.32e−4
1	512^3	0.0	0.0	4.82e−3	100.0	4.82e−3
2	512^3	9.37e−4	21.9	3.33e−3	78.1	4.27e−3
4	512^3	7.01e−4	29.2	1.70e−3	70.8	2.40e−3
8	512^3	9.86e−4	53.2	8.66e−4	46.8	1.85e−3
16	512^3	8.94e−4	66.0	4.60e−4	34.0	1.35e−3
64	1024^3	2.85e−3	76.0	9.03e−4	24.0	3.76e−3

We use the following datasets: **1) SYN** is a synthetic test problem, where the template image is $m_0(\boldsymbol{x}) := \sum_{i=1}^{3} \sin^2(x_i)/3$ and the reference image $m_1(\boldsymbol{x})$ is computed by solving (1b) with initial condition $m_0(\boldsymbol{x})$ and given velocity $\boldsymbol{v}(\boldsymbol{x}) := (\sin(x_i), \cos(x_k), \sin(x_k))_{(i,k)=(3,2),(1,3),(2,1)}$. **2) NIREP** [13] is a standardized repository for assessing registration accuracy that contains 16 T1-weighted MR neuroimaging datasets (`na01`–`na16`) of different individuals (see Figure 1). The original image size is 256×300×256 voxels. **3) CLARITY** [7]–[12], [70] are biomedical imaging datasets with a resolution of 0.60 µm×0.60 µm×6 µm and a grid size at the order of 20 K×20 K×1 K (see Figure 2) . We have affinely pre-registered these datasets (at a much lower resolution) using `FAIR` [71] prior to executing `CLAIRE`.

All runs were executed on ***TACC's Longhorn system*** in single precision. Longhorn hosts 96 NVIDIA Tesla V100 nodes. Each node is equipped with four GPUs with 4×16 GB GPU RAM (64 GB aggregate) and two IBM Power 9 processors with 20 cores (40 cores per node) at 2.3 GHz with 256 GB memory. Our implementation uses `PETSc` [64], [65] for linear algebra, `PETSc`'s `TAO` package for the nonlinear optimization, `CUDA` [72], `thrust` [66], `cuFFT` for FFTs [63], `niftilib` [73] for I/O, IBM Spectrum MPI [69], and the IBM XL compiler [74].

Table IV: *MPI performance analysis for the proposed FFT kernel. We use one GPU per MPI rank. We report the sustained bidirectional CUDA MPI bandwidth in GB/s. The results are averaged over ten runs and the smallest value for all ranks is presented. We compare `MPI_Alltoall` to our own implementation using asynchronous peer-to-peer routines. The local data size per rank is $8N_1N_2(\lfloor N_3/2 \rfloor + 1)/p$ byte. The peer-to-peer communication volume is $8N_1N_2(\lfloor N_3/2 \rfloor + 1)/p^2$ byte. Runs in the shaded cells have a communication volume larger than 512 kB. The fastest runs are highlighted in bold.*

setup		MPI tasks					
size	type	4	8	16	32	64	128
256^3	MPI	5.6	5.0	**3.3**	**2.2**	**2.0**	**1.5**
	P2P	**35.7**	**9.3**	2.2	1.3	1.6	1.4
512×256^2	MPI	5.1	5.2	3.5	**1.5**	**1.9**	**1.9**
	P2P	**36.0**	**9.5**	**5.8**	1.0	1.5	1.4
$512^2\times256$	MPI	5.4	4.6	3.5	**2.8**	1.6	**2.7**
	P2P	**36.6**	**9.9**	**6.1**	0.4	**1.7**	1.4
512^3	MPI	5.9	4.9	3.9	2.7	**2.5**	**2.7**
	P2P	**37.1**	**9.5**	**5.9**	**4.7**	0.5	1.5
1024×512^2	MPI	6.4	5.4	3.9	3.4	**3.2**	**2.2**
	P2P	**32.6**	**10.1**	**5.9**	**4.8**	0.4	0.5
$1024^2\times512$	MPI	6.7	5.5	4.2	3.6	3.4	**2.7**
	P2P	**36.6**	**10.5**	**5.4**	**4.7**	**4.5**	0.3
1024^3	MPI	6.7	5.6	4.4	3.7	3.4	**3.1**
	P2P	**36.8**	**10.6**	**5.2**	**4.6**	**4.3**	0.4

A. Preconditioning

We study different preconditioner variants. We use the datasets `na02`, `na03`, and `na10` from the NIREP repository as template images, and `na01` as reference image.

Results: We report convergence plots for a single Gauss-Newton step in Figure 3. We initialize the solver with `na10` as template and a reference image synthetically generated by solving the forward problem with a true registration velocity (`na10` to `na01`). The true (non-zero) velocity is used as an initial guess for the Gauss-Newton-Krylov method (i.e., we solve (4) at the solution of the inverse problem). This allows

Table V: *Weak (diagonals) and strong (rows) scaling for the proposed 3D FFT kernel in slab decomposition (forward and inverse). We use one GPU per MPI rank. We report the runtime in ms. The FFT uses CUDA-aware MPI. We switch from point-to-point communication to* `MPI_Alltoall` *for small slabs. Results are averaged over 20 runs. For a single rank, the runtime is also given for* `cuFFT` *3D-FFTs (3D). The highlighted runs use peer-to-peer communication.*

		MPI tasks						
size	3D	1	4	8	16	32	64	128
256^3	1.41	1.86	**2.83**	**3.92**	4.17	3.88	2.93	3.76
$512{\times}256^2$	3.20	3.87	**5.39**	**7.65**	**7.33**	5.21	4.09	4.30
$512^2{\times}256$	7.30	7.70	**8.48**	**13.8**	**13.3**	8.29	5.67	5.12
512^3	16.9	16.9	**15.6**	**25.7**	**24.5**	**16.7**	9.63	7.23
$1024{\times}512^2$	31.2	40.1	**31.8**	**51.3**	**43.6**	**31.3**	17.8	11.8
$1024^2{\times}512$	—	—	**65.7**	**100**	**90.5**	**54.2**	**33.4**	21.4
1024^3	—	—	**132**	**198**	**182**	**116**	**62.0**	38.4

