The Dissertation Committee for Amir Gholaminejad certifies that this is the approved version of the following dissertation:

Fast Algorithms for Biophysically-Constrained Inverse Problems in Medical imaging

Committee:

George Biros, Supervisor

J. Tinsley Oden

Omar Ghattas

Robert van de Geijn

Richard Vuduc

Kui Ren

Fast Algorithms for Biophysically-Constrained Inverse Problems in Medical imaging

by

Amir Gholaminejad, B.Sc., M.S.E.

Dissertation

Presented to the Faculty of the Graduate School
of the University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

The University of Texas at Austin
August 2017

To My Parents

Life is the single stage of our art

Everybody sings his own song and goes from the stage

The stage is always on

Good for the song that people remember

- Jale Esfahani

Acknowledgments

This work would not have been possible without the blessing from the two angels of my life, who taught me the value of education. My parents supported me to follow my dreams, even in hard times. I want to also thank my great uncles and my cousins Dr. Behnood Gholami and Dr. Farnood Gholami, who inspired me and gave me valuable feedback.

I was very lucky to have great mentors in my life. I like to start with my 7th grade math teacher, Mr. Sadri. He diligently showed us the beauty of mathematical reasoning and the excitement that follows when you really understand a problem. He made his students believe they can achieve almost anything by hard work and determination. I am in debt to my undergraduate mentor, Prof. Reza Hosseini in Tehran Polytechnic (Amirkabir University). I still remember the first day that I gave him my proposal to study the Mpemba effect. Instead of turning down a first semester undergraduate student, he agreed to advise my project and gave me full access to his lab, where only senior students and graduate students were allowed. That was a great period for me as I met very smart people and lead many different projects. I am really grateful for his trust in my proposal and his support and mentorship. I would like to thank my PhD adviser, Prof. George Biros, who spent countless hours advising me to understand the core topics. To some extent George and Mr. Sadri have a very similar approach to science: Understand instead of memorizing. He is a great mentor who really cares about his group, and treats them like family. George is smart, energetic, and a great adviser. This work would have been impossible without his vision and mentorship. My relationship with him was most of the time not an advisor/advisee, but a friend. I

am grateful for his extensive support and mentorship.

I would also like to thank Dr. Alex Fit-Florea, Dr. Boris Ginsburg, Dr. Sergei Nikolaev, Dr. Pooya Davoodi, Bragadeesh Natarajan, Kent Knox, Timmy Liu, Vivek Viswanathan, and Greg Stoner who were my mentors and colleagues at NVIDIA and AMD, where I did my internships. I learned valuable skills during that time.

I would also like to thank my labmates Bo Xiao, Shashank Subramanian, Naveen Himthani, Sameer Tharakan, Gokberk Kabacaoglu, Kyle Xu, Chenhan Yu, James Levitt, Romir Moza, Hari Sundar, Carlos Borges, Bryan Quaife, and Bill March, as well as Sue Rodriguez and Donna Taylor. I also want to thank Dhairya Malhotra, Andreas Mang, Hari Sundar, Klaudius Scheufele, Florian Tramnitzke, and Prof. Miriam Mehl whom I collaborated with on this project.

I would like to thank my committee members, Prof. Tinsley Oden, Prof. Omar Ghattas, Prof. Robert van de Geijn, Prof. Richard Vuduc, and Prof. Kui Ren. It is a great honor to have their endorsement on my PhD, and I am grateful for the time they spent to review this thesis.

Finally, I would like to thank Prof. Ivo Babuska, an iconic mathematician whom I had the great honor to work with as the Babuska forum host. I am grateful for reading the thesis and providing me with his valuable feedback.

Amir Gholaminejad,

August 2017

Fast Algorithms for Biophysically-Constrained Inverse Problems in Medical imaging

by

Amir Gholaminejad, Ph.D.
The University of Texas at Austin, 2017
Supervisor: George Biros

We present algorithms and software for parameter estimation for forward and inverse tumor growth problems and diffeomorphic image registration. Our methods target the following scenarios: automatic image registration of healthy images to tumor bearing medical images and parameter estimation/calibration of tumor models. This thesis focuses on robust and scalable algorithms for these problems.

Although the proposed framework applies to many problems in oncology, we focus on primary brain tumors and in particular low and high-grade gliomas. For the tumor model, the main quantity of interest is the extent of tumor infiltration into the brain, beyond what is visible in imaging.

The inverse tumor problem assumes that we have patient images at two (or more) well-separated times so that we can observe the tumor growth. Also, the inverse problem requires that the two images are segmented. But in a clinical setting such information is usually not available. In a typical case, we just have multimodal magnetic resonance images with no segmentation. We address this lack of infor-

mation by solving a coupled inverse registration and tumor problem. The role of image registration is to find a plausible mapping between the patient's tumor-bearing image and a normal brain (atlas), with known segmentation. Solving this coupled inverse problem has a prohibitive computational cost, especially in 3D. To address this challenge we have developed novel schemes, scaled up to 200K cores.

Our main contributions is the design and implementation of fast solvers for these problems. We also study the performance for the tumor parameter estimation and registration solvers and their algorithmic scalability. In particular, we introduce the following novel algorithms: An adjoint formulation for tumor-growth problems with/without mass-effect; The first parallel 3D Newton-Krylov method for large diffeomorphic image registration; A novel parallel semi-Lagrangian algorithm for solving advection equations in image registration and its parallel implementation on shared and distributed memory architectures; and Accelerated FFT (AccFFT), an open-source parallel FFT library for CPU and GPUs scaled up to 131,000 cores with optimized kernels for computing spectral operators.

The scientific outcomes of this thesis, has appeared in the proceedings of three ACM/IEEE SCxy conferences (two best student paper finalist, and one ACM SRC gold medal), two journal papers, two papers in review, four papers in preparation (coupling, mass effect, segmentation, and multi-species tumor model), and seven conference presentations.

Table of Contents

Chapter 1

Introduction and Summary of Thesis

1.1 Motivation

Advances in medical imaging and computer vision have had a major impact on diagnosis and treatment of malignant brain tumors. However, there are still major difficulties in extracting information from these images. The main reason is that most imaging modalities (such as Magnetic Resonance Imaging or Computed Tomography) are only sensitive to areas with considerable abnormalities. Therefore, areas with small tumor concentration cannot be detected using existing imaging technologies. A biopsy is required, which is dangerous, invasive, and often not sufficient (since its results depend on where the tissue samples are taken from). This lack of sensitivity to diffuse abnormalities, along with low-resolution, imaging artifacts, and noise, create major challenges in image-based diagnosis and treatment planing. Computational methods that assist in image analysis are of paramount importance in helping improve the quality of interpretation of medical images, in a systematic and reproducible way. But computational methods also come with several challenges: designing algorithms with optimal work complexity, scalability, as well as fundamental mathematical and modeling questions. In this thesis, we focus on algorithmic and scalability challenges for a set of well-established methods in neuroimaging and neurooncology. In particular, we present algorithms and software for parameter estimation for tumor growth problems with/without mass effect, for fast diffeomorphic image registration, and for the coupled inverse tumor and inverse registration problem. Our methods target three different image

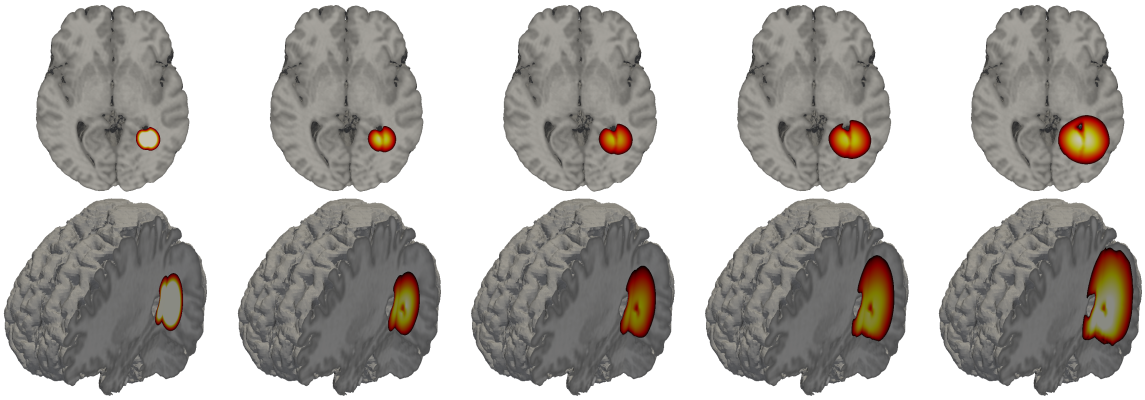


Figure 1.1: Brain tumor simulation results. We display (from left to right) the time progression of a reaction-diffusion model of tumor growth. The leftmost image shows the initial condition; the rightmost image the solution of the forward problem at $t = 1$. The computed tumor cell distribution $c \in [0, 1]$ is overlaid onto brain anatomy used for this simulation. The bottom row shows a 3D illustration of this anatomy and the top row an axial slice cut through the center of the tumor. White areas are locations of high tumor cell density and black areas, locations of low density.

analysis scenarios:

- Parameter estimation and calibration of patient-specific tumor models, given partially observed, noisy data from medical images.
- Fast image registration that can be applied to a wide range of medical image analysis problems.
- Coupled image registration and tumor model calibration, for registration of tumor-bearing images to normal brains (atlases) and for tumor growth model calibration from a single snapshot.

In the clinical setting, these scenarios are used in diagnosis and tumor staging, pre-operative surgical planning, post-operative analysis and prognosis, and epidemiological studies. In this work, we particularly focus on gliomas, a type of malignant primary brain tumor that arise from glial cells in the brain. They account for 29% of all brain and central nervous system tumors, and 80% of all

malignant brain tumors of about 60,000 cases diagnosed each year in the United States (36). Despite advances in surgery, chemotherapy, and radiotherapy, the median survival rate with therapy has remained about one year in the past 30 years (126, 148, 153, 167, 185). Below we explain, the motivation behind each of these objectives.

Every patient's brain structure and the specific genetics of his/her tumor is different from another patient. We cannot use a one-size-fits-all approach and use the same model parameters for all patients. Given a model for tumor growth, we have to perform a calibration to find its patient-specific parameters. The calibration is done by finding the parameters that "best" explain the observed data, without over-fitting. A set of parameters with which we can exactly match all the observed data in the patient, does not necessarily mean that we have found the ground truth. Increasing the number of unknowns in the model can exacerbate this situation, since it gives us more degrees of freedom to match noisy data, which can lead to poor generalization error.

Once the calibration process is done, we can perform the following analyses:

- *Estimate the extent of tumor infiltration* beyond what is visible in imaging. This is important for treatment, where the surgeon/radiologist has to determine how much to cut/radiate beyond the visible abnormality (82, 153). In clinical practice, a margin of 1cm–4cm is typically used (77, 94). This margin is derived by statistical analysis of clinical data (77). But there is no consensus about its particular value and there is no systematic way to select this margin for an individual patient.

- *Image registration*. In surgical planning a key question is path design, how to reach and extract the tumor without damaging eloquent areas of the brain.

Identifying these areas automatically can be done by registering the patient images to an atlas that has functional information. The challenge is that the atlas is a normal brain whereas the subject has a tumor. Standard image registration algorithms fail for such cases. Coupling tumor growth models with registration is a way to address this problem. In this scenario, the tumor growth model serves solely as an image analysis tool.

- *Single-time-snapshot model calibration*. Another instance where we need image registration is when we create patient-specific tumor models using just a single time data. Tumor growth is an evolution problem, so the typical inversion scenario requires at least two time snapshots. In clinical practice however, such snapshots are typically not available for preoperative high-grade gliomas. In this scenario, an atlas can serve as the $t = 0$ data and it requires image registration.

- *Segmentation of images bearing tumors.* Once registration between atlas and subject is possible, the algorithm can be used for segmentation [1]. A very powerful class of segmentation methods is atlas-based segmentation and a key component of this method is image registration (60).

- *Glioma Classification.* Correctly characterizing low and high-grade gliomas with a biopsy remains an open problem. A tumor growth model has the potential to address this.

- *Survival estimation* Given multi-modal images of gliomas, it is important to get an estimate of the expected survival time of the patient. For example,

[1] Image segmentation is the process in which we assign labels to each voxel from different categories (e.g. white matter, gray matter, cerebrospinal fluid, enhancing and non-enhancing tumor, necrosis, and edema)

tumor infiltration has been found to correlate with survival (105).

To perform the tumor model calibration we need to solve an inverse problem given a model for the tumor growth (§1.2). One of the inputs to the inverse problem is the segmentation of the patient's brain – both with tumor and without tumor. The latter gives us the healthy structure of the brain, which can be used as an initial condition to perform the simulations. However, this data is usually unavailable. We address this limitation by coupling the inverse tumor solver with image registration, and solve a coupled inverse problem. We first summarize the inverse registration problem in §1.3, and then motivate the coupling framework in §1.4.

1.2 Inverse Tumor Problem:

Initially, we consider a reaction-diffusion model for tumor growth, which has been widely adopted in the literature (85, 166, 167, 168). An exemplary forward simulation using this model is shown in Fig. 1.1. This is a phenomenological model that captures two distinctive behaviors of gliomas: their growth, and their invasion into surrounding tissues. We then revise this model, and couple it to linear elasticity equations to capture "mass effect", a term-of-art used by clinicians to describe tissue displacement due to tumor growth forces.

We formulate the inverse problem as a PDE-constrained optimization problem. We invert for the initial condition of the tumor concentration and the extent of anisotropic infiltration, given partial and noisy observations of the patient's tumor concentration. We present the formulation and outline the numerical algorithms for solving the resulting equations. The optimization problem is solved with a reduced-Newton method. We test the overall methodology using synthetic

datasets and compute the reconstruction error for different noise levels and detection thresholds for monofocal and multifocal test cases.

The summary of the contributions made for this part is as follows:

- Derivation and implementation of a second-order preconditioned, reduced-Newton framework for the inverse tumor solver with preconditioners for the forward and adjoint operators;
- Support for multiple anisotropic diffusion models, and multifocal tumors;
- Distributed-memory implementation that has been scaled to 131K cores on multiple systems;
- Algorithmic and parallel scalability on real clinical data; and
- Introduction of AccFFT, a new open-source parallel FFT library for CPU and GPUs with fast spectral operators that considerably reduce communication volume;

Results of this work has appeared in the following publications:

- A. Gholami, A. Mang, and G. Biros. *An inverse problem formulation for parameter estimation of a reaction-diffusion model of low grade gliomas.* Journal of mathematical biology, Vol. 72, pp 409-433, 2015.
- A. Gholami, D. Malhotra, H. Sundar and G. Biros. *FFT, FMM, or Multigrid? A comparative Study of State-Of-the-Art Poisson Solvers for Uniform and Nonuniform Grids in the Unit Cube*. SIAM Journal on Scientific Computing, Vol. 38 (3), 2016.

- A. Gholami, A. Mang, K. Scheufele, M. Mehl, and George Biros, *A Framework for Scalable Biophysics- based Image Analysis*. Proceedings of ACM/IEEE Supercomputing Conference (SC'17), 2017.

- A. Gholami, J. Hill, D. Malhotra, and G. Biros. *AccFFT: A library for distributed-memory FFT on CPU and GPU architectures.*
  arXiv preprint: http://arxiv.org/abs/1506.07933.

Related Work.

The complexity of the underlying bio-physiology results in a diversity of mathematical models, accounting for phenomena on the molecular, cellular and/or tissue scale (13). Most work attempting to link mathematical models to images is based on reaction-diffusion type equations (124). The main assumption here is that cancerous cells grow and infiltrate tissue due to cell division (proliferation) and migration (which can be modeled by diffusion). Although this model is simplistic and purely phenomenological, it has been shown to capture tumor progression on a tissue level (28, 89, 90, 113, 144).

The reaction-diffusion model has been used extensively for recovering tumor growth patterns in individual patients (28, 79, 81, 85, 90, 113, 144, 166, 168), for estimating the physiological tumor boundary (29, 89, 123), and for studying effects of clinical intervention (142, 145, 146, 167, 178). The parameter calibration is typically driven using segmented data of patient images. Besides only considering tissue composition (79, 81, 166, 168), some researchers have also considered fiber structures of the brain to account for the hypothesis of faster migration along those structures (16, 28, 29, 40, 89, 90, 113, 123, 133, 144). Models that account for the mechanical interaction of the tumor with its surroundings have been described

in (28, 79, 121). More sophisticated models are discussed in (71, 72, 84, 101, 131). One problem with using a more complex model for the inverse solve, is the potential for over-fitting, as mentioned above.

A common approach for parameter estimation is manual calibration (28, 168). However, it is somewhat difficult to reproduce results and this approach does not scale with the number of parameters. For this reason, several groups have developed automatic parameter calibration algorithms. One approach is to use an asymptotic approximation of the reaction-diffusion type equations on the basis of a traveling wave solution (167). The associated analytical result, establishes a connection between the velocity of the (spherical) tumor front and the parameters of the underlying reaction-diffusion model. An extension that accounts for the heterogeneity of the tissue as well as the structure of white matter pathways has been described in (89, 90). This method has been applied to time series of images (90) and to imaging studies based on a single snapshot in time (144).

Another approach for parameter estimation, based on optimal control theory, is to use PDE-constrained optimization (81, 113). Our approach extends this in that we use second order (Hessian) information for the numerical optimization, instead of using first order information only (81) or falling back to a derivative free optimization (81, 113).

1.3 Image Registration

Image registration is a correspondence problem between a template, $\boldsymbol{m}_T$, and a reference image, $\boldsymbol{m}_R$. The goal is to find a "plausible mapping", $\boldsymbol{y}$, between these two images, such that the mismatch between $\boldsymbol{m}_T(\boldsymbol{y})$ and $\boldsymbol{m}_R$ (i.e. $\|\boldsymbol{m}_T(\boldsymbol{y}) - \boldsymbol{m}_R\|_{L^2}$) is minimized (Fig. 1.2). It is important to add an admissibil-

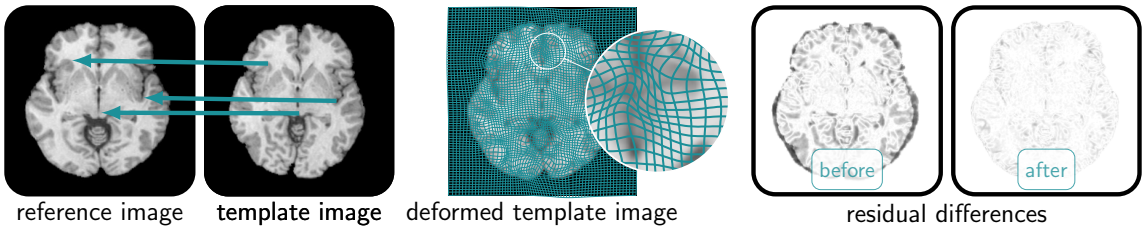


Figure 1.2: Image registration. The inputs to this inverse problem are scalar intensity values of two images of the same object (left). The inherent assumption is that there exists a geometric transformation that relates points in the reference image m_R to its corresponding points in the template image m_T (green arrows).

ity constraint on the mapping, or else the solver will find unphysical shuffling of points to minimize the mismatch. The plausibility constraint requires diffeomorphic mapping. That is, we seek a continuously differentiable, bijective mapping with continuously differentiable inverse. We will first discuss the inverse registration, and then its coupling to the inverse tumor problem.

We write the inverse registration problem as finding a velocity $\boldsymbol{v}$ that can "advect" the template image to become "similar" to the reference image. This requires solving a non-convex optimization problem, with a prohibitive time-to-solution especially in 3D. We have addressed this by developing a distributed-memory large deformation diffeomorphic registration framework, which has been scaled up to 8K cores. The framework uses a novel parallel semi-Lagrangian solver with support for Haswell and KNL architectures and GPUs. Moreover, we have extended the AccFFT library to support the spectral operators used by the registration, and made operator-specific optimizations that significantly reduce the communication volume.

The main bottlenecks of the inverse problem are pseudo-spectral operators (used for evaluating differential operators) and semi-Lagrangian solvers. We use a preconditioned, inexact (Gauss) Newton method, combined with novel distributed-

memory solvers that have been scaled to more than 131K cores. These include development of Accelerated FFT (AccFFT), an open-source parallel FFT library for CPU and GPUs with optimized kernels for computing spectral operators, as well as a novel parallel semi-Lagrangian algorithm for solving advection equations that appear in image registration. The contributions of our framework is as follows:

Interpolation operator: The main bottleneck for the inverse registration problem is solving variants of the advection equation with a stationary but variable velocity. To avoid the CFL condition we use a semi-Lagrangian solver. However, that requires interpolation from a regular grid to an unstructured grid which is very expensive. In our initial work (111), developed as part of this thesis, we presented a novel distributed implementation to solve the advection problem. Even though we achieved very good scalability, but the interpolation kernel was still a significant bottleneck, consuming 60% to 75% of the total runtime. Here, we present a new interpolation kernel that it is nearly 10× faster. It employs a reordering of departure points, blocking, and vectorization.