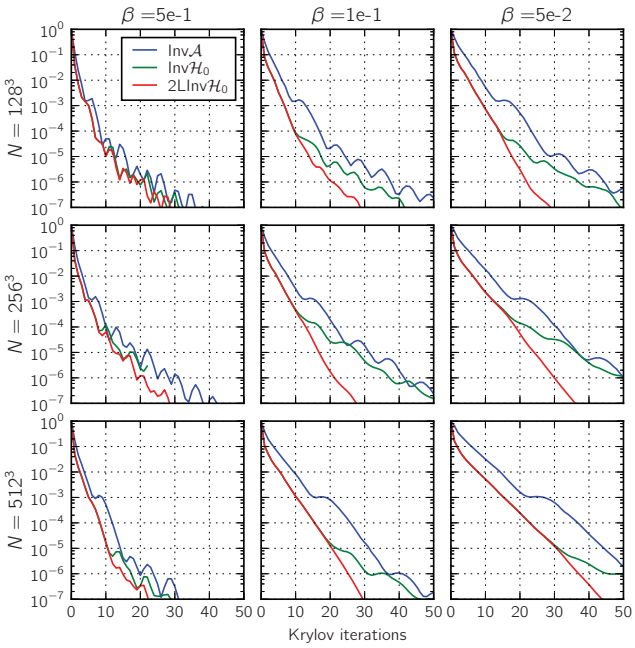

Fig. 3: *We report the trend of the PCG residual versus PCG iterations for the benchmark preconditioner Inv$\mathcal{A}$ used in [20], [21] and the proposed preconditioner variants Inv$\mathcal{H}_0$ and 2LInv$\mathcal{H}_0$. We vary the regularization parameter β (columns; $\beta \in \{5\text{e}{-1}, 1\text{e}{-1}, 5\text{e}{-2}\}$) and the domain size N (rows; $N \in \{128^2, 256^3, 512^3\}$). We solve the problem at the true solution (see text for a description).*

us to assess (*i*) the convergence at a point in the optimization landscape at which we expect the PCG to take many iterations and (*ii*) identify potential issues that may arise due to a zero-velocity approximation at a point at which the velocity is non-zero.We report results for varying grid sizes and values for β.

Observations: The proposed preconditioner leads to faster convergence (fewer iterations) and is less sensitive to a reduction in β than Inv$\mathcal{A}$. We expect the preconditioner to be mesh-independent but not β-independent. All preconditioners exhibit (close to) mesh independent behavior. Interestingly, for the considered range for β, 2LInv$\mathcal{H}_0$ is close to being β-independent; only for $\beta = 5\text{e}{-2}$ we see the performance slightly deteriorate as the mesh size increases. In general, we expect that we might have to use larger values for β for higher resolutions, since higher frequencies can occur in the images and the velocity field (coarsening can be viewed as an additional regularization).

B. Registration Performance

We study the performance of the proposed methods for the solution of the inverse registration problem. We report results for three different template images from the NIREP repository: `na02`, `na03`, and `na10`. For `na10`, we increase the resolution from 256^3 to 1024^3 (spectral prolongation). Results for the registration of the dataset `na10` to `na01` are shown in Figure 1. We expect the convergence behavior of the Gauss-Newton-Krylov method to be independent of the mesh size. In addition to that, we report results for the registration of two representative CLARITY volumes (dataset Cocaine 175 to Control 189; Control 189 is visualized in Figure 2). We consider all preconditioner variants.

Results: The results can be found in Table VI. We report the number of Gauss-Newton iterations, the accumulated number of PCG iterations across all Gauss-Newton iterations, the relative reduction of the mismatch, the relative reduction of the gradient, the number of applications of the inverse regularization operator, the number of applications of Inv$\mathcal{H}_0$ or 2LInv$\mathcal{H}_0$, the number of PCG iterations to invert $\mathbf{H}_0$ (in total and on average), the time spent in the core parts of the solver, and the total runtime. We visualize the runtime of the solver components in Figure 4.

Observations: The most important observation is that our solver converges quickly to accurate solutions. We require 14 to 22 Gauss-Newton-Krylov iterations. The number of Gauss-Newton-Krylov and PCG iterations is approximately mesh-independent. The most effective preconditioner is 2LInv$\mathcal{H}_0$. If we compare the runtime for our new version to the results reported in [21], we can observe a speedup of about 50%. The average time-to-solution for clinically relevant problems on a single GPU is ~5 s. We can reduce the runtime on a single GPU to 3.70 s, which corresponds to a speedup of 70% compared to [21] (for `na02`, 256^3) by storing the gradient of the state variable. Storing the gradient of the state variable reduces the runtime by approximately 15% (but increases the memory pressure). We can also observe that we can solve large-scale real-world imaging problems with grid sizes of 1024^3 for the NIREP data and up to $1024{\times}768{\times}768$ for the CLARITY data on 8 nodes with 32 GPUs or one 4 nodes with 16 GPUs, respectively. In terms of registration quality, we achieve the same accuracy as reported in [20], [21]. These studies also include comparisons to other LDDMM software packages. They demonstrated that their implementation of `CLAIRE` yields results that are significantly more accurate (in terms of data mismatch) than existing methods, and that the single-node GPU version of `CLAIRE` is up to 30× faster than

Table VI: *Results for the registration of different NIREP and CLARITY datasets. We report results for the different preconditioners Inv$\mathcal{A}$ **([A])**, Inv$\mathcal{H}_0$ **([B])**, and 2LInv$\mathcal{H}_0$ **([C])**. We use a parameter continuation scheme for β with target parameter $\beta = 5\text{e}{-4}$. The number of time steps for the semi-Lagrangian method is $N_t = 4, 8, 16$ for domain sizes $N = 256^3, 512^3, 1024^3$, respectively. All runs use linear IP and FD for 1^{st} order derivatives. For each domain size, we use the minimum number of resources possible, i.e., a single GPU for $N = 256^3$, four GPUs on a single node for $N = 512^3$, 32 GPUs on 8 nodes for $N = 1024^3$. We report from left to right: (**data**) the selected template image, (**PC**) the Hessian preconditioner method, (**GN**) the number of Gauss-Newton iterations, (**PCG**) the number of PCG iterations, (**mism.**) the relative mismatch, ($||\mathbf{g}||_{rel}$) the relative gradient norm, (**[A]**) the number of applications of Inv$\mathcal{A}$, (**[B\|C]**) the number of applications of Inv$\mathcal{H}_0$/2LInv$\mathcal{H}_0$ (notice, that we use Inv$\mathcal{A}$ for large values of β in our continuation scheme), (**total**) and (**average**) the number of PCG iterations to invert $\mathbf{H}_0$ (total and average), (**PC**) the overall runtime for preconditioner, (**Obj**) objective function evaluation, (**Grad**) reduced gradient computation, (**Hess**) Hessian matvecs, and (**Total**) the total runtime of the entire solver. All runtimes are in seconds.*