Switching to single precision: Given the level of noise and imaging artifacts in the data, the modeling errors in the biophysical simulations, and the target levels of accuracy, using double precision is not optimal. Using single precision reduces the communication volume and the memory footprint for the adjoint solves (which require storing the time history). For this reason we have implemented our solver in single-precision, and extended AccFFT to single-precision for both forward and backward FFTs as well as the fast spectral operators (56, 57) (for both CPU and GPU). Overall, our registration solver is 3 to 8 times faster than (111), supports stand-alone reaction-advection-diffusion solvers for biophysical inverse problems, and their coupling with registration. In chapter 3, we showcase results where we

solve registration problems of unprecedented scale, 4096^3 resulting in ≈ 200 billion unknowns, a problem size that is $64\times$ larger than the state-of-the-art. For problem sizes of clinical interest, our framework is about $8\times$ faster than the state-of-the-art. To our knowledge, the most scalable deformable image registration algorithm is the one developed in our earlier work (111). For a detailed review on image registration algorithms see (159). There are many scalable solvers for biophysical simulation but not much work for problems that are tightly coupled with MRI. In the latter area, most work is done on single node systems (28, 154).
Results of this work has appeared in the following publications:

- A. Gholami, A. Mang, K. Scheufele, M. Mehl, and George Biros, *A Framework for Scalable Biophysics- based Image Analysis*. Proceedings of ACM/IEEE Supercomputing Conference (SC'17), 2017.

- A. Mang, A. Gholami, and G. Biros, *Distributed-memory large-deformation diffeomorphic 3D image registration*. Proceedings of ACM/IEEE Supercomputing Conference (SC16), 2016.

- A. Mang, A. Gholami, C. Davatzikos, G. Biros, *PDE Constrained Optimization in Medical Image Analysis*, Optimization and Engineering, (in review).

1.4 Coupled image registration and inverse tumor problem

Solving the inverse tumor problem requires access to the healthy state of the patient's brain prior to the tumor. Often times, such data is not available. The tumor growth causes visible deformations and changes to the brain structure. As a result, the tumor bearing image cannot be used as an initial brain geometry for the simulations. We address this limitation by solving a coupled inverse registration

and tumor problem. The role of the image registration in the coupled framework, is to find a plausible mapping between the patient's tumor bearing image and an atlas, whose structure and properties is known.

In the coupled framework, first the patient's MRI is co-registered to an already segmented normal-brain MRI (i.e. the atlas), and then the labels from the normal image are transferred to the patient (through the registration map). Since the atlas doesn't have a tumor (whereas the patient does), a fictitious tumor needs to be grown in the atlas before the registration can be performed. Once this mapping is found, we can solve the parameter estimation problem in the atlas domain and translate the results through the inverse mapping into the patient's domain.

In the coupling, we have to solve two subproblems: identifying initial conditions for a reaction-diffusion tumor growth model (the inverse tumor parameter identification problem discussed above), and seeking a mapping between the atlas image and the patient's, so that their L^2-distance is small (image registration).

We use a Picard's scheme to solve the coupling problem. We first start with an initial guess for the tumor parameters and create a fictitious tumor in the atlas domain. Then we solve the image registration problem, and transport the tumor in the patient space back to the atlas. Using the latter data, we solve the inverse tumor problem and update the model parameters. This process is repeated until convergence.

The contributions of this part is implementation of the first coupled inverse tumor and inverse registration problem with support for distributed-memory parallelism and 3D simulations. We present the first-order optimality conditions for the coupling, and discuss the Picard iterative method for solving the optimization problem. We present numerical experiments and quantify the performance of the

coupoling on multiple synthetically generated test cases.

The most notable work in coupling tumor growth model with imaging is GLISTR (60). Coupled imaging algorithms and biophysical models have been used in other applications as well. Examples include cardiovascular diseases (31, 32, 104, 154, 155, 184), oncology (28, 61, 62, 78, 90, 188), and surgical planing (44, 55, 75, 183). Typical image analysis tasks are segmentation, feature extraction for statistical inference (e.g. outlier detection, population statistics, prognosis), and image registration (for segmentation and surgical planing). Such tasks benefit from an integration with biophysical models that introduce pathology-specific prior information.

1.5 HPC Algorithms for Elliptic PDEs

State-of-the-art Poisson Solvers:

As part of this thesis, we also studied the scalability of state-of-the-art algorithms used for solving elliptic PDEs. We first considered the Poisson problem which has many applications in computational science and engineering, and is used in the tumor solvers (as preconditioner for the diffusion equation). We performed a detailed scaling study and compared FFT, with Fast Multipole Method, the geometric multigrid (GMG) and algebraic multi-grid (AMG). We examined and reported results for weak scaling, strong scaling, and time to solution for uniform and highly refined grids (Fig. 1.3). We present results on the Stampede system at the Texas Advanced Computing Center and on the Titan system at the Oak Ridge National Laboratory.

We compare the methods on two different architectures, and discuss two main questions:

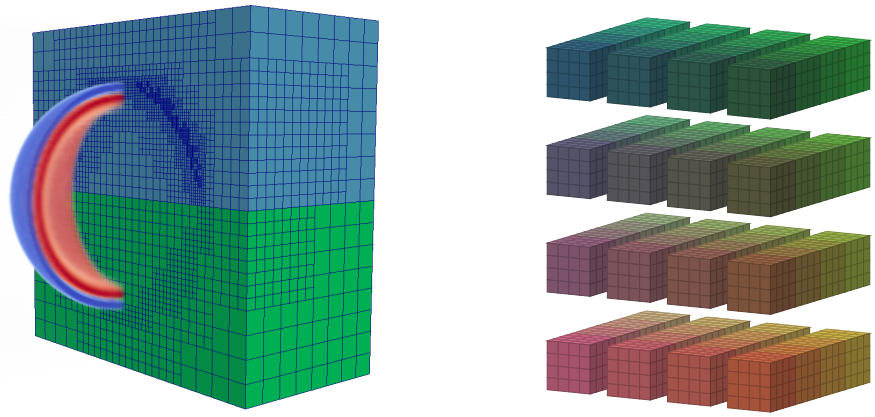


Figure 1.3: (left)Adaptive mesh structure for the forcing term of the Poisson problem (i.e. f). The green cube (bottom quadrant) is the adaptive mesh using 6th order elements and the blue cube (top quadrant) is that of using 14th order elements. The number of unknowns is significantly reduced. (right) Pencil decomposition used in AccFFT

- Which method is faster? What matters the most is the wall-clock time to solution. That is, given f, we would like to evaluate u (typically at a given number of points) to a specified accuracy. We consider two main cases, highly oscillatory fields, for which a regular grid is necessary, and highly localized fields for which adaptively refined meshes are expected to be more effective.
- How does the cost per unknown compare for the different methods given a fixed algebraic accuracy? This test focuses on the complexity estimates which are functions of the problem size, the number of processors, the approximation order, and of course the implementation. We perform weak and strong scaling studies to directly compare the complexity estimates on specific architectures using the same problem size. The goal is to identify whether there are order-of-magnitude differences in the performance between these methods.

In our largest test case, we solved a problem with 600 billion unknowns on 229,379 cores of Titan. We tested all of the methods with different source functions (the

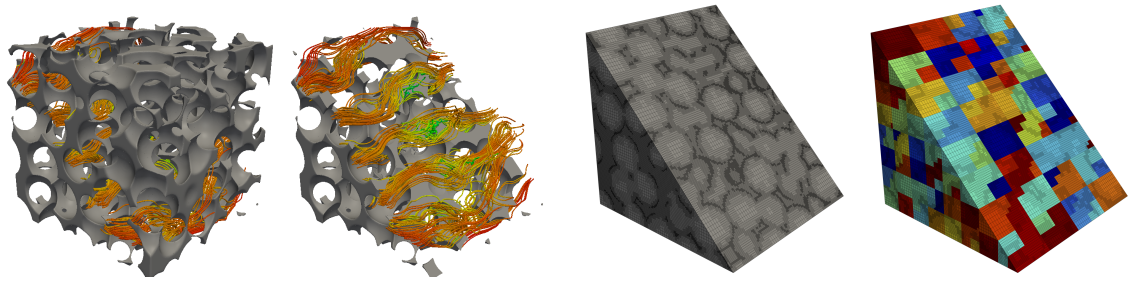


Figure 1.4: Here we illustrate the capabilities of our solver. We simulate Stokes flow through a porous medium. From left: in the **first** figure, the grey color indicates the solid phase geometry and the space in between is the pore space. We also visualize the velocity field using streamlines. In the **second** figure we show the same geometry with clipping to better visualize the streamlines. The **third** figure shows the leaves of the octree and the fourth one the spatial partitioning across different nodes. Details of the algorithm can be found in (107).

right hand side in the Poisson problem). Our results indicated that FFT is the method of choice for smooth source functions that require uniform resolution. However, FFT loses its performance advantage when the source function has highly localized features like internal sharp layers. FMM and GMG considerably outperform FFT for those cases. The distinction between FMM and GMG is less pronounced and is sensitive to the quality (from a performance point of view) of the underlying implementations. In most cases, high-order accurate versions of GMG and FMM significantly outperform their low-order accurate counterparts.

The results of this work has been published in the following paper:

- A. Gholami, D. Malhotra, H. Sundar and G. Biros. *FFT, FMM, or Multigrid? A comparative Study of State-Of-the-Art Poisson Solvers for Uniform and Nonuniform Grids in the Unit Cube*. SIAM Journal on Scientific Computing, Vol. 38 (3), 2016

A volume integral equation Stokes solver

Stokes flow has application in many areas of science from fluid mechanics simulations to preconditioning registration velocity. We present a novel numerical scheme for solving the Stokes equation with variable coefficients in the unit box. We use a volume integral equation formulation. Compared to finite element methods, this formulation decouples the velocity and pressure, generates velocity fields that are by construction divergence free to high accuracy and its performance does not depend on the order of the basis used for discretization. In addition, we employ a novel adaptive fast multipole method for volume integrals to obtain a scheme that is algorithmically optimal. Our scheme supports non-uniform discretizations and is spectrally accurate. To increase per node performance, we have integrated our code with both NVIDIA and Intel accelerators. In the largest scalability test, we solved a problem with 20 billion unknowns, using a 14-order approximation for the velocity, on 2,048 nodes of the Stampede system at the Texas Advanced Computing Center. We achieved 0.656 petaFLOPS for the overall code (23% efficiency) and one petaFLOPS for the volume integrals (33% efficiency). We showcase how our solver can be used to simulate Stokes flow in a porous medium with highly complex pore structure using a penalty formulation to enforce the no slip condition (Fig. 1.4).

The results of this work has been published in the following paper:

- D. Malhotra, A. Gholami, and G. Biros. *A volume integral equation Stokes solver for problems with variable coefficients.* Proceedings of ACM/IEEE Supercomputing Conference (SC14), 2014

1.6 Contributions in CSEM Areas

The contributions of this thesis address all three areas of the Computational Science and Engineering Mathematics (CSEM) program. The detailed contribution in each area is as follows:

Area A: Applicable Mathematics

- Unconditionally stable semi-Lagrangian solver with specific optimizations for source terms in the transport equations of the inverse registration problem;
- A novel Picard scheme for the coupled inverse tumor and registration problem, which decouples the registration velocity and tumor parameters via non-linear iterations;
- Derivation of analytical expressions for the (incremental) forward/adjoint equations with second-order Strang splitting and logistic growth terms; and
- Derivation of the Discretize-Then-Optimize and Optimize-Then-Discretize formulation for reaction-diffusion model with linear elasticity constraints, and verification of the numerical results;

Area B: Numerical analysis and scientific computation

- Distributed-memory implementation that has been scaled to 131K cores on multiple systems;
- Introduction of AccFFT, a new open-source parallel FFT library for CPU and GPUs that has been scaled up to 4,096 GPUs;

- Novel scheme for computing gradient and divergence in parallel that reduce communication volume by $2\times$;
- Algorithmic and parallel scalability on real clinical data up to 200 billion unknowns;
- Design of a state-of-the-art parallel semi-Lagrangian solver applicable to a range of transport problems including the inverse registration problem;
  - Support for high-order interpolation and specific optimization of the cubic interpolation kernel;
  - A novel binning method with negligible overhead which considerably increases temporal locality;
  - SIMD implementation of the cubic kernel for Haswell and KNL architectures as well as GPUs; and
  - Overall speedup of $10\times$ compared to the previous state-of-the-art on the same hardware.
- Minimization of the parallel communication required to compute Hessian matvec of the registration operator by exploiting the stationary registration velocity, and approximation of off-grid spectral operators; and

Area C: Mathematical modeling and applications

- Inverse problem formulation for a reaction-diffusion tumor model that supports the following features:
  - Inhomogeneous and anisotropic diffusion coefficients,

  - Multi-focal tumors, and
  - Multiple time snapshot of tumor concentration data,

- Derivation of the coupled inverse tumor and inverse registration problems and the resulting optimality conditions;
  - Enables registration of healthy reference brain images to tumor-bearing ones; and
  - Addresses the limitation of one time snapshot of the tumor, which is the most common case encountered in practice.

- Development of a mass conserving reaction-diffusion tumor model coupled with linear elasticity equations;

1.7 Outline

In chapter 2, we discuss the reaction-diffusion tumor inverse solver and quantify reconstruction results, using multiple synthetically generated distributions. Then in chapter 3, we discuss the image registration problem and address its computational bottlenecks. We present strong and weak scaling results on several supercomputers for both the inverse registration problem and the inverse tumor solver. In chapter 4, we discuss the formulation for the coupling and provide numerical results. In chapter 5, we discuss how the reaction-diffusion model has to be modified to include "mass-effect", and discuss the corresponding optimality conditions. In chapter 6, we introduce AccFFT, our in-house FFT library for CPU and GPUs. Then in chapter 7, we benchmark and discuss the performance of the scalable methods for the Poisson problem which are used widely in practice: the Fast

Fourier Transform (FFT), the Fast Multipole Method (FMM), the geometric multi-grid (GMG) and algebraic multi-grid (AMG). Finally, in chapter 8, we summarize the contributions of this thesis and discuss the future work.

Chapter 2

Reaction-Diffusion Tumor Model

In this chapter, we will discuss the inverse tumor problem using a reaction-diffusion model for tumor growth. We make a key assumption here that we have the segmentation data from the patient's image with and without tumor. This is the ideal case for solving the inverse problem. In the next chapter we relax this requirement, by coupling the tumor inverse problem with image registration between patient's image and an atlas brain.

The goal of this chapter is to see how the method performs when we have the ground truth data and set the foundation for the coupled problem. In particular, we will discuss the formulation of the forward and inverse problem to estimate distribution of tumor concentration, as well as the magnitude of anisotropic tumor diffusion. We write the problem as a constrained optimization with the reaction-diffusion model that results in a system of nonlinear partial differential equations (PDEs). In our formulation, we estimate the parameters using partially observed, noisy tumor concentration data at two different time instances, along with white matter fiber directions derived from diffusion tensor imaging (DTI). The optimization problem is solved with a reduced Hessian based method (Newton). We present the formulation and outline the numerical algorithms for solv-

This chapter is based on existing publication: *Amir Gholami, Andreas Mang, and George Biros, An inverse problem formulation for parameter estimation of a reaction-diffusion model of low grade gliomas. Journal of mathematical biology, Vol. 72, pp 409-433, 2015.*

A. Gholami derived the algorithms and implemented them from scratch in Matlab and Cpp langauage. A. Mang helped with the visualization and contributed to the introduction. Prof. Biros helped in the editing of the work and provided supervision and guidance.

ing the resulting equations. We test the method using synthetic dataset and compute the reconstruction error for different noise levels and detection thresholds for monofocal and multifocal test cases.

2.1 Tumor Model

The tumor growth model we use has been widely adopted in the literature to model the spatio-temporal spread of cancerous cells on a tissue level (79, 85, 124, 166, 167, 178). It can be stated as follows:

Rate of change of cells in time = proliferation rate + motility (diffusion) rate.

The equivalent mathematical formulation is given by the following partial differential equation:

$$\frac{\partial c}{\partial t} - Dc - R(c) = 0 \text{ in U}, \tag{2.1}$$

$$\frac{\partial c}{\partial n} = 0 \text{ on } \Gamma \times (0,1), \tag{2.2}$$

subjected to an initial distribution $c(t = 0) = c_0$. Here, c is the normalized tumor concentration (i.e. $c \in (0,\ 1]$), D is a linear differential operator modeling the migration of the tumor cells, and $R(c)$ is a nonlinear reaction term, modeling proliferation and necrosis of the tumor cells. Furthermore, $U := \mathcal{B} \times (0, 1]$, where $\mathcal{B}$ is the spatial domain of brain and $(0, 1]$ is the non-dimensional time interval. Moreover, Γ refers to the boundaries of CSF and skull of the brain into which the tumor cells do not infiltrate (178).

The differential operator D that models tumor infiltration is based on a model of

inhomogeneous, anisotropic diffusion:

$$Dc = \nabla \cdot (\mathbf{K}(\mathbf{x}) \nabla c), \tag{2.3}$$

where

$$\mathbf{K}(\mathbf{x}) = k_0(\mathbf{x})\mathbf{I} + k_f \mathbf{T}(\mathbf{x}). \tag{2.4}$$

Here, $k_0(\mathbf{x})$ captures the inhomogeneity due to different diffusion rates in white and grey matter, and $\mathbf{x} = (x, y, z) \in \mathcal{B}$. The inhomogeneity of tumor infiltration follows from the experimental study in (59), in which it was observed that glioma cells have a higher motility rate in white matter as compared to grey matter. This observation has recently been related to the higher cell density in grey matter (157). To account for these different motility rates, the diffusion coefficient is assumed to be inhomogeneous (i.e. k_0 is a function of the location $\mathbf{x}$), with a higher diffusion rate in white matter than in grey matter (156, 166, 168, 178). Therefore, we have

$$k_0(\mathbf{x}) = \begin{cases} k_w, & \text{if } \mathbf{x} \text{ in white matter} \\ k_g, & \text{if } \mathbf{x} \text{ in grey matter} \\ 0, & \text{otherwise} \end{cases}$$

We use a five fold difference, i.e. $\frac{k_w}{k_g} = 5$ (80, 168).

Similarly, it has been suggested that glioma cells have a directional preference in their infiltration. This can be accounted for by introducing an anisotropic diffusion operator (83, 85, 133, 160). The second term in Eq. 2.4, $\mathbf{T}(\mathbf{x})$, captures this behavior. $\mathbf{T}(\mathbf{x})$ is the weighted diffusion tensor derived from diffusion tensor imaging (DTI)

data[1]. We compute $\mathbf{T}(\mathbf{x})$, by scaling the eigendirections and eigenvalues derived from the 3×3 DTI tensor by the so called fractional anisotropy (FA):

$$\mathbf{T} = FA(\lambda_1 \vec{e_1}\vec{e_1}^T + \lambda_2 \vec{e_2}\vec{e_2}^T + \lambda_3 \vec{e_3}\vec{e_3}^T), \tag{2.5}$$

where FA is given by

$$FA = \sqrt{\frac{1}{2}} \frac{\sqrt{(\lambda_1 - \lambda_2)^2 + (\lambda_2 - \lambda_3)^2 + (\lambda_3 - \lambda_1)^2}}{\sqrt{\lambda_1^2 + \lambda_2^2 + \lambda_3^2}}. \tag{2.6}$$

Here, $\vec{e_1}, \vec{e_2}, \vec{e_3}$ and $\lambda_1, \lambda_2, \lambda_3$ are the corresponding eigendirections and eigenvalues, computed at each point in the brain. The diffusion tensor $\mathbf{T}$ is additionally scaled by k_f to account for differences in the diffusion rates between tumor cells and water. Other methods of deriving $\mathbf{T}$ by scaling only the principal direction have been suggested as well (85, 133, 157). For example, $\mathbf{T}$ can have the form of:

$$\mathbf{T} = \lambda_1 \vec{e_1}\vec{e_1}^T. \tag{2.7}$$

In this approach only the principal direction is preferred when there is fiber crossings (the planar case). To address this, (85) suggests a linear scaling of the form:

$$\mathbf{T} = (a_1 \vec{e_1}\vec{e_1}^T + a_2 \vec{e_2}\vec{e_2}^T + a_3 \vec{e_3}\vec{e_3}^T), \tag{2.8}$$

where a_1, a_2, and a_3 are coefficients that depend linearly on k_f and the eigenvalues. Another study in (133) suggests a different derivation of the anisotropic behavior of gliomas from *peanut* or von Mises distribution. In this approach, the macro-

[1]DTI is an MR imaging technique that measures water diffusion tensor at every point in the brain (95)

scopic behavior of glioma cells is derived by considering individual pathways of tumor cells along white matter fibres (74). For example for the 3D case of von Mises-Fisher distribution the diffusion tensor will have the form of:

$$\mathbf{T} = a_1 I + a_2 \lambda_1 e_1 \vec{e}_1{}^T, \tag{2.9}$$

where a_1 and a_2 are two parameters that depend on sensitivity of cells to pathways, cell turning rate, and the degree of randomized turning. In this chapter, we consider Eq. 2.5 and 2.7. But the inverse problem formulation of §2.2 can handle the cases of Eqns. 2.5, and 2.8. For the case of Eq. 2.9, our method requires a slight modification to invert for both a_1 and a_2 instead of just one of them.

A common model (81, 89, 90, 113, 144, 145) for the cell mitosis and necrosis in Eq. 2.1 is the following, self-limiting logistic reaction term:

$$R(c) = \rho c(1 - c), \tag{2.10}$$

where ρ is the reaction coefficient. This model captures the exponential growth of the tumor cells in the areas of low concentration, and necrosis in areas where $c > 1$. We use $\rho = 2$, in non-dimensional form, to test the method in §2.4, which corresponds to $\rho = 0.0006$ per day in dimensional form. For the diffusion coefficient we use $k_0 = 0.1$ in non-dimensional form, which corresponds to $k_0 = 0.01\ \frac{mm^2}{day}$ in dimensional form [2].

As mentioned earlier, with the current clinical imaging technologies only the bulk of the tumor is visible; the full extent of the invasion (physiological tumor boundary) remains undetectable. That is, the tumor concentration is only detectable at

[2]The parameters were selected in the range specified by (161)

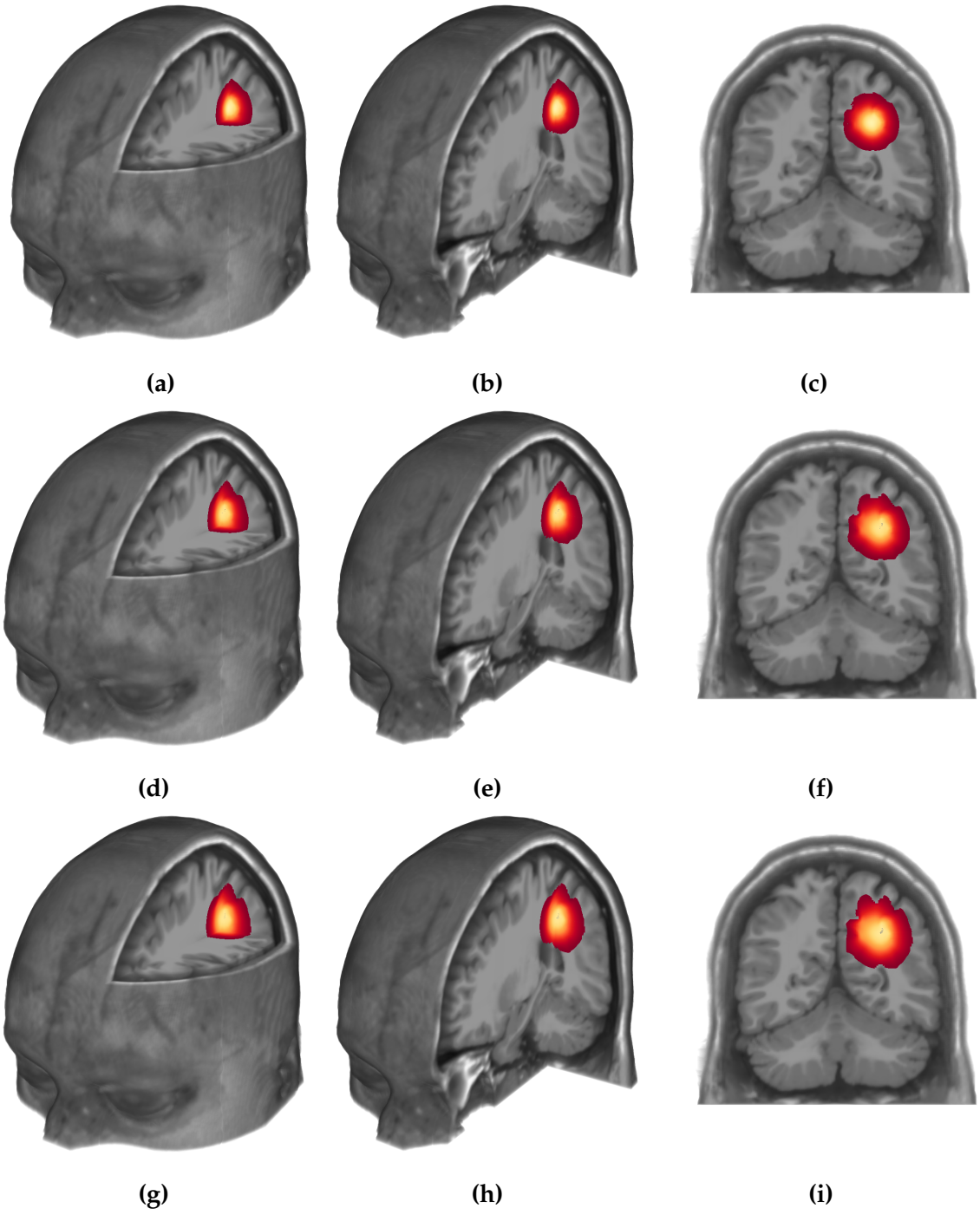


Figure 2.1: Forward simulation of tumor growth using the reaction-diffusion model of Eq. 2.1. From top to bottom, the rows show the tumor distribution at t=0,1 and 2, which in dimensional form corresponds to 0, 14, and 28 months, respectively.

locations in which $c^*(\mathbf{x}) > c_d$. Here $c^*(\mathbf{x})$ refers to the spatial tumor distribution and c_d refers to the detection threshold. This threshold depends on the imaging modality used. In (167) the detection threshold was set to $c_d = 0.16$ and $c_d = 0.8$, for T2-weighted (T2w) and T1-weighted contrast enhanced (T1w-Gd) imaging, respectively. However, there is no consensus in the literature as other values such as $c_d = 0.4$ have been used as well (89, 178). A thorough in vivo/vitro experimental study is needed to identify the correct threshold. To the best of our knowledge, no study on this subject has been published yet. Thus, we will consider a range of values for c_d in our tests. In future, if such data becomes available, one can easily substitute its value and run our inversion algorithm. Nothing changes in our formulation, just the value for c_d. Based on the detection threshold, we define tumor margin, m, to correspond to areas where tumor concentration is below c_d, and above a cutoff value of 1%. Because the tumor concentration is continuous we need to use this cutoff to define the margin.

2.2 Inverse Problem

We consider an inverse problem approach as a modular method of approximating the full extension of tumor invasion, as well as its infiltration rate (k_f). The inverse problem is formulated as a PDE constrained optimization problem:

$$\min_{p,k_f} \mathcal{J} := \frac{1}{2}\|O_0 c_0 - d_0\|_2^2 + \frac{1}{2}\|O_1 c_1 - d_1\|_2^2 + \frac{\beta_p}{2}\|p\|_2^2 \tag{2.11}$$

c	Normalized tumor concentration
c_i	Normalized tumor concentration at $t = i$, $i \in \{0, 1\}$
c_d	Detection threshold
$\mathbf{x} = (x, y, z)$	Spatial location
$\mathcal{B}$	Brain domain
t	Time
D	Tumor diffusion operator
$\mathbf{K}$	Tumor diffusion tensor
k_0	Inhomogeneous diffusion coefficient
k_f	Anisotropic diffusion coefficient
$\mathbf{T}$	DTI weighted diffusion tensor
R	Tumor reaction operator
ρ	Tumor reaction coefficient
d_i	Target tumor concentration at $t = i$, $i \in \{0, 1\}$
O_i	Observation operator at $t = i$, $i \in \{0, 1\}$
p	Reconstruction initial condition parametrization
Φ	Reconstruction initial condition parametrization basis function
β_p	Regularization parameter
α	Adjoint variable

Table 2.1: Basic notations used in this chapter.

subject to:

$$\frac{\partial c}{\partial t} - Dc - R(c) = 0 \qquad \text{in U,} \tag{2.12}$$

$$\frac{\partial c}{\partial n} = 0 \qquad \text{on } \Gamma \times (0,1), \tag{2.13}$$

$$c_0 - \Phi p = 0 \qquad \text{in } \mathcal{B} \text{ (initial condition).} \tag{2.14}$$

Here, O_0 and O_1 are observation operators, $d_1 \in \mathbb{R}^{N_1}$ and $d_0 \in \mathbb{R}^{N_0}$ are the vectors of observation points, $p \in \mathbb{R}^{N_p}$ is a parametrization of tumor distribution at $t = 0$, and β_p is a regularization parameter. All the subscripts refer to time (e.g. O_0 is observation operator at $t = 0$). The commonly used notations are defined in Table 2.1.

The observation operators are defined as:

$$O_0 = \begin{cases} 1, & \text{if } c_0(\mathbf{x}) \geq c_d \\ 0, & \text{otherwise.} \end{cases}$$

$$O_1 = \begin{cases} 1, & \text{if } c_1(\mathbf{x}) \geq c_d \\ 0, & \text{otherwise.} \end{cases}$$

The reason for different observation operators becomes clear now since c_1 has a different distribution than c_0. As mentioned before, only tumor concentrations above a threshold are observable in medical images. Therefore, the second tumor may have a larger or smaller detectable part.

Moreover, the initial distribution of the tumor in Eq. 2.2 is parametrized as a superposition of N_p Gaussian distributions. This mapping is done by $\Phi \in \mathbb{R}^{N_0 \times N_p}$.

The corresponding Lagrangian of this problem is given by:

$$L(c, \alpha, p, k_f) = \mathcal{J} + \int_0^1 \int_{\mathcal{B}} \alpha(\frac{\partial c}{\partial t} - Dc - R(c)) d\mathcal{B}dt + \int_{\mathcal{B}} \alpha_0(c_0 - \Phi p) d\mathcal{B}. \tag{2.15}$$

The first order optimality conditions can be derived by requiring stationarity of the Lagrangian with respect to state c, adjoint α and inversion variables $p,\ k_f$. That

is:

$$\frac{\partial L}{\partial c} = 0 \qquad \text{(adjoint equation)} \tag{2.16}$$

$$\frac{\partial L}{\partial \alpha} = 0 \qquad \text{(state equation)} \tag{2.17}$$

$$\frac{\partial L}{\partial p} = 0 \qquad \text{(inversion equation)} \tag{2.18}$$

$$\frac{\partial L}{\partial k_f} = 0 \qquad \text{(inversion equation)} \tag{2.19}$$

As a result, the adjoint equations are:

$$-\frac{\partial \alpha}{\partial t} - D\alpha - \left.\frac{\partial R(c)}{\partial c}\right|_c \alpha = 0, \qquad \text{in U,} \tag{2.20}$$

$$\frac{\partial \alpha}{\partial n} = 0, \qquad \text{on } \Gamma \times (0,1), \tag{2.21}$$

$$\alpha_1 + O^T(Oc_1 - d_1) = 0, \qquad \text{in } \mathcal{B} \text{ (initial condition).} \tag{2.22}$$

Similarly, the state equations are:

$$\frac{\partial c}{\partial t} - Dc - R(c) = 0, \qquad \text{in U,} \tag{2.23}$$

$$\frac{\partial c}{\partial n} = 0, \qquad \text{on } \Gamma \times (0,1), \tag{2.24}$$

$$c_0 - \Phi p = 0, \qquad \text{in } \mathcal{B} \text{ (initial condition).} \tag{2.25}$$

Finally, the inversion equations are:

$$\beta_p p - \Phi^T \alpha_0 + \Phi^T O_0^T (O_0 \Phi p - d_0) = 0 \tag{2.26}$$

$$\int_0^1 \int_{\mathcal{B}} (\mathbf{T} \nabla c) \cdot (\nabla \alpha) d\mathcal{B} dt = 0. \tag{2.27}$$

Equations 2.2-2.2 form a system of nonlinear PDEs in space and time. In general,

they are not amenable to analytical solutions, and have to be solved with a numerical method. We use a Gauss-Newton method that corresponds to an inexact linearization of the equations. After deriving the optimality conditions, then we discretize and solve the resulting PDEs numerically (86). The scheme can be summarized as follows. Assume c^0, α^0, p^0 and k_f^0 to be the initial guess of the iterative scheme (throughout this chapter superscripts refer to the iteration number of the iterative solver and should not be confused with time, which are always specified by subscripts). The updates to these variables (denoted by $\tilde{c}$, $\tilde{\alpha}$, $\tilde{p}$ and $\tilde{k_f}$) can be found by solving the second order optimality system (i.e. the second variation of the Lagrangian):

$$-\frac{\partial\tilde{\alpha}}{\partial t} - D\tilde{\alpha} - \left.\frac{\partial R}{\partial c}\right|_{c^0}\tilde{\alpha} - \left.\frac{\partial^2 R}{\partial c^2}\right|_{c^0}\tilde{c}\alpha^0 - \nabla\cdot(\tilde{k_f}\mathbf{T}\nabla\alpha^0) = 0 \tag{2.28}$$

$$\tilde{\alpha}_1 + O^T O\tilde{c}_1 = 0 \tag{2.29}$$

$$\beta_p\tilde{p} - \Phi^T\tilde{\alpha}_0 + \Phi^T O_0^T(O_0\Phi\tilde{p} - d_0) = -(\beta_p p^0 - \Phi^T\alpha_0^0 + \Phi^T O_0^T(O_0\Phi p^0 - d_0)) \tag{2.30}$$

$$\frac{\partial\tilde{c}}{\partial t} - D\tilde{c} - \left.\frac{\partial R}{\partial c}\right|_{c^0}\tilde{c} - \nabla\cdot(\tilde{k_f}\mathbf{T}\nabla c^0) = 0 \tag{2.31}$$

$$\tilde{c}_0 - \Phi\tilde{p} = 0 \tag{2.32}$$

$$\int_0^1\int_{\mathcal{B}}\Big((\mathbf{T}\nabla\tilde{c})\cdot(\nabla\alpha^0) + (\mathbf{T}\nabla c^0)\cdot(\nabla\tilde{\alpha})\Big)d\mathcal{B}dt = 0. \tag{2.33}$$

It is useful to rewrite the previous equations using linear operators (definitions are

given in §2.5):

$$J^T\tilde{\alpha} + N\tilde{c} + Z^T\tilde{k_f} = 0, \tag{2.34}$$

$$B_p\tilde{p} - \Phi^T\tilde{\alpha} = -g_p, \tag{2.35}$$

$$J\tilde{c} + W^T\tilde{k_f} - \Phi\tilde{p} = 0, \tag{2.36}$$

$$Z\tilde{c} + W\tilde{\alpha} = -g_k, \tag{2.37}$$

which in matrix form can be written as:

$$\begin{bmatrix} N & 0 & J^T & Z^T \\ 0 & B_p & -\Phi^T & 0 \\ J & -\Phi & 0 & W^T \\ Z & 0 & W & 0 \end{bmatrix}_{(c^0,\alpha^0,p^0,k_f^0)} \begin{bmatrix} \tilde{c} \\ \tilde{p} \\ \tilde{\alpha} \\ \tilde{k_f} \end{bmatrix} = \begin{bmatrix} 0 \\ -g_p \\ 0 \\ -g_k \end{bmatrix}_{(c^0,\alpha^0,p^0,k_f^0)} \tag{2.38}$$

To solve this linear system, we use a reduced Hessian formulation, by eliminating $\tilde{c}$ and $\tilde{\alpha}$ from the system. As a result we obtain:

$$\begin{bmatrix} H_{pp} & H_{pk} \\ H_{kp} & H_{kk} \end{bmatrix} \begin{bmatrix} \tilde{p} \\ \tilde{k_f} \end{bmatrix} = \begin{bmatrix} -g_p \\ -g_k \end{bmatrix}_{(c^0,\alpha^0,p^0,k_f^0)}, \tag{2.39}$$

where the reduced Hessians (i.e. the individual block matrices on the left hand side in Eq. 2.39) are defined in §2.5 (see Eqns. 2.5-2.5).

To compute $\tilde{p}$ we solve the following equation using an iterative scheme such as Generalized Minimal Residual (GMRES) or Conjugate Gradient (CG) method. Because the Hessian is symmetric positive definite, we use CG to solve the Schur

complement of Eq. 2.39, given by:

$$(H_{pp} - H_{pk}H_{kk}^{-1}H_{kp})\tilde{p} = H_{pk}H_{kk}^{-1}g_k - g_p. \tag{2.40}$$

Then $\tilde{k_f}$ can be found by substituting the computed $\tilde{p}$ in the following equation:

$$\tilde{k_f} = -H_{kk}^{-1}H_{kp}\tilde{p} - H_{kk}^{-1}g_k. \tag{2.41}$$

Since the problem is nonlinear, this update process must be repeated. That is, one should update (p^0, k_f^0) with $(\tilde{p}, \tilde{k_f})$ (using a globalization scheme such as a line search) and repeat the solution process (i.e. solve Eq. 2.40 and 2.41 again).

2.3 Numerical Methods

Following our earlier work (81), we use a Strang second-order time splitting method (162) to numerically solve the PDEs. Splitting methods provide exclusive benefits for solving complex PDEs that involve different operators, without loss of accuracy. Here we explain how the forward PDE (i.e. Eq. 2.1) is solved with this method. The other PDEs of the optimality systems in §2.2 can be solved in a similar fashion. Let c^n denote the tumor distribution at the n-th time step. To find c^{n+1} the following steps have to be performed:

1. Solve $\frac{\partial c}{\partial t} = Dc$ over time $\Delta t/2$ using a second order implicit Crank-Nicolson method with c^n as initial condition, to obtain $c^\dagger$.

2. Solve $\frac{\partial c}{\partial t} = R(c)$ over time Δt analytically/numerically with $c^\dagger$ as initial condition, to obtain $c^{\dagger\dagger}$.

3. Solve $\frac{\partial c}{\partial t} = Dc$ using a second order implicit Crank-Nicolson method over time $\Delta t/2$ with $c^{\dagger\dagger}$ as initial condition, to obtain c^{n+1}.

This scheme can more compactly be written as:

$$c^{n+1} = S_D^{\frac{\Delta t}{2}} S_R^{\Delta t} S_D^{\frac{\Delta t}{2}} c^n, \tag{2.42}$$

where S_i^t, $i = D, R$, is the numerical PDE solver corresponding to each of the above steps. For instance, $S_D^{\frac{\Delta t}{2}} c^n$ corresponds to the first step and applies the diffusion operator to c^n and gives $c^\dagger$ as its output (the interested reader is referred to (162) and (147) for more details). To solve for each of these steps we use a pseudo-spectral method. The space is discretized into $N = 64^3$ nodes and $N_t = 10$ time steps. The space discretization corresponds to a domain of $64\ mm^3$ centered around the tumorous region[3]. Finally, we note that the overall numerical scheme is second order in time and has spectral accuracy in space. We have tested and verified this. A key advantage of the splitting method is that the nonlinear logistic growth model has an analytical solution. As a result, $S_R^{\Delta t}$ can be computed with an accuracy that is down to machine precision and with negligible computational cost (this not only reduces the computational complexity, but also allows for very accurate approximations to the Hessian operator). For PDEs that involve the derivative of the reaction term, one can simply use a second order Crank-Nicolson method in the absence of an exact solution. For the diffusion part $S_D^{\frac{\Delta t}{2}}$ we use pseudo-spectral method in space and second order Crank-Nicolson scheme in time. For instance, $S_D^{\frac{\Delta t}{2}} c^n$ is solved to obtain $c^\dagger$ as follows:

[3]This area can be increased for highly infiltrative tumors that spread through a larger portion of the brain.

$$(I - \frac{\Delta t}{4}\nabla \cdot \mathbf{K}\nabla)c^{\dagger} = (I + \frac{\Delta t}{4}\nabla \cdot \mathbf{K}\nabla)c^{n}. \tag{2.43}$$

The implicit scheme of Eq. 2.43 results in a symmetric system that is solved iteratively using the Conjugate Gradient method. A key advantage of implicit methods is that they are, in contrast to explicit methods, unconditionally stable. On the downside, one must solve a system of equations at each iteration.

Hessian Preconditioner.

Here, we briefly explain how we solve and precondition Eq. 2.40. Solving this equation is one of the major bottlenecks of the algorithm, as each application of the Hessian involves the solution of the forward and adjoint PDEs in time. To prevent any ambiguity, we refer to $(H_{pp} - H_{pk}H_{kk}^{-1}H_{kp})\tilde{p}$ as the *Hessian matvec* for a given $\tilde{p}$.