setting		solver				preconditioner				runtimes				
		iterations		relative accuracy		applications		CG steps						
data	PC	GN	PCG	mism.	$\|\|\mathbf{g}\|\|_{\text{rel}}$	A	B|C	total	avg.	PC	Obj	Grad	Hess	Total
NIREP $N = 256^3$, $N_t = 4$, $\epsilon_{\mathcal{H}0}$ =1e-3, 1 node, 1 GPU														
na02	[A]	14	75	2.73e−2	3.09e−2	75	—	—	—	4.43e−1	2.04e−1	4.33e−1	3.82	6.19
	[B]	14	23	2.62e−2	2.82e−2	3	20	235	11.8	2.45	2.04e−1	4.33e−1	1.27	5.54
	[C]	14	28	2.79e−2	3.23e−2	3	25	294	11.8	1.04	2.05e−1	4.35e−1	1.52	4.44
na03	[A]	17	93	2.55e−2	3.11e−2	93	—	—	—	5.50e−1	2.49e−1	5.24e−1	4.69	7.53
	[B]	17	36	2.50e−2	3.04e−2	14	22	255	11.6	2.72	2.48e−1	5.23e−1	1.91	6.80
	[C]	17	39	2.56e−2	3.17e−2	14	25	301	12.0	1.11	2.49e−1	5.24e−1	2.05	5.39
na10	[A]	17	94	1.96e−2	2.94e−2	94	—	—	—	5.58e−1	2.50e−1	5.25e−1	4.76	7.61
	[B]	17	36	1.90e−2	2.81e−2	9	27	299	11.1	3.17	2.48e−1	5.25e−1	1.91	7.25
	[C]	17	38	1.93e−2	2.90e−2	9	29	328	11.3	1.22	2.49e−1	5.26e−1	2.01	5.45
NIREP $N = 512^3$, $N_t = 8$, $\epsilon_{\mathcal{H}0}$ =1e-3, 1 node, 4 GPUs														
na10	[A]	18	107	2.53e−2	3.84e−2	107	—	—	—	5.28	1.68	3.86	3.52e1	5.18e1
	[B]	18	37	2.66e−2	4.38e−2	10	27	307	11.4	2.19e1	1.70	3.89	1.25e1	4.55e1
	[C]	18	37	2.68e−2	4.39e−2	10	27	309	11.4	5.55	1.67	3.87	1.25e1	2.92e1
NIREP $N = 1024^3$, $N_t = 16$, $\epsilon_{\mathcal{H}0}$ =1e-3, 8 nodes, 32 GPUs														
na10	[A]	21	128	3.19e−2	4.41e−2	128	—	—	—	4.63e1	3.55	2.14e1	1.76e2	2.55e2
	[B]	22	59	2.70e−2	3.34e−2	18	41	531	13.0	2.33e2	3.79	2.24e1	8.08e1	3.46e2
	[C]	22	59	2.73e−2	3.77e−2	18	41	533	13.0	5.69e1	3.80	2.24e1	8.11e1	1.71e2
CLARITY $N = 1024\times384\times384$, $N_t = 4$, $\epsilon_{\mathcal{H}0}$ =1e-2, 1 nodes, 4 GPUs														
	[A]	13	205	2.01e−1	4.23e−2	205	—	—	—	2.12e1	8.78e−1	2.53	5.11e1	7.13e1
	[C]	12	75	2.02e−1	4.54e−2	4	71	1007	14.2	1.67e1	8.49e−1	2.34	1.89e1	4.36e1
CLARITY $N = 1024\times768\times768$, $N_t = 4$, $\epsilon_{\mathcal{H}0}$ =1e-2, 4 nodes, 16 GPUs														
	[A]	20	663	1.95e−1	5.81e−2	663	—	—	—	1.96e2	4.02	1.37e1	5.12e2	7.38e2
	[B]	15	52	2.03e−1	4.38e−2	6	46	648	14.1	2.28e2	1.57	1.09e1	4.02e1	2.86e2

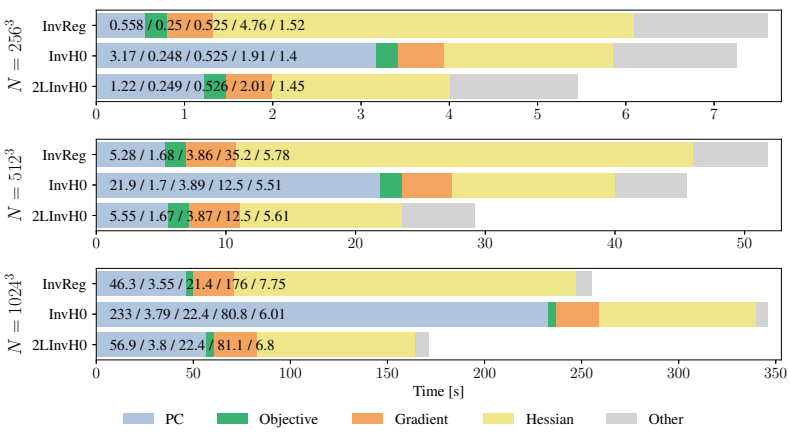

Fig. 4: *Visualization of the allocated runtime for the results reported in Table VI. The color bars (times are in seconds) illustrate the amount of execution time spent in the main mathematical operators of our solver (PC: application of inverse of preconditioner; objective: evaluation of the objective functional; gradient: evaluation of gradient (includes PDE solves for state and adjoint equation); hessian: Hessian matvecs (includes PDE solves for incremental state and adjoint equation). We can observe that we spend a large fraction of our runtime on the computation of the Newton step.*

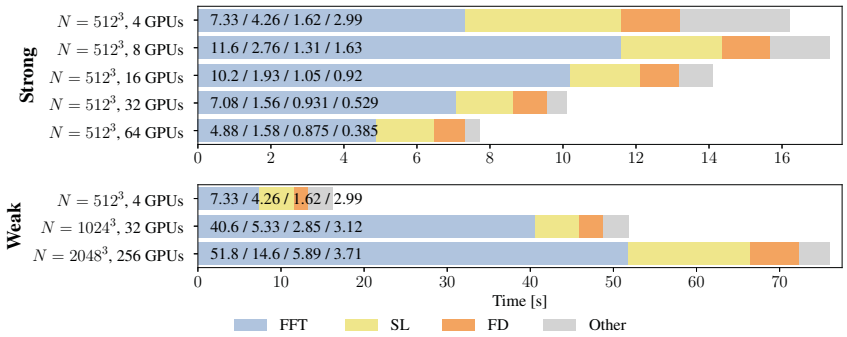

Fig. 5: *We visualize exemplary strong (top block) and weak (bottom block) scaling results for the experiments reported in Table VII. For each run, we report the fraction (color bars and runtime in seconds) spent in the individual kernels. We can see that the runtime is dominated by the FFT kernel. We can also observe that almost the entire runtime of our solver is spent in the three main computational kernels—FFTs, SL, and FD. The scalability of our multi-node multi-GPU implementation is limited due to the high communication costs for small local problem sizes and load imbalance across ranks. We provide a more detailed analysis in the text.*

other available single-GPU implementations. With the present work, we are $50\times$ faster on a single GPU.

C. Strong and Weak Scaling Results

We study weak and strong scaling for our new multi-node multi-GPU implementation. We consider the SYN dataset and use the Inv$\mathcal{A}$ preconditioner for these runs. We fix the number of Gauss-Newton iterations to 5 and the number of PCG iterations per Newton step to 10 to avoid discrepancies arising from the use of relative tolerances.

Results: We present the results in Table VII and report the time-to-solution along with the time spent in individual kernels. We additionally provide the % of the execution time spent for data communication and the total memory consumption per GPU. The strong and weak scaling experiments are restricted by the slab size and available GPU memory, respectively. For the memory restrictions we refer to the analytical estimates given above. Considering the domain decomposition, we cannot use arbitrarily many GPUs per problem size since the slab size (local data volume) per GPU becomes too small for the computations to be efficient. We visualize strong scaling for $N = 512^3$ and weak scaling in Figure 5.