We precondition Eq. 2.40 to increase its convergence rate. As a result, the number of necessary Hessian matvecs will decrease, thus reducing time to solution. A good preconditioner is one that has low overhead of forming and applying, while increasing the convergence rate considerably. An extreme case for a preconditioner is the inverse of the Hessian operator, which would decrease the number of iterations to one, but is too expensive to form and apply. On the other hand, the identity operator is a preconditioner with no overhead and obviously no effect. With this in mind, we preconditioned Eq. 2.40 with an analytical approximation to H_{pp}. The analytic expression is derived by approximating numerical operators of H_{pp} (defined in Eq. 2.5). For this, we use the average value of the variable diffusion coefficient tensor $\mathbf{K}(\mathbf{x})$. This allows an analytical solution to the diffusion operator in the splitting scheme. The other terms such as the reaction operator are

solved numerically as before. The fact that we are not solving an implicit system of equations for the diffusion operator reduces the overhead of forming and applying this preconditioner considerably. This is the only approximation that we are doing and the rest of the operators are the same as the numerical Hessian. This preconditioner performs very well and thus reduces the number of CG iterations to solve Eq. 2.5 by a factor of 5 (from about 25 to 5).

Since we are solving a nonlinear optimization problem, Eq. 2.40 has to be repeatedly solved (we refer to these repetitions as Newton iterations). The number of Newton iterations depends on how far the initial guess of the inversion parameters are from the actual (true) solution. The fact that the Hessian preconditioner performs very well, shows that it is actually a good approximation to the true Hessian. As a result, one can use it to first solve for a good starting point with negligible computational overhead. This means that one first solves Eq. 2.40 and Eq. 2.41 to compute p^0 and k_f^0 by using analytical approximations to the Hessian operators, instead of using their true numerical expressions. The approximation is exactly the same as explained above; that is, we only approximate the costly diffusion operator with an analytical expression by using the average value of the diffusion tensor. With these starting values of p^0 and k_f^0 we then solve Eq. 2.40 and Eq. 2.41 with the correct numerical forms of the Hessian operators. This process is quite effective as it reduces the number of Newton iterations by a factor of 3 (from about 15 to 5).

Regularization Parameter.

Inherent in any inverse problem, is a regularization operator that limits noise amplification. We use a Tikhonov-type regularization model in Eq. 2.11

for this purpose (176). If the problem is under-regularized (i.e. $\beta_p \ll 1$), the solution will be dominated by the inherent noise in the data. On the other hand, over-regularization (i.e. $\beta_p \gg 1$) will result in a poor fit to the data. Therefore, it is necessary to choose an optimal regularization parameter, somewhere in-between these two extremes. It is possible to choose β_p heuristically. However, we will use a systematic approach to compute an optimal β_p. The L-curve method is one of the widely used methods for this purpose (67, 68). We provide a plot of an exemplary L-curve in Fig. 2.2. We obtained $\beta_p = 0.01$ as an optimal regularization parameter for test case 2, with $c_d = 0.2$, and $\eta = 5\%$ (the test case will be defined in §2.4). This process needs to be repeated for new datasets. Although this method has an expensive computational cost, it allows a systematic selection of the regularization parameter. It is also possible to do this process inside the iterative solver, where one starts from the tail of the L-Curve (large regularization) and reduces the regularization parameter as the solution converges (43). In this fashion, part of the computational cost will be overlapped in the non-linear iterations.

2.4 Results

Table 2.2: Reconstruction results for test case 1. For this test case, the target value of the anisotropic diffusion coefficient k_f is known. The reconstruction errors, ϵ , are given at different time frames, for different observation thresholds c_d and different noise levels η.

c_d	η	ϵ_0	JI_0	ϵ_1	JI_1	ϵ_2	JI_2
0.10	1%	5.7e-02	0.794	3.5e-02	0.840	3.4e-02	0.821
0.10	5%	6.2e-02	0.806	3.9e-02	0.854	3.6e-02	0.835
0.10	10%	7.5e-02	0.818	5.4e-02	0.865	4.9e-02	0.846
0.20	1%	6.1e-02	0.816	4.2e-02	0.857	4.1e-02	0.840
0.20	5%	6.7e-02	0.825	4.7e-02	0.866	4.3e-02	0.849
0.20	10%	8.3e-02	0.832	6.3e-02	0.874	5.7e-02	0.857
0.40	1%	6.8e-02	0.845	4.4e-02	0.889	4.0e-02	0.871
0.40	5%	7.7e-02	0.849	5.3e-02	0.896	4.6e-02	0.877
0.40	10%	1.0e-01	0.852	7.7e-02	0.899	6.8e-02	0.880

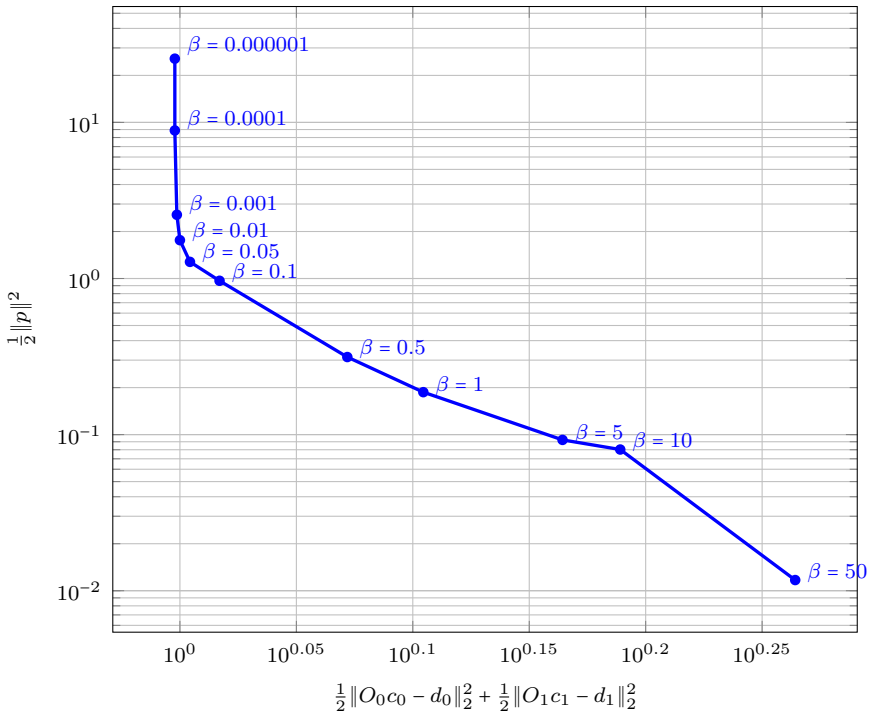


Figure 2.2: L-curve for choosing the regularization parameter. The corner of the L-curve yields $\beta_p = 0.01$ as the optimal regularization parameter for $c_d = 0.2$ and $\eta = 5\%$.

Setup.

We consider tumor distributions that are created synthetically using the reaction-diffusion model for the target distribution. We apply our inversion algorithm to noisy, partial observations of these targets, limited to two time frames, and compute the corresponding reconstructions. This synthetic analysis allows us to compute the exact errors between the reconstructions and the target, which would otherwise not be possible. As mentioned before, only part of the target tumor is detectable. Our observation operator (i.e. O_0 and O_1) captures this, by selecting tumors at points that have a concentration above the detection threshold. Since there is no consensus in the literature about the exact value of this detection threshold (89, 167, 178), we will consider different values to test our algorithm.

Moreover, the data derived from any imaging modality will contain noise. To account for this, we add different levels of white noise (denoted by η) to the observed data (target distribution).

To match our implementation to virtual brain anatomy, we use the BrainWeb atlas (spatial resolution: $1mm \times 1mm \times 1mm$) (30). Moreover, we use the diffusion tensor imaging data provided by the LONI lab of the University of Southern California (122).

To test our inversion algorithm, we consider the following four test cases:

Test case 1: Reconstruct full tumor distribution with known anisotropic diffusion coefficient k_f (using Eq. 2.5 for the diffusion tensor).

Test case 2: Reconstruct full tumor distribution as well as anisotropic diffusion coefficient k_f (using Eq. 2.5 for the diffusion tensor).

Test case 3: Same as test case 2, but for a multifocal tumor (using Eq. 2.5 for the diffusion tensor).

Test case 4: Reconstruct full tumor distribution as well as anisotropic diffusion coefficient k_f (using Eq. 2.7 for the diffusion tensor).

The inversion in all the test cases is derived by partially observed, noisy data of the target tumor distribution at two consecutive time frames of $t = 0$ and $t = 1$, respectively. Test case 1 is rather unrealistic, since there is currently no way to measure k_f in vivo. However, in test case 2 we assume no knowledge of the anisotropic diffusion rate. We include it as an unknown inversion parameter in our algorithm. Other researchers have considered manual tuning of the method with different k_f and then selected the one that has the best fit (85, 157). However, our approach

is a systematic one and does not need any manual parameter tuning. In the last test case we consider a multifocal tumor. This test case is to demonstrate that our inversion algorithm works irrespective of mono- or multifocality of the tumor. Moreover, to show that our method works with other diffusion tensors, we test the inversion using Eq. 2.7 for $\mathbf{T}$.

Table 2.3: Detailed reconstruction results for test case 2. The reconstruction errors, ϵ , are given at different time frames, for different observation thresholds c_d and different noise levels η.

c_d	η	ϵ_{kf}	ϵ_0	JI_0	ϵ_1	JI_1	ϵ_2	JI_2
0.10	1%	5.4e-02	5.7e-02	0.793	3.4e-02	0.804	4.6e-02	0.800
0.10	5%	5.1e-02	6.2e-02	0.806	3.8e-02	0.818	4.8e-02	0.814
0.10	10%	6.8e-02	7.5e-02	0.818	5.3e-02	0.831	5.8e-02	0.827
0.20	1%	1.3e-02	6.1e-02	0.816	4.2e-02	0.829	5.2e-02	0.824
0.20	5%	1.3e-02	6.7e-02	0.825	4.7e-02	0.840	5.4e-02	0.834
0.20	10%	1.3e-02	8.3e-02	0.832	6.3e-02	0.848	6.5e-02	0.843
0.30	1%	2.4e-03	6.4e-02	0.828	4.5e-02	0.844	5.3e-02	0.837
0.30	5%	3.3e-03	7.1e-02	0.834	5.0e-02	0.853	5.6e-02	0.845
0.30	10%	1.2e-02	9.1e-02	0.839	6.9e-02	0.862	6.9e-02	0.853
0.40	1%	1.4e-02	6.8e-02	0.844	4.4e-02	0.866	5.2e-02	0.858
0.40	5%	3.6e-02	7.6e-02	0.849	5.2e-02	0.873	5.5e-02	0.864
0.40	10%	6.6e-02	9.8e-02	0.853	7.5e-02	0.879	7.2e-02	0.869

Table 2.4: Detailed reconstruction results for multifocal test case 3. The reconstruction errors, ϵ, are given at different time frames, for different observation thresholds c_d and different noise levels η.

c_d	η	ϵ_{kf}	ϵ_0	JI_0	ϵ_1	JI_1	ϵ_2	JI_2
0.10	1%	1.0e-02	9.7e-02	0.695	6.1e-02	0.748	5.5e-02	0.726
0.10	5%	2.3e-02	1.0e-01	0.695	6.6e-02	0.753	6.1e-02	0.726
0.10	10%	3.8e-02	1.1e-01	0.688	7.9e-02	0.747	7.4e-02	0.722
0.20	1%	8.0e-02	1.0e-01	0.748	7.2e-02	0.795	6.6e-02	0.775
0.20	5%	8.0e-02	1.1e-01	0.742	7.8e-02	0.790	7.3e-02	0.771
0.20	10%	8.1e-02	1.2e-01	0.739	9.2e-02	0.790	8.7e-02	0.769
0.30	1%	7.4e-02	1.2e-01	0.764	9.0e-02	0.801	8.5e-02	0.785
0.30	5%	6.5e-02	1.2e-01	0.766	9.6e-02	0.804	9.1e-02	0.788
0.30	10%	5.5e-02	1.4e-01	0.760	1.1e-01	0.804	1.1e-01	0.786
0.40	1%	1.7e-01	1.3e-01	0.763	1.0e-01	0.810	9.4e-02	0.790
0.40	5%	1.7e-01	1.3e-01	0.766	1.0e-01	0.807	9.7e-02	0.789
0.40	10%	1.7e-01	1.4e-01	0.773	1.1e-01	0.811	1.1e-01	0.793

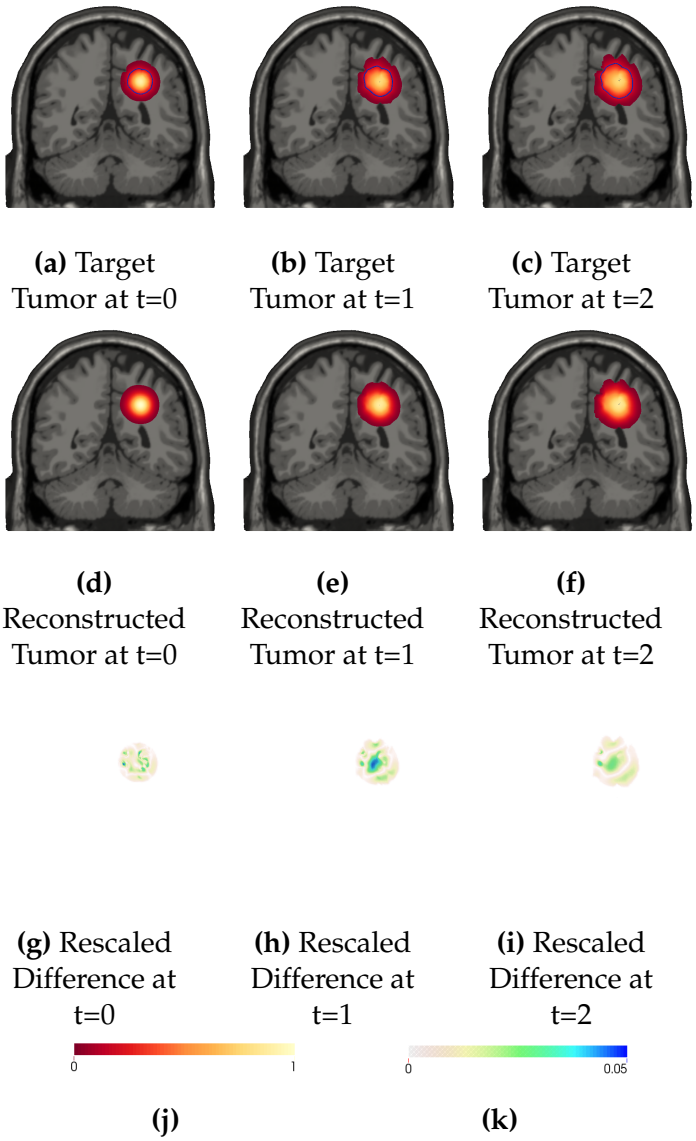
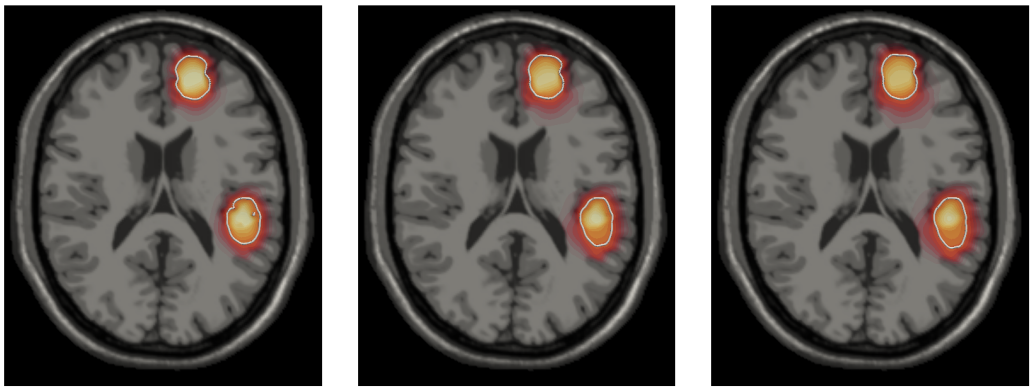
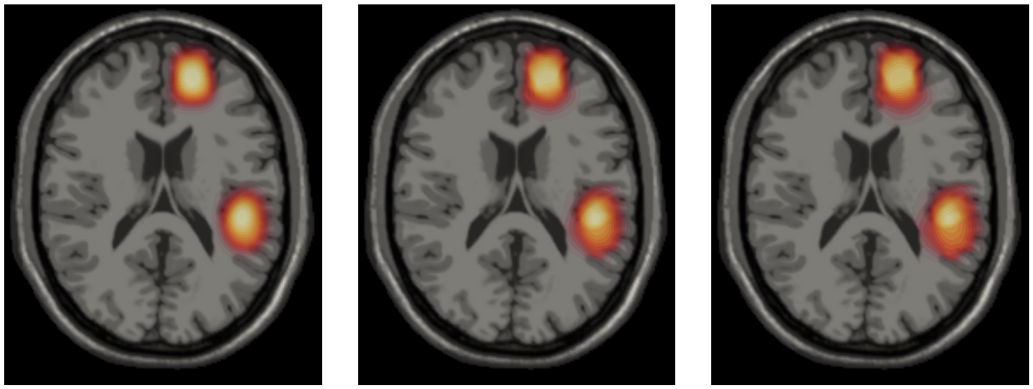


Figure 2.3: Reconstruction results for test case 2. The top row shows the target tumor distribution at different points in time. The three columns show the tumor distribution at $t = 0, 1$, and 2 which in dimensional form corresponds to 0, 14, and 28 months, respectively. The blue contour indicates an observable tumor concentration of $c_d = 0.2$. The reconstruction is driven by noisy observations ($\eta = 5\%$) of the cell density on the inside of this contour (at $t = 0$ and $t = 1$). The second row shows the reconstruction results at the corresponding times. The reconstruction relative errors are 6.6%, 4.5% and 5.3%, respectively. The last row shows the same difference but with a rescaled colormap (n).



(a) Target Tumor at t=0 | **(b)** Target Tumor at t=1 | **(c)** Target Tumor at t=2



(d) Reconstructed Tumor at t=0 | **(e)** Reconstructed Tumor at t=1 | **(f)** Reconstructed Tumor at t=2

Figure 2.4: Reconstruction results for multifocal test case 3. The top row shows the target tumor distribution at different times. The three columns show the tumor distribution at $t = 0, 1,$ and 2 which in dimensional form corresponds to 0, 5, and 10 months, respectively. The white contour indicates the observable tumor concentration of $c_d = 0.2$. The bottom row shows the reconstruction results at $t = 0, 1,$ and 2 for $\eta = 5\%$. The reconstruction errors are 10.3%, 6.97% and 6.56%, respectively.

Table 2.5: Detailed reconstruction results for multifocal test case 4. This test case is different, in that we use Eq. 2.7 for $\mathbf{T}$. The reconstruction errors, ϵ, are given at different time frames, for different observation thresholds c_d and different noise levels η.

c_d	η	ϵ_{kf}	ϵ_0	JI_0	ϵ_1	JI_1	ϵ_2	JI_2
0.20	1%	1.09e-02	7.58e-02	0.806	4.81e-02	0.820	5.69e-02	0.814
0.20	5%	2.39e-02	8.36e-02	0.815	5.51e-02	0.829	6.07e-02	0.824
0.20	10%	4.23e-02	1.04e-01	0.823	7.57e-02	0.839	7.49e-02	0.833
0.40	1%	6.66e-03	6.96e-02	0.834	4.60e-02	0.857	5.74e-02	0.848
0.40	5%	8.65e-03	7.52e-02	0.838	5.03e-02	0.863	5.90e-02	0.853
0.40	10%	1.01e-02	9.02e-02	0.841	6.60e-02	0.867	6.98e-02	0.857

Performance Measures.