Observations: The most important observations are (*i*) we can solve problems of unprecedented scale (the 1024^3 and the 2048^3 problem can not be solved on a single GPU; the largest problem solved in [21] is 384^3) and (*ii*) the scalability of our solver suffers from high communication costs for small local problem sizes. In particular, the runtime in FFTs is dominated by communication because of the required all-to-all collective. For a single GPU, we utilize the `cuFFT` 3D FFT and need no additional memory transfers. For small problem sizes (e.g., 128^3 or 256^3), the additional communication costs for strong scaling cannot be compensated by the reduced computations per rank. For all FFTs, scaling above a single node (4 ranks) increases the runtime due to off-node communication, which is the limiting factor. In Table II, we considered GPU-TXTLAG to test the scalability of the semi-Lagrangian method. However, here we use GPU-TXTLIN, which has much lower computational complexity. This results in an increased percentage of communication in the overall runtime, and as we reduce the local problem size (slab width <16 voxels), this effect is further amplified. At this slab size, the communication of the query points can become non-uniform (subject to local variations in length of the characteristics). This can cause a significant load imbalance among MPI ranks and by that negatively affects the scaling because of the implicit synchronization for the next communication step (which is ghost layer sharing). The scaling performance of the FD kernel is consistent with the results in Table III. For weak scaling, when switching from 512^3 on 4 GPUs to 2048^3 on 256 GPUs, the communication time increases by ~4x; the kernel execution time stays roughly the same. For the strong scaling for resolutions 512^3 and 1024^3, the communication time stays roughly the same while the kernel execution time reduces by ~2x. However, the overall time spent in FD does not scale well because of GPU memory constraints, as explained in §III-B.

V. Conclusions

We presented a novel multi-node multi-GPU implementation for diffeomorphic registration. Our work extends the publicly available software package `CLAIRE`. `CLAIRE` relies on three main computational kernels: FFTs and FD kernels for differentiation and the evaluation of IP kernels in a semi-Lagrangian solver for the solution of transport equations. Our approach to port these kernels to a multi-GPU environment is highly adapted to the target architecture in various ways: (*i*) We replace FFT-based (spectral) first-order derivative evaluations used in `CLAIRE` with an 8^{th} order FD scheme for the multi-GPU version. This yields a scheme that is more accurate (for the considered resolutions and precision; not asymptotically) and, at the same time, requires substantially less communication. (Similar results are reported in [21] for a single-GPU implementation.) (*ii*) We choose texture-based Lagrange polynomial third order IP over spline IP (which had

Table VII: *Strong and weak scaling results for* CLAIRE *using synthetic data. The number of Gauss–Newton iterations is fixed to 5 and we use 10 PCG steps per Gauss-Newton iteration. We consider Inv$\mathcal{A}$ as a preconditioner. We report runtimes in seconds and total memory consumtion in GB per GPU. We use a fixed regularization of $\beta = 1\text{e}{-3}$ and set $N_t = 4$ with a linear interpolation (IP) model for the semi-Lagrangian (SL) method. All 1^{st} order derivatives are computed with FDs. The parallel layout (number of GPUs) for our experiments is restricted by the local slab size (can become too small) and the available GPU memory, respectively. The 2048^3 is the largest problem we could fit on TACC's Longhorn system. We cannot use less resources for this problem due to memory restrictions.*

nodes	#GPUs	FFT		SL		FD		overall		
		time	% comm.	time	% comm.	time	% comm	time	% comm.	memory
$N = 128^3$										
1	1	1.03e−1	0.0	1.82e−1	0.0	6.12e−2	0.0	5.11e−1	0.0	1.11
1	2	1.74e−1	44.5	3.88e−1	69.3	1.52e−1	54.3	8.37e−1	51.3	0.95
1	4	2.35e−1	59.8	4.13e−1	76.4	1.44e−1	62.0	9.17e−1	59.5	0.79
2	8	6.95e−1	85.5	5.56e−1	83.9	2.87e−1	84.4	1.66	78.4	0.71
4	16	5.38e−1	90.0	6.19e−1	85.5	5.72e−1	92.1	1.87	82.3	0.66
$N = 256^3$										
1	1	7.74e−1	0.0	1.16	0.0	3.72e−1	0.0	3.32	0.0	5.09
1	2	7.47e−1	42.3	1.20	61.0	4.64e−1	34.1	2.99	40.5	3.18
1	4	9.84e−1	74.7	8.20e−1	66.5	3.20e−1	45.4	2.56	55.6	1.95
2	8	1.69	89.2	1.23	85.2	3.90e−1	71.8	3.60	78.9	1.29
4	16	1.96	91.8	1.26	89.4	3.70e−1	79.6	3.81	84.5	0.94
8	32	1.36	95.3	1.24	91.4	3.59e−1	84.0	3.15	86.8	0.78
$N = 512^3$										
1	4	7.33	74.0	4.26	60.6	1.62	32.2	1.62e1	52.5	11.2
2	8	1.16e1	90.0	2.76	68.0	1.31	56.4	1.73e1	75.5	5.84
4	16	1.02e1	94.5	1.93	74.5	1.05	70.3	1.41e1	83.9	3.32
8	32	7.08	94.3	1.56	81.3	9.31e−1	80.4	1.01e1	85.9	2.00
16	64	4.88	96.8	1.58	87.9	8.75e−1	86.9	7.72	89.1	1.31
$N = 1024^3$										
8	32	4.06e1	95.0	5.33	73.4	2.85	69.6	5.19e1	85.7	11.5
16	64	2.44e1	95.0	4.17	81.9	2.48	81.4	3.27e1	87.4	6.23
32	128	1.47e1	96.9	3.94	89.2	2.20	88.2	2.18e1	90.2	3.43
64	256	1.00e1	97.5	6.64	96.2	2.04	92.3	1.95e1	92.9	2.12
$N = 2048^3$										
64	256	5.18e1	93.1	1.46e1	92.4	5.89	88.5	7.60e1	88.1	12.5

been shown to be superior on a single GPU [21]) to further reduce the communication between GPUs. (*iii*) We propose an efficient combination of cuFFT within nodes and a 2D slab decomposition approach across nodes, combined with an in-house developed, optimized all-to-all communication for regimes for which we could show that the available vendor MPI all-to-all [69] was sub-optimal. In addition to these kernel optimizations, we are able to substantially reduce the number of PCG iterations for computing the search direction within a Gauss–Newton–Krylov scheme and, thus, reduce the runtime by a factor of up to 2.5 compared to the prior version of CLAIRE. This is achieved through a new two-level (coarse grid) preconditioner based on a zero-velocity approximation of the Hessian operator, which eliminates expensive PDE solves. The entire solver is matrix-free. We optimized the memory footprint of the proposed solver. This allows us to solve larger problems on a single GPU, and to tackle problems of unprecedented scale. We ported CLAIRE to multi-GPU architectures as a whole, and support direct GPU-GPU communication through CUDA-aware MPI; no explicit host-to-device communication is required. The largest run reported in this study is $152\times$ larger than the results reported for the state-of-the-art [21]. Combining all improvements, we achieved a speedup of up to 70% compared to [21] on a single GPU. To showcase the capabilities of the proposed methodology, we reported results for the registration of real imaging data for resolutions of up to 1024^3 for MR neuroimaging data (on 8 nodes with a total of 32 GPUs) and $1024\times768\times768$ for CLARITY imaging data (on 4 nodes with a total of 16 GPUs). The achieved accuracy is equivalent to the results provided in prior work on CLAIRE [20]–[22], [24], [25], and on par or superior to other state-of-the-art software for diffeomorphic registration (see [20], [21] for a comparison).

Our work applies to other transport dominated forward and inverse problems. For example, the semi-Lagrangian GPU algorithm applies to particle-in-cell and weather/climate codes. The code basis of our solver (optimization scheme, linear algebra solvers, and preconditioning) are hardware agnostic. Our three main computational kernels should translate to other GPU accelerators as long as they provide some specialized hardware support. For example, the IP kernel relies on texture memory, which needs to be supported by the hardware. Also, certain parameters will need to be retuned. Most of the kernels are written in CUDA, so—although the algorithms won't change—the implementation will have to be ported to the new GPU programming interface.

Acknowledgments: We thank Nicolas Charon and Joshua T. Vogelstein at Johns Hopkins University for assisting us with gaining access to the CLARITY data.

REFERENCES

[1] B. Fischer and J. Modersitzki, "Ill-posed medicine – an introduction to image registration," *Inverse Problems*, vol. 24, no. 3, pp. 1–16, 2008.

[2] J. Modersitzki, *Numerical methods for image registration*. New York: Oxford University Press, 2004.

[3] A. Sotiras, C. Davatzikos, and N. Paragios, "Deformable medical image registration: A survey," *Medical Imaging, IEEE Transactions on*, vol. 32, no. 7, pp. 1153–1190, 2013.

[4] M. F. Beg, M. I. Miller, A. Trouvé, and L. Younes, "Computing large deformation metric mappings via geodesic flows of diffeomorphisms," *International Journal of Computer Vision*, vol. 61, no. 2, pp. 139–157, 2005.

[5] A. Trouvé, "Diffeomorphism groups and pattern matching in image analysis," *International Journal of Computer Vision*, vol. 28, no. 3, pp. 213–221, 1998.

[6] L. Younes, *Shapes and diffeomorphisms*. Springer, 2010.

[7] K. Chung, J. Wallace, S.-Y. Kim, S. Kalyanasundaram, A. S. Andalman, T. J. Davidson, J. J. Mirzabekov, K. A. Zalocusky, J. Mattis, A. K. Denisin, S. Pak, H. Bernstein, L. G. C. Ramakrishnan, V. Gradinaru, and K. Deisseroth, "Structural and molecular interrogation of intact biological systems," *Nature*, vol. 497, pp. 332–337, 2013.

[8] S.-Y. Kim, K. Chung, and K. Deisseroth, "Light microscopy mapping of connections in the intact brain," *Trends in Cognitive Sciences*, vol. 17, no. 12, pp. 596–599, 2013.

[9] K. S. Kutten, N. Charon, M. I. Miller, J. T. Ratnanather, K. Deisseroth, L. Ye, and J. T. Vogelstein, "A diffeomorphic approach to multimodal registration with mutual information: Applications to CLARITY mouse brain images," *ArXiv e-prints*, 2016.

[10] ——, "A diffeomorphic approach to multimodal registration with mutual information: Applications to CLARITY mouse brain images," in *Proc Medical Image Computing and Computer-Assisted Intervention*, vol. LNCS 10433, 2017, pp. 275–282.

[11] R. Tomer, L. Ye, B. Hsueh, and K. Deisseroth, "Advanced CLARITY for rapid and high-resolution imaging of intact tissues," *Nature protocols*, vol. 9, no. 7, pp. 1682–1697, 2014.

[12] J. T. Vogelstein, E. Perlman, B. Falk, A. Baden, W. G. Roncal, V. Chandrashekhar, F. Collman, S. Seshamani, J. L. Patsolic, K. Lillaney, M. Kazhdan, R. Hider, D. Pryor, J. Matelsky, T. Gion, P. Manavalan, B. Wester, M. Chevillet, E. T. Trautman, K. Khairy, E. Bridgeford, D. M. Kleissas, D. J. Tward, A. K. Crow, B. Hsueh, M. A. Wright, M. I. Miller, S. J. Smith, R. J. Vogelstein, K. Deisseroth, and R. Burns, "A community-developed open-source computational ecosystem for big neuro data," *Nature Methods*, vol. 11, no. 15, pp. 846–847, 2018.

[13] G. E. Christensen, X. Geng, J. G. Kuhl, J. Bruss, T. J. Grabowski, I. A. Pirwani, M. W. Vannier, J. S. Allen, and H. Damasio, "Introduction to the non-rigid image registration evaluation project," in *Proc Biomedical Image Registration*, vol. LNCS 4057, 2006, pp. 128–135.

[14] A. R. Jones, C. C. Overly, and S. M. Sunkin, "The Allen Brain Atlas: 5 years and beyond," *Nature Reviews Neuroscience*, vol. 10, pp. 821–828, 2009.

[15] V. Barbu and G. Marinoschi, "An optimal control approach to the optical flow problem," *Systems & Control Letters*, vol. 87, pp. 1–9, 2016.

[16] A. Borzì, K. Ito, and K. Kunisch, "Optimal control formulation for determining optical flow," *SIAM Journal on Scientific Computing*, vol. 24, no. 3, pp. 818–847, 2002.

[17] F.-X. Vialard, L. Risser, D. Rueckert, and C. J. Cotter, "Diffeomorphic 3D image registration via geodesic shooting using an efficient adjoint calculation," *International Journal of Computer Vision*, vol. 97, pp. 229–241, 2012.

[18] K. Chen and D. A. Lorenz, "Image sequence interpolation using optimal control," *Journal of Mathematical Imaging and Vision*, vol. 41, pp. 222–238, 2011.

[19] A. Mang and G. Biros, "Constrained H^1-regularization schemes for diffeomorphic image registration," *SIAM Journal on Imaging Sciences*, vol. 9, no. 3, pp. 1154–1194, 2016.

[20] A. Mang, A. Gholami, C. Davatzikos, and G. Biros, "CLAIRE: a distributed-memory solver for constrained large deformation diffeomorphic image registration," *SIAM Journal on Scientific Computing*, vol. 41, no. 5, pp. C548–C584, 2019.

[21] M. Brunn, N. Himthani, G. Biros, M. Mehl, and A. Mang, "Fast GPU 3D diffeomorphic image registration," *arXiv e-prints*, 2020.

[22] A. Gholami, A. Mang, K. Scheufele, C. Davatzikos, M. Mehl, and G. Biros, "A framework for scalable biophysics-based image analysis," in *Proc ACM/IEEE Conference on Supercomputing*, 2017, pp. 1–13.

[23] A. Mang and G. Biros, "An inexact Newton–Krylov algorithm for constrained diffeomorphic image registration," *SIAM Journal on Imaging Sciences*, vol. 8, no. 2, pp. 1030–1069, 2015.