To quantify the reconstruction performance, we report the mismatch between the target and reconstructed tumor distributions:

$$\epsilon_i = \frac{\|c_i - c_i^*\|_2}{\|c_i^*\|_2}, \tag{2.44}$$

where c_i^* is the target tumor distribution and c_i is the reconstructed one, at time $t = i$. This mismatch error is reported for the two consecutive time frames, at which the data was observed (i.e. $t = 0$ and $t = 1$). We also report the error for an additional time point $t = 2$. This test is included to study how well does the reconstruction capture the growth pattern of an untreated tumor (i.e. the tumor growth prediction capabilities of our method). Furthermore, we provide Jaccard Index (JI) to assess how well our method approximates the tumor margin:

$$\text{JI}_i = \frac{|H(m_i) \cap H(m_i^*)|}{|H(m_i) \cup H(m_i^*)|}, \tag{2.45}$$

where H is the Heaviside function, m_i is the reconstructed margin at time $t = i$, and m_i^* is the target one. Note that we are using the tumor margin, m, instead of the full tumor distribution, c. This is because the tumor margin does not involve areas

above the detection threshold. Those areas have a JI of one, and including them will increase JI artificially. This is specially important for cases where c_d is small, and using m will give a better metric of how well the margin is reconstructed.
For test case 2 and 3, we also report the error in the reconstruction of k_f compared to the target anisotropy coefficient k_f^*:

$$\epsilon_{kf} = \frac{|k_f - k_f^*|}{|k_f^*|}, \tag{2.46}$$

The target tumor distribution c^* is shown in Fig. 2.1 at these three time frames. The results are computed by solving the forward problem in 3D. For the test case 1 and 2 we perform the reconstruction in 3D. For the test case 3 (multifocal tumor) we limit the reconstruction to a 2D slice.

Numerical Results.

The reconstruction results for test case 1 are shown in Table 2.2. We report the relative reconstruction error (computed according to Eq. 2.44) at different time frames, for different threshold values, c_d, and different noise levels, η. Note that only the two time frames $t = 0, 1$ are used to solve the inverse problem. The error given at $t = 2$ demonstrates how well the reconstruction matches the target if the tumor was left untreated. The reconstruction error increases with an increasing detection threshold. This behavior is expected, since a higher threshold means less data is available to drive the inversion. Moreover, the reconstruction error is higher, as noise is increased. However, the difference is not significant, because the regularization operator counter balances noise amplifications.

But what if the value of the anisotropic diffusion rate is unknown as well? This is exactly what we study in test case 2. The corresponding errors are shown in

Table 2.3. As one can see, the inversion algorithm approximates the correct value of k_f with a very good accuracy. As a result, the errors in tumor distribution reconstruction at $t = 0$ and $t = 1$ are comparable to those reported for test case 1. Figure 2.3 illustrates how well the reconstruction compares to the target distribution for test case 2 with $c_d = 0.20$ and $\eta = 5\%$. The target distributions are shown with a blue contour marking the area where the tumor concentration is equal to the detection threshold. The reconstruction, shown in the second row, is computed using only the target cell density that is within this contour. As can be seen, the reconstruction qualitatively captures the growth pattern very well.

As a proof-of-concept, we test how the inversion performs when we consider a multifocal tumor as shown in Fig. 2.4. This test case is performed in 2D, in contrast to case 1 and 2. The corresponding errors are reported in Table 2.4. Overall, we observe the same behavior as in the previous test cases. Similarly, to show that our scheme works for other diffusion tensors, we consider the case of Eq. 2.7 for $\mathbf{T}$. In this case, only the dominant eigendirection is considered. The results are given in Table 2.5.

2.5 Conclusions

We presented an inverse problem formulation to determine the full extent of tumor infiltration in the brain based on a PDE-constrained optimization problem. The key quantities of interest are (i) the full extent of tumor invasion, and (ii) the rate of anisotropic diffusion. We used a nonlinear reaction-diffusion model for glioma growth, and solved the optimization problem with a reduced space Hessian method. State of the art numerical techniques were presented to speed up the time to solution. The design criteria for these techniques were low computational

cost and robustness. We tested the resulting algorithm, using synthetic tumors with different levels of noise and different detection thresholds.

In the next chapter we will discuss the distributed-memory implementation of this framework, and present weak and strong scaling results performed on multiple systems.

Appendix A: Operator Definitions

The definitions of operators in Eqs 2.2, 2.2, 2.2, and 2.2 are as follows:

$$J^T\tilde{\alpha} := -\frac{\partial\tilde{\alpha}}{\partial t} - D\tilde{\alpha} - \left.\frac{\partial R}{\partial c}\right|_{c^0}\tilde{\alpha} + \int_0^1 \delta(t-T)\tilde{\alpha}dt \tag{2.47}$$

$$N\tilde{c} := -\left.\frac{\partial^2 R}{\partial c^2}\right|_{c^0}\tilde{c}\alpha^0 + O^T O\int_0^1 \delta(t-T)\tilde{c}dt \tag{2.48}$$

$$Z^T\tilde{k_f} := -\nabla\cdot(\tilde{k_f}\mathbf{T}\nabla\alpha^0) \tag{2.49}$$

$$g_p := Bu - \Phi^T\alpha_0^0 \tag{2.50}$$

$$Bu := (\beta_p + \Phi^T O_0^T O_0\Phi)p \tag{2.51}$$

$$Z\tilde{c} := \int_0^1\int_\Omega(\mathbf{T}\nabla\tilde{c})\cdot(\nabla\alpha^0)d\Omega dt \tag{2.52}$$

$$W\tilde{\alpha} := \int_0^1\int_\Omega(\mathbf{T}\nabla c^0)\cdot(\nabla\tilde{\alpha})d\Omega dt \tag{2.53}$$

$$g_k := \int_0^1\int_\Omega(\mathbf{T}\nabla c^0)\cdot(\nabla\alpha^0)d\Omega dt \tag{2.54}$$

$$J\tilde{c} := \frac{\partial\tilde{c}}{\partial t} - D\tilde{c} - \left.\frac{\partial R}{\partial c}\right|_{c^0}\tilde{c} + \int_0^1 \delta(t)\tilde{c}dt \tag{2.55}$$

$$W^T\tilde{k_f} := -\nabla\cdot(\tilde{k_f}\mathbf{T}\nabla c^0) \tag{2.56}$$

The reduced Hessians of Eq. 2.39 are defined as follows:

$$H_{pp} = B + \Phi^T J^{-T} N J^{-1} \Phi, \tag{2.57}$$

$$H_{pk} = -\Phi^T J^{-T} (-Z^T + N J^{-1} W^T), \tag{2.58}$$

$$H_{kp} = (Z - W J^{-T} N) J^{-1} \Phi, \tag{2.59}$$

$$H_{kk} = -Z J^{-1} W^T + W J^{-T} (-Z^T + N J^{-1} W^T). \tag{2.60}$$

To compute the matvec of $H_{pp}\tilde{p}$ one needs to take the following steps:

1. $J^{-1}\Phi$:

   Solve Eq. 2.2 with $\tilde{k_f} = 0$ and initial condition of $\tilde{c}_0 = \Phi\tilde{p}$ to get $\tilde{c}$

2. $-J^{-T} N J^{-1} \Phi$:

   Solve Eq. 2.2 with $\tilde{k_f} = 0$ and initial condition of $\tilde{\alpha}_1 = -O^T O \tilde{c}_1$ to get $\tilde{\alpha}$

3. $B + \Phi^T J^{-T} N J^{-1} \Phi$:

   Compute $-\Phi^T \alpha_0$ and add Bu

To compute the matvec of $H_{pk}\tilde{k_f}$ one needs to take the following steps:

1. $J^{-1} W^T$:

   Solve Eq. 2.2 with zero initial condition to get $\tilde{c}$

2. $J^{-T}(-Z^T + N J^{-1} W^T)$:

   Solve Eq. 2.2 with initial condition of $\tilde{\alpha}_1 = -O^T O \tilde{c}_1$ to get $\tilde{\alpha}$

3. $-\Phi^T J^{-T}(-Z^T + NJ^{-1}W^T)$:

   Compute $-\Phi^T \alpha_0$

To compute the matvec of $H_{ku}\tilde{p}$ one needs to take the following steps:

1. $J^{-1}\Phi$:

   Solve Eq. 2.2 with $\tilde{k_f} = 0$ and initial condition of $\tilde{c}_0 = \Phi\tilde{p}$ to get $\tilde{c}$

2. $ZJ^{-1}\Phi$:

   Compute $\int_0^1 \int_\Omega (\mathbf{T}\nabla\tilde{c}) \cdot (\nabla\alpha^0) d\Omega dt$

3. $-J^{-T}NJ^{-1}\Phi$:

   Solve Eq. 2.2 with $\tilde{k_f} = 0$ and initial condition of $\tilde{\alpha}_1 = -O^T O\tilde{c}_1$ to get $\tilde{\alpha}$

4. $-WJ^{-T}NJ^{-1}\Phi$:

   Compute $\int_0^1 \int_\Omega (\mathbf{T}\nabla c^0) \cdot (\nabla\tilde{\alpha}) d\Omega dt$

5. Add 2 and 4

To compute the matvec of $H_{kk}\tilde{k_f}$ one needs to take the following steps:

1. $J^{-1}W^T$:

   Solve Eq. 2.2 with zero initial condition to get $\tilde{c}$

2. $ZJ^{-1}W^T$:

   Compute $\int_0^1 \int_\Omega (\mathbf{T}\nabla\tilde{c}) \cdot (\nabla\alpha^0) d\Omega dt$

3. $J^{-T}(-Z^T + NJ^{-1}W^T)$:

   Solve Eq. 2.2 with initial condition of $\tilde{\alpha}_1 = -O^T O\tilde{c}_1$ to get $\tilde{\alpha}$

4. $WJ^{-T}(-Z^T + NJ^{-1}W^T)$:

   Compute $\int_0^1 \int_\Omega (\mathbf{T}\nabla c^0) \cdot (\nabla\tilde{\alpha}) d\Omega dt$

5. Add 2 and 4.

Appendix B: Fictitious Domain Method

We use a fictitious domain method in which the original brain domain, $\mathcal{B}$, is extended to a cubic box, denoted by Ω (81, 113, 178). The original homogeneous boundary conditions imposed on Γ can be satisfied using a penalty method (34). To do so we define a new diffusion coefficient $\mathbf{K}_\epsilon(\mathbf{x})$, $\mathbf{x} \in \Omega$ as follows:

$$\mathbf{K}_\epsilon(\mathbf{x}) = \begin{cases} \mathbf{K}(\mathbf{x}), & \text{if } \mathbf{x} \in \mathcal{B} \\ \epsilon\, \mathbf{K}(\mathbf{x}), & \text{otherwise (i.e. } \mathbf{x} \in \Omega \backslash \bar{\mathcal{B}}) \end{cases}$$

where the penalty parameter ϵ, is a small positive number. The actual boundary condition on Γ will be satisfied in the limit of $\epsilon \to 0$ (34). The original boundary conditions can be re-imposed on the extended cubic domain, Ω, for both the forward Eq. 2.1 and adjoint equation Eq. 2.2.

Chapter 3

Tumor Model and Image Registration Scaling

In the previous chapter, we discussed our inversion framework for the reaction-diffusion tumor model. To solve the forward problem, we need data from patient's anatomy with and without tumor. This data includes the segmentation (or labeling) of the brain structures such as white matter, grey matter, cerebrospinal fluid, and tumor concentration. However, in practice the data for the healthy state of the brain is typically unavailable, and neither the labeling of the tumor bearing images.

It is possible to get a manual segmentation of the tumorous state, but that is time consuming (expensive), not reproducible, and error prone. Studies have show significant variability (across experts) (117).

We address this limitation by solving a coupled problem, where we solve the tumor model in an atlas and register the results in to the patient's domain. The goal of the registration phase is to find a plausible mapping between the patient's brain and the atlas, whose properties and its segmentation is known. Using this mapping, we can transport the results from the atlas domain and compute the mismatch between the observations from the patient.

Solving this coupled problem is time consuming and prohibitive in 3D. The main

This chapter is based on *A. Gholami, A. Mang, K. Scheufele, M. Mehl, and George Biros, A Framework for Scalable Biophysics-based Image Analysis. Proceedings of ACM/IEEE SuperComputing Conference (SC'17), 2017.*

The author implemented the numerical framework of the forward and inverse tumor solver, the semi-Lagrangian and spectral operators for the registration problem, and contributed to the coupling algorithm. Andreas implemented the forward and inverse registration and Klaudius implemented the coupling algorithm and performed the benchmarking and visualizations. Prof. Biros and Prof. Mehl contributed to the algorithms and supervised the work.

bottlenecks are pseudo-spectral operators (used for evaluating differential operators) and semi-Lagrangian solvers. We have developed state-of-the-art solvers to reduce the time to solution, resulting in considerable speedup so that the problem can be solved in reasonable time.

In this chapter we will introduce the mathematical formulation for the registration problem and its coupling with the tumor inverse problem. We will then present efficiency and scalability results for the computational kernels and the whole inversion on two x86 systems, *Lonestar 5* at the Texas Advanced Computing Center and *Hazel Hen* at the Stuttgart High Performance Computing Center. We showcase results that demonstrate that our solver can be used to solve registration problems of unprecedented scale, 4096^3 resulting in ~ 200 billion unknowns—a problem size that is $64\times$ larger than the state-of-the-art. For problem sizes of clinical interest, our framework is about $8\times$ faster than the state-of-the-art.

3.1 Mathematical Formulation

In this section we will present the mathematical formulation for the two inverse problems. We refer to (15, 18, 76) for excellent surveys on theory and algorithmic developments in PDE constrained optimization.

The optimality conditions, of our problems are complex, multi-component, nonlinear operators for the state, adjoint, and control fields. The overall strategy for solving the optimization problem is the same in both cases: We employ an inexact, globalized, preconditioned Gauss–Newton–Krylov method. However, certain parts of the associated operators are different(parabolic, hyperbolic, and elliptic operators). This in turn requires the design of tailored algorithms for each part. The discretization in space and time, and algorithmic details of the respec-

tive solvers are presented in §3.2.

Data Assimilation in Brain Tumor Imaging

The formulation for this section has been discussed in detail in chapter 2. Our reaction-diffusion model for the spatio-temporal spread of cancerous cells within brain parenchyma is widely adopted in the literature (4, 28, 58, 112, 118, 169). Our formulation captures the rate of change of cancerous cells represented as a population density $c(\boldsymbol{x}, t)$ based on two phenomena: ***proliferation*** and ***net migration*** of cells.[1] The proliferation model is logistic, and the net migration of cancerous cells is modeled using an inhomogeneous (potentially anisotropic) diffusion operator (see below).

$$\partial_t c - \nabla \cdot \boldsymbol{K} \nabla c - \rho c(1-c) = 0 \qquad \text{in } \Omega_B \times (0,1], \tag{3.1a}$$

$$c - c_0 = 0 \qquad \text{in } \Omega_B \times \{0\}, \tag{3.1b}$$

This is the ***forward model***, a *non-linear* parabolic PDE, for the tumor concentration. $c \in [0,1]$ defined on the non-dimensionalized space-time interval $\Omega_B \times [0,1]$, with *Neumann boundary conditions* prescribed on $\partial\Omega_B$, where $\Omega_B \subset \Omega$, $\Omega := [0, 2\pi)^3 \subset \mathbb{R}^3$, is the spatial domain occupied by brain and $t \in [0,1]$. We remove all units from (3.1) by non-dimensionalization. We follow (58) and parameterize the initial condition

[1] We note that this model is only a crude approximation of the complex phenomena associated with cancer progression. More complicated models have appeared in the past (71) that, e.g., account for multiple cell species, chemotaxis, haptotaxis, or vascularization. Increasing the complexity of our model results in an excessive number of parameters, which have to be estimated from data or determined heuristically. This makes the calibration of our model to patient specific data much more difficult and prone to overfitting. However, increasing the complexity of our model is ultimately inevitable to be predictive.

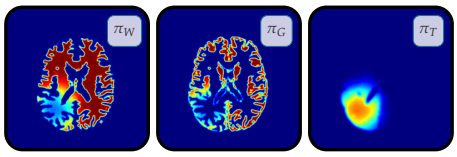


Figure 3.1: Illustration of the probability maps for different types of brain tissue. We display (from left to right) the probability maps for white matter $\pi_W(\boldsymbol{x})$, gray matter $\pi_G(\boldsymbol{x})$, and the tumor concentration $\pi_T(\boldsymbol{x})$ for an exemplary forward simulation.

as $c_0 = \Phi \boldsymbol{p}$ with the parameter vector $\boldsymbol{p} \in \mathbb{R}^{N_p}$ and Gaussian distributions $\Phi \in \mathbb{R}^{N \times N_p}$. $\boldsymbol{K}(\boldsymbol{x}) \in \mathbb{R}^{3\times3}$ is an inhomogenous, anisotropic tensor that controls the diffusion of cancerous cells c. The parameter $\rho > 0$ controls the proliferation.

In general, we do not know these parameters for an individual patient. We might be able to deduce them from additional data (e.g., genetic profiling or biopsies), or set them empirically. A more rigorous and consistent approach, is to solve an inverse parameter-identification problem (58, 81, 113). The inputs to our problem are probability maps of tissue types obtained from the patient's imaging data (62, 116), in particular white matter $\pi_W(\boldsymbol{x})$, grey matter $\pi_G(\boldsymbol{x})$, cerebrospinal fluid $\pi_C(\boldsymbol{x})$, and $\pi_T(\boldsymbol{x})$. Figure 3.1 shows an exemplary dataset *in the patient space*.[2] In many existing approaches the probability maps $\pi_W(\boldsymbol{x})$ (white matter) and $\pi_G(\boldsymbol{x})$ (grey matter) control weights that enter $\boldsymbol{K}(\boldsymbol{x})$ with the common assumption that the cell diffusivity is larger in white matter than in grey matter (28, 89, 112, 169). In our inverse tumor problem, we assume that we know $\boldsymbol{K}$ and ρ from experimental data; and we seek to find the parameters in teh initial distribution of the tumor (i.e. $\boldsymbol{p}$). In the results for this chapter we consider isotropic diffusion, for which $\boldsymbol{K}(\boldsymbol{x}) := (k_W \pi_W(\boldsymbol{x}) + k_G \pi_G(\boldsymbol{x}))\,\mathrm{diag}(1,1,1)$, parameterized by k_W and k_G. However, we our solver is capable of solving anisotropic diffusion case as discussed in chapter 2.

[2] This is a synthetic results based on a forward simulation on real brain imaging data.

Given $\pi_j(\boldsymbol{x})$, $j \in \{T, C, W, G\}$, $\boldsymbol{K}$, and ρ, *in the patient space*, our task is to find the initial tumor density $c_0 = \Phi \boldsymbol{p}$ for (3.1) that best explains the tumor cell distribution $\pi_T(\boldsymbol{x})$ observed in the imaging data at $t = 1$; i.e., we seek to minimizes the L^2-distance between model output c_1 and patient observation π_T. It is well known from inverse problem theory that the solution $\boldsymbol{p}$ is neither stable nor unique (69, 88). One remedy is to stably compute the solution to a nearby problem by augmenting our formulation with a Tikhonov regularization model. The resulting *inverse problem* for recovering $\boldsymbol{p}$ (i.e., the initial condition c_0) from π_T reads:

$$\min_{\boldsymbol{p}} \frac{1}{2} \int_{\Omega_B} (c_1 - \pi_T)^2 \, \mathrm{d}\boldsymbol{x} + \frac{\gamma}{2} \|\Phi \boldsymbol{p}\|_2^2 \, \mathrm{d}\boldsymbol{x} \tag{3.2}$$

subject to c_1 being computed by the forward model in (3.1). $\beta_p > 0$ balances regularity of $\boldsymbol{p}$ against the mismatch between c_1 and π_T.

As discussed before, the tumor inverse problem cannot be solved in the patient's domain because we typically do not have its healthy state. We address this limitation, by solving the inverse tumor problem in an atlas, and then registering the results to the patient's domain and minimize the mismatch of the registered result and the observations in the patient domain. However, a simple registration between the patient's anatomy with tumor and the atlas anatomy is not possible due to ill-defined correspondences (i.e., presence of the tumor in only one image). Therefore, we have to solve a coupled system of tumor and registration inverse problems jointly. Next we will first discuss the registration problem for two healthy brains and then in the next chapter we will discuss how we solve the coupled problem.

Diffeomorphic Image Registration

We refer to (46, 120, 159) for an introduction into the field of image registration and its applications. Image registration is a correspondence problem. The basic assumption is that there exists a geometric transformation that relates each point in one image, the so called *reference image* $m_R(\boldsymbol{x})$, to its corresponding point in another image, the so called *template image* $m_T(\boldsymbol{x})$. Given a reference image $m_R(x)$ (atlas brain) and a template image $m_T(x)$ (patient's brain), the goal of image registration is to find a diffeomorphic mapping $\boldsymbol{y(x)}$ between these two, such that $m_T(\boldsymbol{y(x)})$ is *close* to $m_R(x)$. The typical metric used is usually L^2 or L^1 distance between the two. We rewrite this as an advection problem and instead of directly finding $\boldsymbol{y(x)}$, we seek a stationary velocity $\boldsymbol{v}$ that *advects* the template image to morph into the reference image. We illustrate this in Figure 3.2. We introduce a pseudo-time variable $t \in [0, 1]$ and model this geometric transformation based on a transport equation for the intensity values of m_T. The *forward model* of our problem is: Given a *stationary velocity field* $\boldsymbol{v}(\boldsymbol{x})$ and a template image $m_T(\boldsymbol{x})$ compute the transported intensities $m_1(\boldsymbol{x}) := m(\boldsymbol{x}, t = 1)$ at $t = 1$ by solving

$$\partial_t m + \boldsymbol{v} \cdot \nabla m = 0 \qquad \text{in } \Omega \times (0, 1], \tag{3.3a}$$

$$m = m_T \qquad \text{in } \Omega \times \{0\}, \tag{3.3b}$$

with periodic boundary conditions on $\partial\Omega$ forward in time.