[24] A. Mang, A. Gholami, and G. Biros, "Distributed-memory large-deformation diffeomorphic 3D image registration," in *Proc ACM/IEEE Conference on Supercomputing*, 2016.

[25] A. Mang and G. Biros, "Constrained large deformation diffeomorphic image registration (CLAIRE)," 2019, [Commit: v0.07-131-gbb7619e]. [Online]. Available: https://andreasmang.github.io/claire

[26] T. Vercauteren, X. Pennec, A. Perchant, and N. Ayache, "Diffeomorphic demons: Efficient non-parametric image registration," *NeuroImage*, vol. 45, no. 1, pp. S61–S72, 2009.

[27] B. B. Avants, N. J. Tustison, G. Song, P. A. Cook, A. Klein, and J. C. Gee, "A reproducible evaluation of ANTs similarity metric performance in brain image registration," *NeuroImage*, vol. 54, pp. 2033–2044, 2011.

[28] B. B. Avants, C. L. Epstein, M. Brossman, and J. C. Gee, "Symmetric diffeomorphic image registration with cross-correlation: Evaluating automated labeling of elderly and neurodegenerative brain," *Medical Image Analysis*, vol. 12, no. 1, pp. 26–41, 2008.

[29] B. B. Avants, N. J. Tustison, and H. J. Johnson, "ANTs." [Online]. Available: http://stnava.github.io/ANTs

[30] J. Ashburner, "A fast diffeomorphic image registration algorithm," *NeuroImage*, vol. 38, no. 1, pp. 95–113, 2007.

[31] A. Bone, O. Colliot, and S. Durrleman, "Learning distributions of shape trajectories from longitudinal datasets: A hierarchical model on a manifold of diffeomorphisms," *arXiv e-prints*, no. 1803.10119, 2019.

[32] A. Bone, M. Louis, B. Martin, and S. Durrleman, "Deformetrica 4: An open-source software for statistical shape analysis," in *Proc International Workshop on Shape in Medical Imaging*, vol. LNCS 11167, 2018, pp. 3–13.

[33] J. Fishbaugh, S. Durrleman, M. Prastawa, and G. Gerig, "Geodesic shape regression with multiple geometries and sparse parameters," *Medical Image Analysis*, vol. 39, pp. 1–17, 2017.

[34] S. Durrleman, A. Brone, M. Louis, B. Martin, P. Gori, A. Routier, M. Bacci, A. Fouquier, B. Charlier, J. Glaunes, J. Fishbaugh, M. Prastawa, M. Diaz, and C. Doucet, "deformetrica." [Online]. Available: http://www.deformetrica.org

[35] M. Zhang and P. T. Fletcher, "Fast diffeomorphic image registration via Fourier-approximated Lie algebras," *International Journal of Computer Vision*, pp. 1–13, 2018.

[36] Center for Imaging Science, Johns Hopkins University, "LDDMM Suite." [Online]. Available: http://cis.jhu.edu/software

[37] neurodata, "ARDENT." [Online]. Available: https://ardent.neurodata.io

[38] Insight Software Consortium, "ITKNDReg." [Online]. Available: https://github.com/InsightSoftwareConsortium/ITKNDReg

[39] J. S. Preston, "Python for computational anatomy." [Online]. Available: https://bitbucket.org/scicompanat/pyca

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

[41] R. Shams, P. Sadeghi, R. A. Kennedy, and R. I. Hartley, "A survey of medical image registration on multicore and the GPU," *Signal Processing Magazine, IEEE*, vol. 27, no. 2, pp. 50–60, 2010.

[42] A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte, "Medical image processing on the GPU–past, present and future," *Medical Image Analysis*, vol. 17, no. 8, pp. 1073–1094, 2013.

[43] D. Budelmann, L. Koenig, N. Papenberg, and J. Lellmann, "Fully-deformable 3D image registration in two seconds," in *Bildverarbeitung für die Medizin*, 2019, pp. 302–307.

[44] N. Courty and P. Hellier, "Accelerating 3D non-rigid registration using graphics hardware," *International Journal of Image and Graphics*, vol. 8, no. 1, pp. 81–98, 2008.

[45] S. Durrleman, M. Prastawa, N. Charon, J. R. Korenberg, S. Joshi, G. Gerig, and A. Trouve, "Morphometry of anatomical shape complexes with dense deformations and sparse parameters," *NeuroImage*, vol. 101, pp. 35–49, 2014.

[46] N. D. Ellingwood, Y. Yin, M. Smith, and C.-L. Lin, "Efficient methods for implementation of multi-level nonrigid mass-preserving image registration on GPUs and multi-threaded CPUs," *Computer Methods and Programs in Biomedicine*, vol. 127, pp. 290–300, 2016.

[47] X. Gu, H. Pan, Y. Liang, R. Castillo, D. Yang, D. Choi, E. Castillo, A. Majumdar, T. Guerrero, and S. B. Jiang, "Implementation and evaluation of various demons deformable image registration algorithms on a GPU," *Physics in Medicine and Biology*, vol. 55, no. 1, pp. 207–219, 2009.

[48] D. Grzech, L. Folgoc, M. P. Heinrich, B. Khanal, J. Moll, J. A. Schnabel, B. Glocker, and B. Kainz, "FastReg: Fast non-rigid registration via accelerated optimisation on the manifold of diffeomorphisms," *arXiv e-prints*, 2019. [Online]. Available: https://arxiv.org/abs/1903.01905

[49] L. K. Ha, J. Krüger, P. T. Fletcher, S. Joshi, and C. T. Silva, "Fast parallel unbiased diffeomorphic atlas construction on multi-graphics processing units," in *Proc Eurographics Conference on Parallel Grphics and Visualization*, 2009, pp. 41–48.

[50] L. Ha, J. Krüger, S. Joshi, and C. T. Silva, "Multiscale unbiased diffeomorphic atlas construction on multi-GPUs," in *CPU Computing Gems Emerald Edition*. Elsevier Inc, 2011, ch. 48, pp. 771–791.

[51] S. Joshi, B. Davis, M. Jornier, and G. Gerig, "Unbiased diffeomorphic atlas construction for computational anatomy," *NeuroImage*, vol. 23, no. 1, pp. S151–S160, 2005.

[52] L. Koenig, J. Ruehaak, A. Derksen, and J. Lellmann, "A matrix-free approach to parallel and memory-efficient deformable image registration," *SIAM Journal on Scientific Computing*, vol. 40, no. 3, pp. B858–B888, 2018.

[53] M. Modat, G. R. Ridgway, Z. A. Taylor, M. Lehmann, J. Barnes, D. J. Hawkes, N. C. Fox, and S. Ourselin, "Fast free-form deformation using graphics processing units," *Computer Methods and Programs in Biomedicine*, vol. 98, no. 3, pp. 278–284, 2010.

[54] S. Sommer, "Accelerating multi-scale flows for LDDKBM diffeomorphic registration," in *Proc IEEE International Conference on Computer Visions Workshops*, 2011, pp. 499–505.

[55] J. Shackleford, N. Kandasamy, and G. Sharp, "On developing B-spline registration algorithms for multi-core processors," *Physics in Medicine and Biology*, vol. 55, no. 21, pp. 6329–6351, 2010.

[56] D. P. Shamonin, E. E. Bron, B. P. F. Lelieveldt, M. Smits, S. Klein, and M. Staring, "Fast parallel image registration on CPU and GPU for diagnostic classification of Alzheimer's disease," *Frontiers in Neuroinformatics*, vol. 7, no. 50, pp. 1–15, 2014.

[57] P. Valero-Lara, "A GPU approach for accelerating 3D deformable registration (DARTEL) on brain biomedical images," in *Proc European MPI Users' Group Meeting*, 2013, pp. 187–192.

[58] ——, "Multi-GPU acceleration of DARTEL (early detection of Alzheimer)," in *Proc IEEE International Conference on Cluster Computing*, 2014, pp. 346–354.

[59] A. Mang and G. Biros, "A Semi-Lagrangian two-level preconditioned Newton–Krylov solver for constrained diffeomorphic image registration," *SIAM Journal on Scientific Computing*, vol. 39, no. 6, pp. B1064–B1101, 2017.

[60] A. Alexanderian, N. Petra, G. Stadler, and O. Ghattas, "A fast and scalable method for A-optimal design of experiments for infinite-dimensional Bayesian nonlinear inverse problems," *SIAM Journal on Scientific Computing*, vol. 38, no. 1, pp. A243–A272, 2016.

[61] T. Bui-Thanh, O. Ghattas, J. Martin, and G. Stadler, "A computational framework for infinite-dimensional Bayesian inverse problems Part I: The linearized case, with application to global seismic inversion," *SIAM Journal on Scientific Computing*, vol. 35, no. 6, pp. A2494–A2523, 2013.

[62] A. Mang, A. Gholami, C. Davatzikos, and G. Biros, "PDE-constrained optimization in medical image analysis," *Optimization and Engineering*, vol. 19, no. 3, pp. 765–812, 2018, https://doi.org/10.1007/s11081-018-9390-9.

[63] Nvidia, "CUDA CUFFT Library," 2007. [Online]. Available: https://docs.nvidia.com/cuda/cufft/index.html

[64] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. Gropp, D. Karpeyev, D. Kaushik, M. Knepley, D. May, L. C. McInnes, R. Mills, T. Munson, K. Rupp, P. Sanan, B. Smith, S. Zampini, H. Zhang, and H. Zhang, "PETSc users manual," Argonne National Laboratory, Tech. Rep. ANL-95/11 - Revision 3.13, 2020.

[65] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, "PETSc and TAO webpage (PETSc version 3.12.4)." [Online]. Available: https://www.mcs.anl.gov/petsc

[66] J. Hoberock and N. Bell, "Thrust, the cuda c++ template library," 2010. [Online]. Available: https://docs.nvidia.com/cuda/thrust/index.html

[67] A. Gholami and G. Biros, "AccFFT," 2017. [Online]. Available: https://github.com/amirgholami/accfft

[68] ——, "AccFFT home page," 2017. [Online]. Available: http://www.accfft.org

[69] "IBM Spectrum MPI (version 10.3.0)." [Online]. Available: https://www.ibm.com/us-en/marketplace/spectrum-mpi

[70] V. Chandrashekhar, A. Crow, J. Bogelstein, and K. Deisseroth, "NEURODATA CLARITOMES." [Online]. Available: https://neurodata.io/project/claritomes

[71] J. Modersitzki, *FAIR: Flexible algorithms for image registration*. Philadelphia, Pennsylvania, US: SIAM, 2009.

[72] NVIDIA, "CUDA Toolkit (version 10.1)." [Online]. Available: https://developer.nvidia.com/cuda-downloads

[73] K. Fissell and R. Reynolds, "niftilib (version 2.2.0)," 2020. [Online]. Available: http://niftilib.sourceforge.net

[74] IBM, "IBM XL C/C++ (version 16.1.1)." [Online]. Available: https://www.ibm.com/us-en/marketplace/xl-cpp-linux-compiler-power

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

Hardware: We execute all runs of CLAIRE and the implemented submodules for the computational kernels on TACC's Longhorn system. Longhorn hosts 96 NVIDIA Tesla V100 nodes. Each node is equipped with four GPUs with 4x16 GB GPU RAM (64GB aggregate). Each node has 2 IBM Power 9 processors (model: IBM Power System AC922 (8335-GTH)) with 20 cores (40 cores per node) at 2.3 GHz with 256 GB memory. There are also 8 NVIDIA Tesla V100 large memory nodes (same specs) with 512GB of memory (not used in our experiments). Details can be found here: https://portal.tacc.utexas.edu/user-guides/longhorn. (Architecture: ppc64le; Byte Order: Little Endian; Thread(s) per Core: 4; Core(s) per Socket: 20; Socket(s): 2; NUMA node(s): 6; Model: 2.2 (pvr 004e 1202); Model name: POWER9, altivec supported; CPU max MHz: 3800; CPU min MHz: 2300; L1d cache: 32K; L1i cache: 32K; L2 cache: 512K; L3 cache: 10240K)

Developed Code: Our code is written in C++ using the c++11 standard. CLAIRE is compiled using the IBM XL compiler (version 16.1.1) with CUDA 10.1. The parameters used to execute CLAIRE and its submodules are described in detail in the respective sections in the manuscript. As mentioned above, all runs are executed on TACC's Longhorn system. A detailed documentation on how to execute CLAIRE to be able to reproduce the runs is provided on a webpage maintained by the original authors of CLAIRE (see https://github.com/andreasmang/claire and https://andreasmang.github.io/claire). The source code of CLAIRE for prior versions is publicly available. CLAIRE includes the following binaries (which we use for the production runs in the present manuscript):

- **bin/claire** to run the registration on synthetic and real data,
- **bin/clairetools** to post-process registration results, and
- **bin/test** to run unit tests for single components of the registration.

General instructions on how to compile and run the registration code are made available online. Upon acceptance of the article, the proposed GPU code will released to the public (which includes scripts for the GPU version to execute CLAIRE on some of the NIREP test data sets).

Third Party Software Packages and Libaries: We use the PETSc library (version 3.12.4; https://www.mcs.anl.gov/petsc) for linear algebra operations, PETSc's TAO package for the nonlinear optimization, cuFFT for Fast Fourier Transforms, niftilib (version 2.2.0; http://downloads.sourceforge.net/project/niftilib) for I/O, and IBM Spectrum MPI version 10.3.0; The modules that are loaded on Longhorn for compilation are

- `xl/16.1.1`
- `autotools/1.2`
- `TACC`
- `spectrum_mpi/10.3.0`
- `cmake/3.16.1`
- `cuda/10.1`
- `git/2.24.1`
- `xalt/2.8.1`

We used the following flags to compile PETSc in single precision for the GPU:

```
MPI_DIR=/usr/include/openmpi/
./configure
   --download-f2cblaslapack
   --with-mpi-dir=$MPI_DIR
   --known-mpi-shared-libraries=0
   --download-fblaslapack=$LP_DIR
   --with-64-bit-indices
   --with-precision=single
   --with-shared=0
   --with-x=0 --with-fc=0
   --with-ssl=0
   --with-cuda=1
   --download-cusp=yes
   --CUDAFLAGS='-arch=sm_70'
   --CUDAOPTFLAGS='-O3'
   -use-gpu-aware-mpi
```

We compile all of the codes with -O3 -qarch=pwr9 -qtune=pwr9 flags. The external libraries are linked statically.