We again formulate the ***inverse problem*** as a PDE constrained optimization problem. The task is to find a *plausible* velocity field $\boldsymbol{v}(\boldsymbol{x})$ so that the transported intensities of m_T at $t = 1$, i.e., the solution $m_1(\boldsymbol{x}) := m(\boldsymbol{x}, t = 1)$ of (3.3) is *similar* to

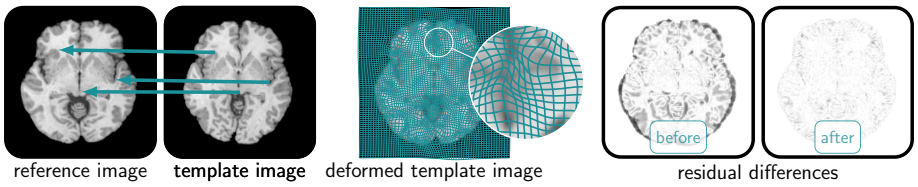


Figure 3.2: Image registration. The inputs to this inverse problem are scalar intensity values of two images of the same object (left). The inherent assumption is that there exists a geometric transformation that relates points in the reference image m_R to its corresponding points in the template image m_T (green arrows). We model this geometric transformation based on the transport equation (3.3). The deformed template image is illustrated in the middle. The residual between the images before and after registration is shown on the right. The images depicted here are a 2D illustration (axial slice) of the 3D image registration problem solved in §3.4.

$m_R(\boldsymbol{x})$ for all $\boldsymbol{x}$. Related formulations can, e.g., be found in (12, 17, 25, 70, 96, 108, 111, 181). We search for a minimizer

$$\min_{\boldsymbol{v}} \frac{1}{2} \int_\Omega (m_1 - m_R)^2 \, \mathrm{d}\boldsymbol{x} + \frac{\beta_v}{2} \int_\Omega \nabla \boldsymbol{v} : \nabla \boldsymbol{v} \, \mathrm{d}\boldsymbol{x} \tag{3.4}$$

subject to the forward model in (3.3). The second term in Eq. 3.4 enforces smoothness for $\boldsymbol{v}$ with the ***regularization parameter*** $\beta > 0$. To be able to guarantee the existence and uniqueness of a solution of both, the forward and the optimization problem, one has to impose appropriate smoothness requirements on the images and the velocity.[3] A key requirement in medical imaging is that the solution of (3.3) does not introduce any foldings, i.e., a volume element does not collapse to a single point, and characteristics traced by $\boldsymbol{v}$ do not cross. This is ensured by the regularization operator in Eq. 3.4. In this work, we additionally control the volume change by introducing a penalty on the divergence of $\boldsymbol{v}$, i.e., we add the constraint $\nabla \cdot \boldsymbol{v} = w$ and penalize variations in w by introducing an addition regularization norm with regularization weight $\beta_w > 0$ (109). Setting w to zero yields an incompressible diffeomorphism (25, 108, 111).

[3] We refer to (11, 12, 25) for a theoretical discussion.

3.2 Algorithms

In this section, we discuss the discretization in space and time, the solvers, the computational kernels, and the parallel implementation of the optimization problems in §3.1. We use a globalized, preconditioned, inexact, reduced space Gauss–Newton–Krylov method for both problems. Based on the derivation of the optimality conditions, we describe the individual numerical building blocks of our scheme.

Optimality Conditions

We use the method of Lagrange multipliers (102) in an *optimize-then-discretize* approach. That is, we first compute variations of the Lagrangian functional with respect to the state, adjoint, and control variables, and then discretize them. We will see that the resulting equations are complex multi-physics operators that are challenging to solve in an efficient way.

Tumor. The Langrangian for the tumor reads

$$\begin{aligned}\mathcal{L}_T = \frac{1}{2}\int_{\Omega_B}(c_1 - \pi_T)^2\,\mathrm{d}\boldsymbol{x} + \frac{\gamma}{2}\|\Phi\boldsymbol{p}\|_2^2 d\boldsymbol{x} + \int_{\Omega_B}\alpha_0(c_0 - \Phi p)\,\mathrm{d}\boldsymbol{x} \\ + \int_0^1\int_{\Omega_B}\alpha\left(\partial_t c - \nabla\cdot\boldsymbol{K}\nabla c - \rho c(1-c)\right)\mathrm{d}\boldsymbol{x}\,\mathrm{d}t\end{aligned} \tag{3.5}$$

with the Langrange multiplier function α, and $\alpha_0(\boldsymbol{x}) = \alpha(\boldsymbol{x}, t = 0)$. We invert for the parametrization $\boldsymbol{p}$ for c_0 in Eq. 3.1a. The ***gradient*** of $\mathcal{L}_T$ in terms of $\boldsymbol{p}$ is given by

$$\boldsymbol{g}_{\boldsymbol{p}} := \gamma\Phi^T\Phi\boldsymbol{p} - \Phi^{\mathsf{T}}\alpha_0 \tag{3.6}$$

This defines the non-linear problem we have to solve for $\boldsymbol{p}$. To obtain α_0, we have

to solve the ***adjoint equation*** stemming from the gradient of $\mathcal{L}_T$ in terms of c:

$$-\partial_t \alpha - \nabla \cdot \boldsymbol{K} \nabla \alpha - \alpha\rho(1-2c) = 0 \qquad \text{in } \Omega_B \times [0,1), \tag{3.7a}$$

$$\alpha - (\pi_T - c) = 0 \qquad \text{in } \Omega_B \times \{1\}, \tag{3.7b}$$

with Neumann boundary conditions on $\partial\Omega_B$, backward in time. Note that the final condition in Eq. 3.7 at $t = 1$ depends on c, which is the solution of the forward problem Eq. 3.1.

Registration. The Lagrangian function of the image registration problem reads[4]

$$\begin{aligned}\mathcal{L}_R = &\frac{1}{2}\int_\Omega (m_1 - m_R)^2 \,\mathrm{d}\boldsymbol{x} + \frac{\beta_v}{2}\int_\Omega \nabla \boldsymbol{v} : \nabla \boldsymbol{v} \,\mathrm{d}\boldsymbol{x} \\ &+ \int_0^1 \int_\Omega \lambda \left(\partial_t m + \boldsymbol{v} \cdot \nabla m\right) \mathrm{d}\boldsymbol{x}\,\mathrm{d}t + \int_\Omega \lambda_0 \left(m_0 - m_T\right) \mathrm{d}\boldsymbol{x}\end{aligned} \tag{3.8}$$

with the Lagrangian multiplier function λ and $\lambda_0(\boldsymbol{x}) = \lambda(\boldsymbol{x}, t = 0)$. We invert for the transformation velocity $\boldsymbol{v}$. The ***gradient*** of $\mathcal{L}_R$ in terms of $\boldsymbol{v}$ is given by

$$\boldsymbol{g}_{\boldsymbol{v}} := -\beta_v \Delta \boldsymbol{v} + \mathcal{K}[\int_0^1 \lambda \nabla m]\,\mathrm{d}t, \tag{3.9}$$

where $\mathcal{K}$ is a pseudo-differential operator that enforces additional constraints on the divergence of $\boldsymbol{v}$ (see (109); for Eq. 3.8 $\mathcal{K}$ is simply an identity operator). To evaluate $\boldsymbol{g}_{\boldsymbol{v}}$ in Eq. 3.9 for a candidate velocity field $\boldsymbol{v}$, we need to find the *state* and *adjoint* variable $m(\boldsymbol{x}, t)$ and $\lambda(\boldsymbol{x}, t)$, respectively. We obtain them from setting the

[4] We neglect the incompressibility constraint for clarity

variations of $\mathcal{L}_R$ in terms of λ and m to zero, i.e., by solving Eq. 3.3a forward in time and using m at $t = 1$ as an initial condition for the ***adjoint equation***:

$$-\partial_t \lambda - \nabla \cdot \lambda \boldsymbol{v} = 0 \qquad \text{in } \Omega \times [0,1), \tag{3.10a}$$

$$\lambda - (m_R - m_1) = 0 \qquad \text{in } \Omega \times \{1\}. \tag{3.10b}$$

with periodic boundary conditions on $\partial\Omega$, backward in time. Having found $\lambda(\boldsymbol{x}, t)$, we can evaluate Eq. 3.9. In our actual implementation, we do not store λ. We integrate the second term in Eq. 3.9 directly when solving (3.10), instead. This allows us to reduce the memory footprint significantly. We only have to store the transported intensities m (to be able to evaluate the Hessian matvec).

The registration Hessian matvec given a velocity $\tilde{v}$, can be computed by first solving the incremental state equation:

$$\partial_t \tilde{m} + \nabla \tilde{m} \cdot v + \nabla m \cdot \tilde{v} = 0 \qquad \text{in } \Omega \times (0,1], \tag{3.11}$$

$$\tilde{m} = 0 \qquad \text{in } \Omega \times \{0\}, \tag{3.12}$$

$$-\partial_t \tilde{\lambda} - \nabla \cdot (v\tilde{\lambda} + \tilde{v}\lambda) = 0 \qquad \text{in } \Omega \times [0,1), \tag{3.13}$$

$$\tilde{\lambda} + \tilde{m}_1 = 0 \qquad \text{in } \Omega \times \{1\}, \tag{3.14}$$

$$\mathcal{H}(v)\tilde{v} = \beta\Delta^2\tilde{v} + \mathcal{K}[\int_0^1 \tilde{\lambda}\nabla m + \lambda\nabla\tilde{m}\,\mathrm{d}t\,] \qquad \text{in } \Omega, \tag{3.15}$$

Spatial Discretization

We use regular grids to discretize the space-time interval $\Omega \times [0,1]$, $\Omega := [0,2\pi)^3 \subset \mathbb{R}^3$. The spatial grid consists of $N_0 \times N_1 \times N_2$, $N_i \in \mathbb{N}$ grid points $\mathbf{x}_{\boldsymbol{i}} := (x_{0,\boldsymbol{i}}, x_{1,\boldsymbol{i}}, x_{2,\boldsymbol{i}}) \in \mathbb{R}^3$, $x_{j,\boldsymbol{i}} := 2\pi i_j / N_j$, $0 \le i_j \le N_j - 1$, $j = 0,1,2$. We follow (111) and use a spectral projection scheme for all spatial operations. That is, we approximate spatial functions u as

$$u(\boldsymbol{x}) = \sum_{\boldsymbol{k}} \hat{u}_{\boldsymbol{k}} \exp(-\boldsymbol{k} \cdot \boldsymbol{x}), \tag{3.16}$$

where $\boldsymbol{k} = (k_0, k_2, k_2) \in \mathbb{N}^3$, $-N_j/2 + 1 \le k_j \le N_j/2$, $j = 0,1,2$, is the grid index and $\hat{u}_{\boldsymbol{k}}$ are the spectral coefficients of u. The mappings between the spatial and spectral coefficients are done using FFTs. We approximate derivative operators by applying the appropriate weights in the spectral domain; i.e., spatial differentiations involve a forward FFT, modification of weights in the spectral representation Eq. 3.16, and subsequently an inverse FFTs. Given that we use a spectral collocation scheme, we assume that the functions in our formulation (including images) are periodic and continuously differentiable. We apply appropriate filtering operations and periodically extend or mollify the discrete data to meet these requirements.

The tumor problem in (3.1) requires Neumann boundary conditions on the surface of the brain $\partial\Omega_B$. We follow (58, 80) and use a penalty method to approximate these boundary conditions. We apply periodic boundary conditions on $\partial\Omega$ and set the diffusion coefficient outside of Ω_B equal to a small penalty parameter $\boldsymbol{K}^{\epsilon}$.

Numerical Time Integration

Fulfilling the first order optimality conditions, i.e., zero gradients of the Lagrange formulation, requires a repeated solution of parabolic or hyperbolic PDEs. In the following, we sketch our time integration schemes for these systems.

Parabolic PDEs. We follow (58, 80) and use a second-order Strang-splitting method to solve the parabolic equations (3.1) and (3.7). We explain this for the forward problem (3.1). Let $\mathbf{c}^j \in \mathbb{R}^N$ denote the tumor distribution at time $t^j = j\Delta t$, $\Delta t = 1/n_t$. We apply an implicit Crank–Nicolson method for diffusion and solve the reaction part analytically:

$$(\mathbf{I} - 0.25\Delta t\mathbf{D})\mathbf{c}^\dagger = (\mathbf{I} + 0.25\Delta t\mathbf{D})\mathbf{c}^j \tag{3.17}$$

$$\partial_t \mathbf{c} = \rho\mathbf{c}(1-\mathbf{c}), \quad \mathbf{c}(t^j) = \mathbf{c}^\dagger \qquad \text{in } (t^j, t^{j+1}) \tag{3.18}$$

$$(\mathbf{I} - 0.25\Delta t\mathbf{D})\mathbf{c}^{j+1} = (\mathbf{I} + 0.25\Delta t\mathbf{D})\mathbf{c}(t^{j+1}). \tag{3.19}$$

We use a PCG method with a fixed tolerance of $1e{-}6$ to solve Eq. 3.17 and Eq. 3.19. Equation 3.18 is solved analytically. The preconditioner is based on a constant coefficient approximation of $\mathbf{D}$ given by $\tilde{\mathbf{D}} = \mathbf{I} - 0.25\Delta t\tilde{K}\triangle$, where $\tilde{K} > 0$ is the average diffusion coefficient. The inversion and construction of this preconditioner has negligible computational cost due to our spectral scheme; it only requires one Hadamard product in the Fourier domain and one forward and backward FFT. This preconditioner can be shown to result in a mesh independent condition number as long as the mesh is fine enough to resolve the diffusion coefficient. Since, for medical images, we can have large contrast and sharp transitions,

the grid to resolve the operator can become prohibitively large; a lack in resolution manifests in an increase in the number of Krylov iterations, as we increase the mesh size.

Hyperbolic PDEs (Transport Equations). We employ a semi-Lagrangian scheme (42) to solve the hyperbolic transport equations. The use of semi-Lagrangian schemes in the context of image registration is not new; see (12, 25, 110, 111) for details. Semi-Lagrangian schemes are unconditionally stable, which allows us to keep the number of time-steps small[5]. This is critical for large-scale 3D applications.

The semi-Lagrangian algorithm is as follows. Consider the following equation:

$$\partial_t m + \boldsymbol{v} \cdot \nabla m = f(m, x), \tag{3.20}$$

To solve for one time step we have to first derive the characteristic equation. Using the change of variables $\xi = x - vt$ and $\rightarrow m(x,t) = \hat{m}(\xi, \tau)$, we have:

$$\partial_t \hat{m}(\xi,t) - \boldsymbol{v} \cdot \partial_\xi \hat{m}(\xi,t) + \boldsymbol{v} \cdot \partial_\xi \hat{m}(\xi,t) = f(\hat{m}, \xi, t) \tag{3.21}$$

$$\partial_t \hat{m}(\xi,t)|_{\xi=\text{const}} = f(\hat{m}, \xi + \boldsymbol{v}t) \tag{3.22}$$

The condition of ξ = const gives us the characteristics equation:

$$\frac{\partial x}{\partial t} = \boldsymbol{v}(x), \tag{3.23}$$

[5]Given that a spectrally accurate interpolation method is used.

To solve Eq. 3.22 we need to compute x^n, which is commonly referred to as the *departure point*. We use RK2 scheme as follows:

$$x^\star = x^{n+1} - \mathrm{d}t\boldsymbol{v}(x), \tag{3.24}$$

$$x^n = x - \frac{\mathrm{d}t}{2}(\boldsymbol{v}(x) + \boldsymbol{v}(x^\star)) \tag{3.25}$$

After computing x^n from Eq. 3.25, we use Heun's method to solve Eq. 3.22. For brevity of notation we set $t^{n+1} = 0$ and $t^n = -\mathrm{d}t$:

$$\bar{m} = m^n(x^n, -\mathrm{d}t) + \mathrm{d}t f(m^n, x^n, -\mathrm{d}t) \tag{3.26}$$

$$m^{n+1}(x^{n+1}, 0) = m^n(x^n, -\mathrm{d}t) + \frac{\mathrm{d}t}{2}(f(m^n, x^n, -\mathrm{d}t) + f(\bar{m}, x^{n+1}, 0)) \tag{3.27}$$

In summary to compute m^{n+1} we need to follow these steps:

1. Compute $x^\star$ and evaluate $\boldsymbol{v}(x^\star)$ by interpolation
2. Compute the departure point x^n
3. Evaluate m^n by interpolation
4. Compute the intermediate point in Heun's method $\bar{m}$
5. Compute m^{n+1}

The interpolation phase is the major computational bottleneck in this algorithm. Note that, since we invert for a stationary velocity field $\boldsymbol{v}(\boldsymbol{x})$, Eq. 3.25 needs to be solved only once during each Newton step. Therefore the interpolation for

computing $\boldsymbol{v}(x^\star)$ can be done only once. But the interpolation for $\bar{m}$ needs to be performed for each time step.

Inversion

We use a matrix-free, globalized, inexact Gauss–Newton algorithm for numerical optimization. The *update rule* for a control variable $\mathbf{u}_k$ at iteration k reads

$$\mathbf{u}_{k+1} = \mathbf{u}_k - \alpha_k \mathbf{H}^{-1}\mathbf{g}, \qquad \alpha_k > 0, \tag{3.28}$$

where $\mathbf{H}^{-1}$ is a Gauss–Newton approximation of the inverse of the Hessian[6], $\mathbf{g}$ a discrete representation of the gradient of the optimization problem. For the tumor case, the control variable $\mathbf{u}_k$ is given by $\boldsymbol{p} \in \mathbb{R}^{N_p}$ and for the registration by $\mathbf{v} \in \mathbb{R}^{3N}$ with $N = N_0 N_1 N_2$. The step length $\alpha_k > 0$ is determined by an Armijo line search and is chosen such that we decrease the value of the objective function in every iteration. Storing and inverting $\mathbf{H}$ using a direct solver is prohibitively expensive. We use an iterative PCG method instead, which only requires an expression for the action of the Hessian on a vector (***Hessian matvec***). We further reduce the computational cost by inverting $\mathbf{H}$ only inexactly (39, 127). Our solver uses PETSc's TAO module (8). Our code implements the operations for evaluating the objective functions, the gradient, and the Hessian matvec.

Despite the complexity of the whole formulation, it turns out the main computational bottlenecks are FFTs and interpolations. We will see in §3.4 that these computations make up 80% to 90% of the entire time spent in the solver. We will describe the parallel implementation of these kernels and present dedicated strate-

[6]We can derive the Hessian from the second variations of the Lagrangian functional of our problem. We omit these details and refer to related work (6, 58, 108, 109, 111).

gies to significantly speed up their single core performance next.

3.3 Parallel Algorithms and Computational Kernels

We use a 2D pencil decomposition for 3D FFTs (33, 63) to distribute the data among processors. We denote the number of MPI tasks by $P = P_0 P_1$. Each MPI task gets a total of $N_0/P_0 \times N_1/P_1 \times N_2$ values. We denote the number of grid points owned by a single MPI task by n_{local}. We use our in-house open source library AccFFT (56, 57), which supports parallel FFT for CPU/GPU in both single and double precision. For parallel linear algebra operations we use PETSC as well as it's optimization interface TAO (8).

The main computational kernels of SIBIA are the FFT, used for spatial differential operators in our spectral approach, and the interpolations used in the semi-Lagrangian scheme for advection. Below, we discuss specific performance optimizations for each of these two.

FFT for Spectral Operators

Spatial differential operators such as gradient or divergence can be computed with spectrally by first transforming the input field into the frequency domain (by FFT), followed by a Hadamard product and an inverse FFT. Exemplarily, we consider computing the x-derivative of a scalar field f:

$$f_x = \mathcal{F}_x^{-1}(-i\omega_x \mathcal{F}_x(f)), \tag{3.29}$$

where $\mathcal{F}_x$ denotes the FFT transform in x direction. Note that, to compute the x-derivative, we only need a batched 1D FFT instead of a 3D FFT. This saves

a large amount of communication since no repartitioning of data for the pencil decompositions in y direction is required. However, we still have to perform a global transpose to establish the pencil decomposition in x direction. This is followed by a forward FFT transform in x-direction, a Hadamard product, a local inverse FFT, and another global transpose to redistribute the data. Parallel FFT libraries such as P3DFFT (135) allow the user to specify which directions the local forward/inverse FFT is needed but it is currently not possible to avoid unnecessary global transposes in y direction (or x when computing y derivative). We have enhanced AccFFT such that it avoids all unnecessary global transposes, which for the x-derivative reduces 4 global transposes to just 2 (see chapter 6 for details). This new implementation reduces the total global transposes to 4 for the gradient/divergence operators as opposed to 8. This reduces the communication volume by a factor of two.