Data Sets: We consider an open-access data repository that has been widely used in the medical image computing community to study the performance of diffeomorphic image registration algorithms: The data is taken from the Non-rigid Registration Evaluation Project (NIREP). We consider the datasets na01, na02, na03, and na10 for our experiments. The original data is stored in *.nii.gz format. The original resolution of the data is 256x300x256. The CLARITY images used in this study are also publicly available. We resampled the data in the spectral domains. We also execute runs for synthetic datasets. How these data are generated is described in detail in the manuscript. The NIREP data has been provided online by the original authors of CLAIRE: https://github.com/andreasmang/nirep.

ARTIFACT AVAILABILITY

Software Artifact Availability: All author-created software artifacts are maintained in a public repository under an OSI-approved license.

Hardware Artifact Availability: There are no author-created hardware artifacts.

Data Artifact Availability: There are no author-created data artifacts.

Proprietary Artifacts: None of the associated artifacts, author-created or otherwise, are proprietary.

Author-Created or Modified Artifacts:

```
Persistent ID: https://github.com/andreasmang/claire
Artifact name: CLAIRE
```

```
Citation of artifact: A. Mang, A. Gholami, C.
↪ Davatzikos \& G. Biros. CLAIRE: A
↪ distributed-memory solver for constrained large
↪ deformation diffeomorphic image registration.
↪ SIAM Journal on Scientific Computing
↪ 41(5):C548--C584, 2019
```

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: TACC's Longhorn system (Longhorn hosts 96 NVIDIA Tesla V100 nodes. Each node is equipped with four GPUs with 4x16 GB GPU RAM (64GB aggregate). Each node has 2 IBM Power 9 processors (model: IBM Power System AC922 (8335-GTH)) with 20 cores (40 cores per node) at 2.3 GHz with 256 GB memory.) There are also 8 NVIDIA Tesla V100 large memory nodes (same specs) with 512GB or memory. Details can be found here: https://portal.tacc.utexas.edu/user-guides/longhorn. (Architecture: ppc64le; Byte Order: Little Endian; Thread(s) per Core: 4; Core(s) per Socket: 20; Socket(s): 2; NUMA node(s): 6; Model: 2.2 (pvr 004e 1202); Model name: POWER9, altivec supported; CPU max MHz: 3800; CPU min MHz: 2300; L1d cache: 32K; L1i cache: 32K; L2 cache: 512K; L3 cache: 10240K).

Operating systems and versions: Linux c003-011.longhorn.tacc.utexas.edu 4.14.0-115.10.1.el7a.ppc64le SMP Wed Jun 26 09:32:17 UTC 2019 ppc64le ppc64le ppc64le GNU/Linux

Compilers and versions: IBM XL version 16.1.1

Applications and versions: FAIR (git@github.com:C4IR/FAIR.m.git; commit: df184cdda6dd15cc947b726e2d0cb42aa10fd8b5)

Libraries and versions: Libraries and Versions: PETSc version 3.12.4 (https://www.mcs.anl.gov/petsc), IBM Spectrum MPI version 10.3.0; NIfTI version 2.0.0 (http://downloads.sourceforge.net/project/niftilib), CUDA version 10.1 (https://developer.nvidia.com/cuda-downloads)

Key algorithms: Preconditioned Conjugate Gradient, Newton–Krylov–Method, Fast Fourier Transform, Lagrange Interpolation

Input datasets and versions: NIREP imaging data (datasets na01, na03, na10); CLARITY imaging data (Control 189 and Cocaine 175); Both publicly available online as disclosed in the manuscript.

URL to output from scripts that gathers execution environment information.

`https://github.com/SC-Tech-Program/Author-Kit`

ARTIFACT EVALUATION

Verification and validation studies: We have implemented several unit tests to confirm that the implemented computational kernels follow the theoretically expected behavior. This includes tests for (i) the transport equations (flowing data backward and forward in time and measure the error), (ii) studies of grid convergence to assess asymptotic behaviour of the transport equations, computational kernels, and the Newton solver (how errors behave with reducing the mesh size of our problem), (iii) synthetic test cases to study error behavior in the computational kernels (FD gradient and divergence operators, spectral laplacian operator) (iv) tests to check for errors in the gradient and Hessian operators (essentially, an asymptotic comparison against finite differences) (v) synthetic test cases for the inverse problem (i.e., setup a test problem with a known solution) (vi) symmetry checks to assess the symmetry of the Hessian operator as well as the error in the forward and adjoint operators. We have measured memory allocation and compared it to our analytical estimates. Most of these tests are not reported in the manuscript. They are for debugging purposes and for ensuring that the computational results reported in the present study are trustworthy.

Accuracy and precision of timings: For the evaluation of the run and communication times for the kernels on a standalone basis, we execute them multiple times and report average numbers to avoid outliers. We report several runs on different datasets none of which revealed any discrepancies in the reported timings. All runs reported in this study were performed on TACC's Longhorn system. Considering the accuracy of the timings in the kernels: We compared the performance of the standalone kernels for different GPU and node layouts to the evaluation within the solver. When executing the solver, these kernels are called hundreds to thousands of times. We did not observe any outliers.

Used manufactured solutions or spectral properties: We have used synthetic problems to test the numerical accuracy of the implemented kernels. For testing the kernels used for differentiation we compare the numerical results to analytic functions of varying complexity. For the interpolation kernel, we explored the error of evaluating a synthetic test function at off-grid-locations to the numerical value obtained by the interpolation kernel. For the registration problem, we set up synthetic test problems by generating artificial reference images by applying the velocity field to a given image. We then try to recover this very velocity field by executing the solver on these images.

Quantified the sensitivity of results to initial conditions and/or parameters of the computational environment: We report and have created additional results not reported in the manuscript to test the sensitivity of the produced results to parameters used in our solver. These parameters include Newton iteration counts, relative tolerance for the Newton solver, relative tolerance for the PCG methods, regularization parameters, grid sizes, number of time points. We have performed numerous runs with different settings of our software and did not observe any sensitivities to parameters of the computational environment.

Controls, statistics, or other steps taken to make the measurements and analyses robust to variability and unknowns in the system. We test our methodology on synthetic and real data. For the synthetic cases, we use velocities generated from real data. We use different datasets as input to our problem. For the brain imaging data, we consider the dataset na01 as a reference image and register three datasets from the publicly available data repository (na02, na03, and na10) to it. These datasets represent registration problems of varying complexity with na02 to na01 being the "easiest" case (small

deformations) and na10 to na01 a more "difficult" case (large deformations). For the CLARITY imaging data, we use two datasets from the data repository and register them (Cocaine 175 and Control 189). All computational components in the pipeline are deterministic and can be reproduced.