Interpolation

As explained in §3.2, the transport equations in the registration are solved using Semi-Lagrangian (SL) scheme that requires costly interpolation of velocities and image data along backward characteristics. Compared to Runge-Kutta methods, the main advantage of SL is its unconditional stability. However, this comes at the price of costly off grid interpolations. We first discuss single node optimization and then explain our distributed method.

Single-Node Optimization

We use Lagrange polynomials for interpolation. The value of a scalar (or vector) field f, at an off-grid point (x, y, z) (departure coordinate) can be computed

as:

$$f(x,y,z) = \sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{d} \ell_i(x)\ell_j(y)\ell_k(z) f_{ijk}, \tag{3.30}$$

$$l_i(x) = \prod_{n=0;i\neq n}^{d} \frac{x - x_n}{x_i - x_n}, \tag{3.31}$$

where f_i is the function value at grid point i, ℓ_i is the i^{th} Lagrange basis polynomial, and d is the interpolation order (which is cubic in our algorithm). Using a higher order method would provide more opportunities for vectorization and would generally lead to better hardware performance, but in practical applications with real data, the use of higher order interpolation does not improve the registration quality. Therefore, we focus on optimizing cubic order interpolation, which requires the computation of 12 Lagrange basis polynomials ($\ell_i(x)$) and the evaluation of Eq. 3.30 for each departure point. Evaluation of the basis polynomial requires reading three single-precision departure coordinates from the memory, and the interpolation kernel (Eq. 3.30) requires 64 non-aligned memory loads [7]. In total we have 1 memory write operation (for storing the result), 68 memory reads and 239 floating point operations to compute the interpolation for each departure point. In total there are N departure points, same as the grid size. Note that each departure point requires a different set of 64 f_{ijk} grid values, which creates a significant cache misses that cannot be hidden by the few floating point computations in Eq. 3.30. We address this by using a novel binning method: There is no reason to process the departure points based on the order that they were received. Instead, we process departure points in groups such that we can reuse the data already loaded

[7] Notice that the non-aligned memory part is not due to non-aligned allocation. The stencil of function points, f_{ijk} required for the interpolation is determined by the departure point coordinates, which can start from a non-aligned memory address.

into the cache for processing the previous points. This means that parts of the data required for the grid values can be fetched from the L1 or L2 cache instead of the main memory. One can achieve this by using a space filling sort on the departure coordinates such as Morton sorting. However, sorting is itself a memory bound problem and imposes an additional overhead. Instead of sorting the departure points themselves, we partition the domain into patches of size $16 \times 16 \times 16$ grid cells. Then we use Morton sorting on the bin ids to determine which bin has to be processed first. This phase can actually be performed offline analytically, since it does not depend on the departure point coordinates (only depends on the grid and bin size).

Table 3.1: Experiment to evaluate the efficiency of the binning method in the interpolation. We report the GFLOPS for the interpolation kernel for the two experiments in Fig. 3.3. We report results with/without binning and SIMD. Adding binning method along with results in significant speedup. For large problem sizes, the main speedup is achieved by the binning. The experiment was performed on 1 Haswell node with 24 MPI tasks. The reported GFlops include the time needed to reshuffle the interpolated values in the binning method.

	Fixed Region	Random Departure Points			
N		No Bin, No SIMD	No Bin + SIMD	Bin + SIMD	Speedup
64^3	356	**45**	280	**295**	**6.56 x**
128^3	347	**42**	146	**256**	**6.1 x**
256^3	287	**28**	54	**173**	**6.18 x**
512^3	287	**13**	19	**112**	**8.62 x**

Vectorized Implementation:

We use AVX2-256 (Advanced Vector Extensions) to execute Eq. 3.30 using SIMD as follows:

$$f(x,y,z) = \sum_{k=0}^{3} \ell_k(z) \sum_{i=0}^{3} \ell_i(x) \sum_{j=0}^{3} \ell_j(y) f_{ijk}, \tag{3.32}$$

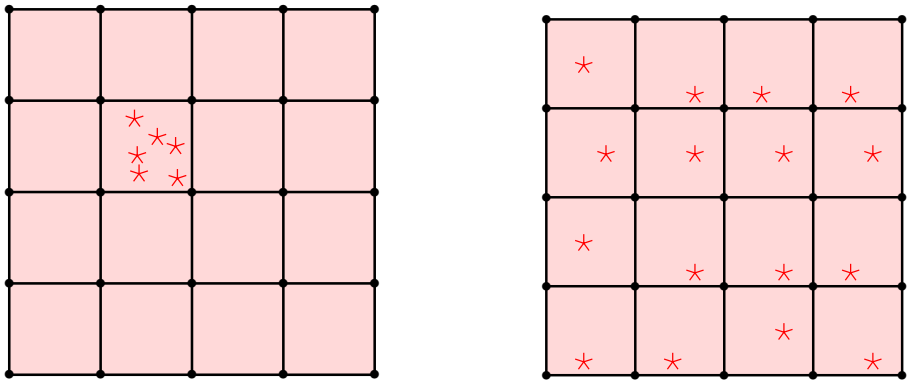


Figure 3.3: (left) Fixed region departure points experiment; (right) random departure points experiment performed in Table 3.1. The red stars show the departure coordinates, where interpolation needs to be performed.

With AVX2 on Haswell, one can perform two Fused Multiply-Add operations on 8 single precision data per CPU cycle. To take advantage of this, we load contiguous unaligned 128 bit chunks of f_{ijk} into AVX registers. Note that the data is stored in C-major order, meaning that the kth index runs fastest in the memory and the i and j indices require strided access. Therefore, to fill up each AVX register, we need two load instructions. After loading the data, we compute the $\sum_j \ell_j$ sum using two FMA operations for $i = 0, 1, 2, 3$. The 4 resulting AVX vectors are then multiplied by the corresponding ℓ_i and the results are added. The final result is then multiplied by ℓ_k, followed by a horizontal reduce-add operation. These dependencies besides the memory load are the bottlenecks of the kernel. When possible we try to avoid ADD operations as on Haswell architecture this operation is executed through one of its ports (where as a MULT or FMA can be done through both ports). The interpolation kernel has a lot of dependencies which makes it hard to keep the execution units busy.

To test the effectiveness of our method, we perform two experiments. First we restrict all of the departure points to be in a fixed region in space so that we only have

to issue one load for the 64 neighboring grid values (f_{ijk}) for all of the interpolations. In the second experiment, we test a realistic case where the departure points fall into random regions in space as shown in Fig. 3.3. The fixed region experiment is an upper bound for the performance of our kernel. In the random test, different grid values have to be loaded per departure point. The fixed region case achieves consistently high GFLOPS for different problem sizes. The drop in performance is because we have to load data from lower levels of the cache as the problem size increases. However, the performance drops significantly in the random region experiment. Using the AVX implementation only increases the performance for the case where the data fits into the cache. For larger problem sizes the cache misses become dominant and the performance drops significantly. However, when we combine the SIMD implementation with the binning approach this overhead gets reduced significantly, resulting in $6-8\times$ speedup.

Distributed Memory Semi-Lagrangian

The numerical algorithm for the semi-Lagrangian is the same in parallel: we have to first compute the departure coordinates (x^n in Eq. 3.23) and then perform interpolation to compute the new time point data. In a distributed setting, three cases are possible as shown in Fig. 3.4: (i) the departure coordinate would fall into the locally owned domain of the process, (ii) x^n would fall in between the domain of each process, and (iii) x^n would fall into another processor's domain. For the first case, no change in the single-node algorithm is needed. In the second case, we need to first perform a ghost cell communication (also referred to as halo exchange) to receive the neighboring elements before we can perform the interpolation. For corner cases, we have to request neighboring elements from processes on the opposite

sides of the domain due to periodic boundary conditions.

The third case is more involved. Each MPI process has to first compute the departure point and determine which MPI process's domain it belongs to. After all of the processes have computed this, then a (scatter) alltoallv is performed where each process sends its own departure point coordinates and receives other processes' departure points as well. We refer to this as the ***Scatter*** phase. Each MPI task would then perform the interpolations on all of the case (i-ii) points as well as the points it received from other tasks (***Execution*** phase). Once completed, a final (scatter) alltoallv operation needs to be performed to gather case (iii) interpolation results. We refer to this step as ***Gather*** phase.

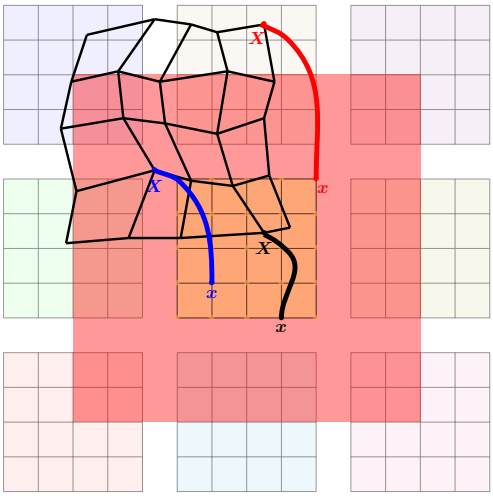


Figure 3.4: The characteristics line is shown for the three cases that may occur in parallel Semi-Lagrangian: (i) the departure coordinate (shown with X) would fall into the locally owned domain of the process (black line), (ii) X would fall in between the domain of each process (blue line), and (iii) X would fall into another processor's domain (red line). Each square is representative of the locally owned domain by each MPI process.

Notice that the scatter and gather phase involve alltoall communication which is costly and adversely affects scalability. In a naive implementation we need to perform all the three phases (i.e. Scatter, Execution, and Gather) for each time step. However, since the velocity field is stationary, we need to only scatter

the off-grid departure coordinates once for the entire advection solve. We can further reduce this to one complete Newton iteration. The only time that the scatter phase will be required would be in the gradient computation. This would reduce the communication overhead significantly, and will completely mask the overhead of the binning method.

For each gradient evaluation, one has to first solve the forward problem Eq. 3.3 and the adjoint problem Eq. 3.10a. Using the RK2 scheme for each equation requires one intermediate and one final departure point interpolation. Thus we would need four scatter operations to compute the gradient, and zero for each subsequent Hessian matvec. The number of Execution and Gather phases required to compute a Hessian matvec and compute the gradient is given in Table 3.2.

Table 3.2: The complexity for the number of Execution/Gather phases in the interpolation required for forward, adjoint, incremental forward and incremental adjoint is given in terms of the number of the number of time steps, N_t. For the incremental forward equation (3.11), it is possible to reduce the complexity by first computing the dot product of $\tilde{v} \cdot \nabla \boldsymbol{m}$ and then perform the interpolation on the scalar result. The corresponding complexity for that approach is given in the parenthesis.

Phase	Execution/Gather
intermediate velocity in Eq. 3.3	3
forward Eq. 3.3	N_t
intermediate velocity in Eq. 3.10a	3
intermediate div term in Eq. 3.10a	1
adjoint Eq. 3.10a	N_t
Gradient Evaluation	$2N_t + 7$
source term in Eq. 3.11	$3N_t + 3$ / (N_t)
incr forward Eq. 3.11	$4N_t + 3$ / $(2N_t)$
div term in Eq. 3.13	1
incr adjoint Eq. 3.13	$N_t + 1$
Hessian Matvec	$5N_t + 4$ / $(3N_t + 1)$

In summary, our optimizations for the interpolation kernel include a novel approach to reduce cache misses by using a binning method, AVX vectorization of

the kernel, and a novel distributed memory semi-Lagrangian algorithm. Our kernel has support for GPU and KNL in addition to CPUs[8]. Overall our optimizations has resulted in a significantly better interpolation kernel than the one presented in our earlier work (109). We are now mainly bound by the communication time instead of the time spent in the execution phase of the kernel.

3.4 Results

We report results for the tumor inversion and for the registration. We report the overall runtime,[9] execution and communication times for the computational kernels, and the total time spent to evaluate these kernels.[10] We always report the maximum time across all MPI tasks. We will also discuss the ***algorithmic scalability*** of the tumor diffusion solver, i.e., the number of PCG iterations as we increase the problem size. We also discuss parallel scalability of the different kernels, and report the overall scalability.

Setup, Implementation, and Hardware

We execute runs on *Lonestar 5* (2-socket Xeon E5-2690 v3 (Haswell) with 12 cores/socket, 64 GB memory per node) and *Hazel Hen* (same node as Lonestar but with 128 GB memory per node). Our code is written in C++ and uses MPI for parallelism. It is compiled with the default Intel compilers available on the systems (Intel 16). We use PETSc's implementations for linear algebra operations, PETC's TAO package for the nonlinear optimization (8), AccFFT for Fourier

[8] Although the execution phase on those architectures have not been optimized yet.
[9] The runtime is the time spent on the entire inversion (excluding setup and I/O times).
[10] The overall kernel evaluation time includes execution, communication, and shuffling of the data.

transforms (56, 58), and PnetCDF for I/O (98).

Data assimilation in brain tumor imaging

In this section, we study the computational performance and parallel scalability of the tumor inversion solver for real brain data as illustrated in Figure 1.1. We use resampled medical brain image data with multiple spatial resolutions $N \in \{64^3, 128^3, 256^3, 512^3\}$ and present scalability results for up to $16,384$ MPI tasks on *Hazel Hen* system.

Setup. For all tumor inversion experiments, we set the regularization parameter $\beta_p = \text{E-3}$, the number of time steps $n_t = 4$ and use double precision. We perform three Gauß-Newton iterations for the inversion and limit the number of Hessian matvecs in the PCG solver to three. This setup results in an overall gradient reduction by one order of magnitude and 13% relative mismatch for the reconstructed tumor. Using $n_t = 4$ is not sufficient to resolve the dynamics accurately but it puts pressure on the forward solver and allows us to monitor its algorithmic scalability. We report the time to solution as well as the percentage and absolute time spent in the Hessian matvec, the PCG solver and the FFT, which is the main computational kernel in the tumor inversion. For all the strong scaling runs, we use 12 MPI tasks per node.

Strong Scaling. We report scaling results for different spatial resolutions $N \in \{64^3, 128^3, 256^3, 512^3\}$ and different numbers of MPI tasks $P \in \{2^1, \ldots, 2^{14}\}$. The results are reported in Table 3.3. For $N = 256^3$, we observe excellent strong scaling performance with a parallel efficiency of $98\,\%$, going from 32 to 2048 MPI

Table 3.3: Computational performance of the tumor solver for real brain data, illustrated in Fig. 1.1 on HLRS's *Hazel Hen*. We perform three Gauß-Newton iterations and limit the number of Hessian matvecs in the PCG solver to a maximum of three. We report time to solution, time spent in the Hessian matvec, in the FFT, and in the diffusion solver, respectively (in seconds). We show the strong scaling efficiency eff_S and the algorithmic weak scaling efficiency eff_W. For the parallel weak scaling efficiency $\widetilde{\text{eff}}_W$, the number of PCG iterations for the diffusion solver is fixed for all spatial resolutions. Timings are reported as a function of the number of unknowns (in space), and the number of nodes and tasks. For runs marked with † we encountered MPI problems due to non-optimal settings. Performance could be improved by increasing the MPI buffer and the maximum message size for the MPI eager protocol (run marked ‡).

	N	nodes	tasks	runtime	eff_S	eff_W	$(\widetilde{\text{eff}}_W)$	FFT	([%])	H-matvec	diffusion	([%])
#1	64^3	1	2	4.07e+1	100.0	100.0	(100.0)	3.23e+1	(79.5)	2.57e+1	3.98e+1	(97.8)
#2		1	4	2.60e+1	78.2	100.0	(100.0)	2.12e+1	(81.7)	1.64e+1	2.55e+1	(98.1)
#3		1	8	1.33e+1	76.6	100.0	(100.0)	9.92	(74.7)	8.38	1.30e+1	(97.9)
#4		2	16	6.05	84.0	100.0	(100.0)	5.05	(83.5)	3.82	5.89	(97.4)
#5		3	32	4.81	52.8	100.0	(100.0)	4.14	(86.1)	3.02	4.68	(97.3)
#6	128^3	1	4	4.91e+2	100.0			4.24e+2	(86.3)	3.14e+2	4.87e+2	(99.2)
#7		1	8	3.07e+2	80.0			2.56e+2	(83.2)	1.96e+2	3.04e+2	(99.1)
#8		2	16	1.37e+2	89.6	29.7	(43.7)	1.12e+2	(81.5)	8.77e+1	1.36e+2	(99.2)
#9		2	32	8.46e+1	72.6	30.7	(45.2)	6.72e+1	(79.5)	5.41e+1	8.38e+1	(99.1)
#10		6	64	4.15e+1	73.9	32.0	(46.8)	3.24e+1	(78.1)	2.65e+1	4.11e+1	(99.0)
#11		11	128	2.08e+1	73.7	29.0	(42.0)	1.71e+1	(82.2)	1.33e+1	2.06e+1	(98.7)
#12		22	256	1.13e+1	67.7	42.5	(60.9)	9.49	(83.7)	7.20	1.11e+1	(98.0)
#13	256^3	3	32	1.58e+3	100.0			1.31e+3	(82.9)	1.02e+3	1.57e+3	(99.1)
#14		6	64	8.06e+2	98.2			6.75e+2	(83.7)	5.19e+2	7.99e+2	(99.1)
#15		11	128	3.63e+2	109.0	11.2	(22.9)	3.01e+2	(82.9)	2.35e+2	3.61e+2	(99.5)
#16		22	256	1.81e+2	109.6	14.4	(29.3)	1.49e+2	(82.6)	1.16e+2	1.80e+2	(99.5)
#17		43	512	1.00e+2	98.9	13.3	(26.6)	8.26e+1	(82.5)	6.45e+1	9.95e+1	(99.3)
#18		86	1024	4.03e+1	123.0	15.0	(31.0)	3.45e+1	(85.7)	2.55e+1	3.97e+1	(98.6)
#19		172	2048	2.50e+1	98.9	19.2	(38.5)	2.17e+1	(86.9)	1.59e+1	2.38e+1	(95.0)
#20	512^3	22	256	2.52e+3	100.0			2.15e+3	(85.2)	1.65e+3	2.50e+3	(99.4)
#21		43	512	1.32e+3	95.8			1.13e+3	(86.2)	8.61e+2	1.31e+3	(99.4)
#22		86	1024	7.54e+2	83.5	5.4	(13.6)	6.24e+2	(82.8)	4.93e+2	7.52e+2	(99.7)
#23		172	2048	3.47e+2	90.7	7.5	(19.0)	2.87e+2	(82.8)	2.32e+2	3.45e+2	(99.4)
#24		342	4096	2.57e+2	61.2	5.2	(11.6)	1.98e+2	(77.1)	1.66e+2	2.50e+2	(97.3)
#25†		683	8192	2.00e+2	39.3	3.0	(5.6)	1.30e+2	(65.1)	1.32e+2	1.71e+2	(85.5)
#26‡		683	8192	1.43e+2	55.1	4.2	(7.8)	9.20e+1	(64.4)	8.96e+1	1.15e+2	(80.7)
#27†		1366	16384	3.89e+2	10.1	1.2	(2.0)	2.16e+2	(55.6)	2.34e+2	2.69e+2	(69.2)
#28	1024^3	2732	32768	1.61e+3	100.0	0.8	(1.2)	6.58e+2	(40.9)	9.25e+2	8.19e+2	(50.8)
#29	64^3	2	4	2.20e+1		100.0	(100.0)	1.85e+1	(84.2)	1.40e+1	2.15e+1	(97.7)
#30	128^3	16	32	4.00e+1		55.0	(80.7)	3.49e+1	(87.2)	2.56e+1	3.94e+1	(98.6)
#31	256^3	128	256	6.96e+1		31.6	(64.8)	6.05e+1	(87.0)	4.52e+1	6.90e+1	(99.1)
#32	512^3	1024	2048	1.39e+2		15.9	(37.5)	1.14e+2	(82.3)	8.82e+1	1.36e+2	(98.2)

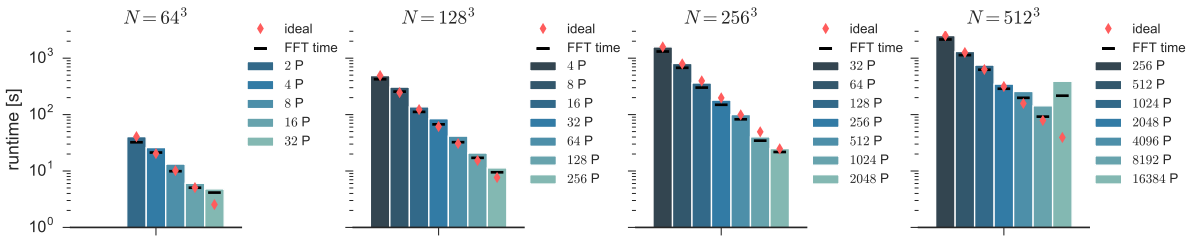


Figure 3.5: Summary of strong scaling efficiency of tumor inversion for real data. See Table 3.3 for exact timings. We display time to solution and time spent in the FFT (in seconds) as a function of the number of unknowns (in space) and the number of tasks.

tasks (runs #13–#19). The time consumed by the FFT accumulates to approximately 82 % of the overall runtime. Thus, the tumor inversion scalability is mainly inherited from the AccFFT. Similar conclusions can be drawn from the $N = 128^3$ and $N = 64^3$ experiments, yielding a slightly lower but still acceptable parallel efficiency of 67 % from 4 to 256 MPI tasks and 52 % from 2 to 32 MPI tasks, respectively The ideal runtime (assuming 100% parallel efficiency) along with the actual runtime and the amount of time spent in the FFT are summarized in Figure 3.5 for the strong scaling experiments given in Table 3.3. Considering the $N = 512^3$ runs, we observe a degradation of the parallel scalability for more than 2048 MPI tasks (runs #24–#27). Our analysis showed that MPI routines generate an increasing overhead using more than 2048 MPI tasks with 12 tasks per node. Increasing the MPI buffer size and the maximum message size for the MPI eager messaging protocol improved the performance of run #25 by around 30 % (run #26). Usually, the diffusion solve consumes up to 98 % of the overall runtime, which, drops down to 50 % for the runs that show poor scalability, indicating that performance is lost due to non-optimized MPI settings. Summarizing, for up to a total number of 2048 MPI tasks, we get almost optimal strong scaling results with a parallel efficiency from 60 % up to 100 %, cf. Figure 3.5. We encounter some MPI related communication issues going beyond 2048 MPI tasks.

Weak Scaling. For the same setting, we analyze algorithmic and parallel weak scaling[11], indicated by eff_W and $\widetilde{\text{eff}}_W$, respectively, in Table 3.3 (exemplarily highlighted, runs #3, #10, #17, #24, #26). We observe an algorithmic weak scaling efficiency of around 30 %, increasing both, the number of unknowns in space

[11] Algorithmic scaling includes numerical effect such as increasing number of iterations, whereas parallel scaling only measures the scalability of a fixed number of executions of the components.

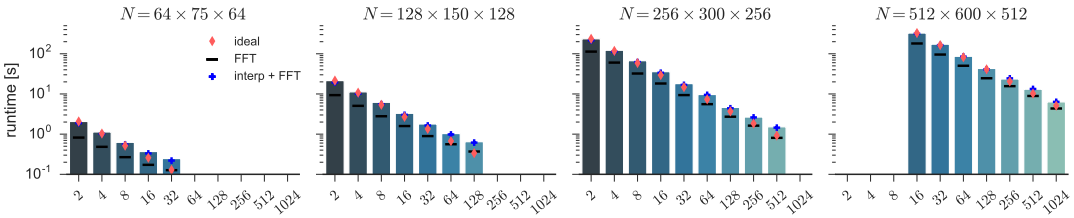


Figure 3.6: Strong scaling performance for diffeomorphic image registration. The results correspond to Table 3.4. We report the time to solution and the time spent in the computational kernels (summarized), respectively (in sec.) as a function of the number of unknowns (in space), and the number of MPI tasks.

and the number of MPI tasks by a factor of eight. Since the grid is not sufficiently large to resolve the diffusion operator, we observe a mesh-dependent number of iterations with increasing resolution—from 10 iterations per diffusion solve (on average) for $N = 64^3$ to 50 per diffusion solve for $N = 1024^3$. For the parallel weak scaling efficiency $\widetilde{\text{eff}}_W$, where we keep the number of PCG iterations constant yielding an efficiency of about $45\,\%$, again increasing the number of unknowns and the number of tasks by a factor of eight. With respect to parallel efficiency, the weak scaling performance perfectly correlates with the FFT, which in turn deteriorates due to the communication. Using only two MPI tasks per node (one per socket), the (parallel) weak scaling efficiency increases to $80\,\%$ going from resolution 64^3 to 128^3 (runs $\#29$–$\#32$), which is acceptable considering the overhead for memory allocation and increasing communication time. We solve the inverse tumor problem for realistic brain geometries with a 256^3 resolution up to a relative gradient of $1.20\text{e}{-4}$ in 22 minutes using 512 MPI tasks on *Hazel Hen*. This corresponds to a reduction of the objective function by three orders of magnitude and a relative reconstruction mismatch of $0.2\,\%$.

Table 3.4: Computational performance for our distributed-memory algorithm for diffeomorphic image registration. This study is performed on TACC's *Lonestar 5* (runs #1 to #28; NIREP datasets; see Figure 3.2) and HLRS' *Hazel Hen* (SYN datasets). We use 12 MPI tasks per node. We set the upper limit for the Gauss–Newton iterations to three/five and the number of PCG iterations to five/ten for NIREP/SYN. We report (from left to right) the total time spent in the inversion (runtime/time-to-solution), the strong scaling efficiency, the time spent in the computational kernels (spectral operations/FFT and interpolation), respectively (in seconds) as a function of the number of unknowns N (in space), and the number of nodes and tasks. We report the max. value across all MPI tasks; "total FFT" corresponds to the time spent in all spectral operations; "FFT comm." is the communication time; "total int." is the overall time spent in the interpolation; "int. kernel" is the time spent on the execution of the interpolation operator, and "int. comm" is the communication time for the interpolation. We also report the strong scaling efficiency and the percentage of the total interpolation and FFT time with respect the overall runtime.

	N	nodes	tasks	runtime	eff.	total FFT	([%])	FFT comm.	total int.	([%])	int. kernel	int. comm.
#1	$(64, 75, 64)$	1	2	2.07	100.0	8.21e−1	(39.7)	9.31e−2	1.05	(51.0)	7.18e−1	6.71e−2
#2		1	4	1.13	91.4	4.83e−1	(42.7)	7.36e−2	5.30e−1	(46.8)	3.70e−1	3.68e−2
#3		1	8	6.26e−1	82.6	2.67e−1	(42.7)	4.44e−2	2.87e−1	(45.9)	1.95e−1	1.78e−2
#4		2	16	3.70e−1	69.9	1.72e−1	(46.6)	6.88e−2	1.59e−1	(43.1)	9.84e−2	9.95e−3
#5		3	32	2.47e−1	52.3	1.27e−1	(51.3)	7.51e−2	9.28e−2	(37.5)	5.00e−2	5.54e−3
#6	$(128, 150, 128)$	1	2	2.15e+1	100.0	9.35	(43.4)	5.07e−1	9.91	(46.0)	6.83	8.22e−1
#7		1	4	1.13e+1	95.4	5.06	(44.9)	6.73e−1	5.05	(44.7)	3.42	4.28e−1
#8		1	8	6.19	86.9	2.79	(45.0)	3.89e−1	2.78	(44.9)	1.82	2.32e−1
#9		2	16	3.30	81.6	1.59	(48.2)	5.52e−1	1.44	(43.6)	9.24e−1	1.14e−1
#10		3	32	1.79	75.2	8.94e−1	(50.0)	3.86e−1	7.50e−1	(41.9)	4.63e−1	5.93e−2
#11		6	64	1.04	65.0	5.61e−1	(54.2)	3.68e−1	4.22e−1	(40.8)	2.32e−1	3.05e−2
#12		11	128	6.49e−1	51.8	3.69e−1	(56.9)	2.64e−1	2.47e−1	(38.1)	1.22e−1	1.90e−2
#13	$(256, 300, 256)$	1	2	2.36e+2	100.0	1.13e+2	(48.0)	6.40	1.02e+2	(43.4)	5.88e+1	1.05e+1
#14		1	4	1.22e+2	96.4	6.04e+1	(49.3)	8.33	5.25e+1	(42.9)	2.95e+1	5.51
#15		1	8	6.73e+1	87.6	3.23e+1	(48.0)	4.58	2.99e+1	(44.5)	1.56e+1	3.95
#16		2	16	3.59e+1	82.1	1.82e+1	(50.7)	7.80	1.55e+1	(43.2)	7.82	2.69
#17		3	32	1.81e+1	81.4	9.38	(51.8)	3.60	7.48	(41.3)	3.92	1.18
#18		6	64	9.70	76.0	5.57	(57.5)	3.30	3.93	(40.5)	1.99	7.07e−1
#19		11	128	4.63	79.5	2.72	(58.6)	1.50	1.69	(36.5)	9.98e−1	1.49e−1
#20		22	256	2.66	69.2	1.63	(61.1)	1.21	9.81e−1	(36.9)	4.98e−1	8.26e−2
#21		43	512	1.52	60.5	8.09e−1	(53.1)	6.15e−1	6.43e−1	(42.2)	2.54e−1	6.80e−2
#22	$(512, 600, 512)$	2	16	3.28e+2	100.0	1.81e+2	(55.0)	4.84e+1	1.35e+2	(41.2)	6.33e+1	2.50e+1
#23		3	32	1.73e+2	94.7	9.57e+1	(55.2)	2.85e+1	6.86e+1	(39.6)	3.17e+1	1.15e+1
#24		6	64	8.66e+1	94.7	5.04e+1	(58.2)	2.42e+1	3.48e+1	(40.2)	1.59e+1	6.31
#25		11	128	4.32e+1	94.9	2.46e+1	(56.9)	1.10e+1	1.63e+1	(37.8)	7.95	2.84
#26		22	256	2.36e+1	87.0	1.56e+1	(66.3)	1.08e+1	8.83	(37.5)	4.03	1.64
#27		43	512	1.31e+1	78.1	8.92	(67.9)	6.56	4.41	(33.6)	2.03	7.27e−1
#28		86	1024	6.35	80.7	4.35	(68.5)	3.52	2.02	(31.9)	1.02	2.08e−1
#29	1024^3	11	128	1.97e+2	100.0	1.20e+2	(60.9)	3.30e+1	6.90e+1	(35.0)	2.35e+1	2.23e+1
#30		22	256	9.88e+1	99.7	6.17e+1	(62.5)	2.15e+1	3.49e+1	(35.4)	1.16e+1	1.16e+1
#31	2048^3	86	1024	2.10e+2	100.0	1.37e+2	(65.0)	4.33e+1	7.21e+1	(34.3)	2.73e+1	2.40e+1
#32		171	2048	1.11e+2	94.8	7.17e+1	(64.7)	2.63e+1	3.64e+1	(32.8)	1.35e+1	1.05e+1
#33	4096^3	342	4096	4.42e+2	100.0	3.22e+2	(72.8)	1.31e+2	1.17e+2	(26.4)	4.20e+1	3.97e+1
#34		684	8192	2.38e+2	93.1	1.73e+2	(72.9)	8.27e+1	6.25e+1	(26.3)	2.10e+1	2.30e+1

Diffeomorphic registration

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 *Non-rigid Registration Evaluation Project* (NIREP) (27).[12] This data is illustrated in Figure 3.2. The original resolution of the data is $(N_0, N_1, N_2) = (256, 300, 256)$.[13] We also consider a simple test example to demonstrate large-scale results for grid sizes of up to 4096^3. This results in an inversion for ~ 200 billion unknowns (if we just count the velocity field $\boldsymbol{v}$ and ignore the state and adjoint fields). This synthetic problem (SYN) is generated by solving the forward problem. We set the template image to $m_T(\boldsymbol{x}) = (\sin(x_1)^2 + \sin(x_2)^2 + \sin(x_3)^2))/3$ and transport it with the velocity velocity $\boldsymbol{v}(\boldsymbol{x}) = (v_1(\boldsymbol{x}), v_2(\boldsymbol{x}), v_3(\boldsymbol{x}),\ v_1(\boldsymbol{x}) = \sin(x_3)\cos(x_2)\sin(x_2),\ v_2(\boldsymbol{x}) = \sin(x_3)\cos(x_3)\sin(x_3),$ and $v_3(\boldsymbol{x}) = \sin(x_2)\cos(x_1)\sin(x_1)$.

Setup. We fix several parameters across all runs: We use twelve MPI tasks per node. We set the number of time steps to $n_t = 4$. We use an H^1 regularization model with a penalty on the divergence of $\boldsymbol{v}$ to control volume change (regularization weight $\beta_w = 1\text{e}{-4}$). We use a quadratic forcing sequence. We report a strong scaling analysis using the NIREP datasets with resolution levels $\kappa_l \cdot (256, 300, 256)$, $\kappa_l \in \{1/4, 1/2, 1, 2\}$. We empirically set the regularization parameter for the velocity $\boldsymbol{v}$ to $\beta_v = 1\text{e}{-2}$. We limit the number of Newton iterations to three and the number of PCG iterations to five. The relative tolerance for the gradient is set to $1\text{e}{-1}$

[12] The data can be downloaded from http://nirep.org; the interested reader is referred to (27) for more details. We consider the datasets `na01` and `na02` for our experiments.

[13] We transfer the images to a finer or coarser grid based on a cubic interpolation model. We band-limit the data by applying a Gaussian smoothing operator with a spatial bandwidth of h_i, $i = 1, 2, 3$.

(we do not reach this tolerance for these runs). We limit the Newton iterations to five and the Krylov iterations to ten for the synthetic large scale runs ($N = 1024^3$, $N = 2048^3$, and $N = 4096^3$). The relative tolerance for the gradient is $1e{-}2$. We also report a run for the entire inversion. We set the relative tolerance for the gradient to $1e{-}1$. This run is performed on the same images used in (109).

Results. We report the strong scaling analysis for the NIREP datasets as well as the synthetic large scale run in Table 3.4. We illustrate the strong scaling results for the real data also in Figure 3.6. We report accumulated timings for the max values across all MPI tasks in seconds as well as the strong scaling efficiency. We also report the percentage of the total interpolation and FFT time with respect to the overall runtime.

Observations. We spend almost all time (~90%) in the execution of the FFT and the interpolation (see Table 3.4 and Figure 3.6). We achieve excellent strong scaling results for clinically relevant problem sizes, with an almost perfect scaling for the execution of our computational kernels (application of the FFT and evaluation of the interpolation operator locally). We can fit conically relevant problems on one a single node with 64GB of memory (see, e.g., run $\#13$). The number of MPI tasks for a specific problem size is, in our current implementation, limited by the support of the interpolation kernel/the size of the ghost layer; for a cubic interpolation model, we need at least $3 \times 3 \times N_2$ points on each MPI task.

We showcase results that demonstrate that our solver can be used to solve registration problems of unprecedented scale (4096^3 resulting in ~ 200 billion unknowns (a problem size that is $64\times$ bigger than the state-of-the-art presented at last year's SC (109)). The weak scaling efficiency for these runs is 62.7%, 44.6%,

and 32.6% when comparing the runtime for run $\#1$ to the runs #9, #19, through #28. The communication time for the FFT increases as we increase the problem size. We expect to see this behavior, since the FFT is communication bound. This negatively affects our weak scaling results (the total FFT time is for run #28 almost exclusively dominated by the FFT communication time; 80%). This observation is consistent with the results we report for the tumor case. The weak scaling performance for the interpolation is very good (50% in total when comparing runs $\#1$ and $\#28$). Having good strong scaling performance is critical for image registration. We can see that we achieve a quite good strong scaling for our runs (between 52.1% (run $\#5$) and 80% (run $\#28$); this is in particular visible in Figure 3.6. The execution times for our computational kernels show excellent strong scaling results. The efficiency eventually deteriorates as we increase the number of cores. The amount of data we have to communicate in the interpolation phase essentially depends complexity of the characteristic. In a worst case scenario (which we do not expect in practical applications) all departure points may end up on one node. Semi-Lagrangian schemes can help us to alleviate potential imbalances (in contrast to purely Lagrangian schemes (12)). If the characteristic becomes to complex, we can introduce more time points to our problem. However, this will negatively affect the time-to-solution, as we have to perform more interpolations.[14]

We analyze our results for original resolution of the NIREP data sets in more detail: For these experiments, we can reduce the gradient by a factor of $3.17\text{e}{-1}$ and the mismatch between the transported template image and the reference image by a factor of $1.89\text{e}{-1}$. The balance of the runtime spent on the individual components of our scheme is the same throughout all runs. We perform 32 PDE

[14] An analysis of the number of interpolations we have to perform can be found in §3.2.

solves. The PDE solves is where the entire runtime goes. For instance, for run $\#13$ we spend 87% of the total runtime (2.36e+2 sec. of 2.06e+2 sec.) on solving the transport equations of our system. These solves are done in the evaluations of (i) the objective functional (four evaluations.; 1.85e+1 sec.), the gradient (four evaluations.; 2.69e+1 sec.), and the Hessian matvec (twelve evaluations.; 1.78e+2 sec.). As we can see from this analysis, we spend most of the time runtime on the Hessian matvec. When we switch from two MPI tasks to 512 we can reduce the run time by a factor of about 150 (1.52 sec.; 60.5% efficiency). For the synthetic case we can solve our problem (reduce the gradient by two orders of magnitude) within 2 iterations (14 PDE solves; 4 Hessian matvecs; relative change in the gradient: 9.66e-3; relative change of the mismatch: 7.18e-2; run#30). The overall weak scaling efficiency for these runs is 82.9% (128 MPI tasks for 1024^3 versus 8192 MPI tasks for 4096^3). For an entire registration solve on 256 MPI tasks on 32 nodes, the new formulation requires 14 Hessian matvecs for the same problem reported in Table 5 in (111) (which required 43 matvecs) with similar registration quality. Bringing everything together (formulation and fast computational kernels), we can reduce the runtime by a factor of $8\times$ over the state-of-the art.

Conclusions

We presented SIBIA, a computational framework for coupling biophysical models with medical image analysis. To the best of our knowledge, SIBIA is the first work on scalable algorithms for an integrated approach for biophysics-based image analysis in brain tumor imaging. Our major accomplishments and observations are the following: we achieve ***excellent strong scaling*** performance on clinically relevant problem sizes for both algorithms (from 60% up to 100% parallel effi-

ciency). Moreover, 80% to 90% of the runtime is spent in the computational kernels (FFT and interpolation). We were able to improve the performance of these kernels by a factor of roughly 8 (interpolation) and 2 (FFT) over the state of the art. For the tumor problem, we saw that the scalability is inherited from AccFFT (56, 58). The preconditioner for the tumor solver is not mesh independent for real data at this resolution levels. For the registration, we solved an inverse problem with ***~200 billion unknowns on up to 8192 cores***—a problem size that is 64× larger than (111), rendering our solver applicable to, e.g., the registration of high-resolution CLARITY imaging data (177). Overall we have improved the time-to-solution by a factor of about 8 compared to (111).

Chapter 4

Coupled Inverse Tumor Problem and Registration

needs at least two-time snapshots of segmented patient images. As discussed in chapter 1, the inverse tumor solver requires the healthy structure of the patient's brain prior to the tumor as input, or at least a previous time snapshot of the tumor distribution so that we can run the forward solve. This data necessary to perform the simulations and invert for the unknowns to match the observed data in the tumor-bearing image. However, in most cases such data is not available. We address this limitation by solving a coupled inverse tumor and inverse registration problem. In this approach, we solve the inverse tumor problem in an "atlas" brain, whose structure is known to us. Then through the registration inverse problem, we compute correspondences between the images in the atlas domain and the patient's. The coupled optimization problem would be to find this correspondence (through registration velocity), and the tumor parameters that can minimize the mismatch between reconstructed and observed data from the patient.

In this chapter, we will discuss the formulation for the coupled problem and present our Picard iterative algorithm for solving the inverse problem. We present numerical experiments and quantify the performance of the coupled algorithm on synthetically generated tests.

This chapter is based on joint collaboration between the author and Klaudius Scheufele, Andreas Mang, Prof. Miriam Mehl and Prof. George Biros. The author implemented the forward and inverse tumor problems, as well as semi-Lagrangian and spectral solvers for the registration problem. Klaudius implemented the coupling algorithm and performed the benchmarks and created the visualizations. Andreas implemented the registration framework. Prof. Biros and Prof. Mehl contributed to the algorithms, and supervised the research. In total, all authors contributed to the coupling formulation and its implementation.

4.1 Coupling Formulation

c	Normalized tumor concentration
c_i	Normalized tumor concentration at $t = i$, $i \in \{0, 1\}$
c^A	Normalized tumor concentration in atlas domain
c^P	Normalized tumor concentration in patient domain
m^A	Vector of material properties (white/gray matter, etc.) in atlas domain
m^P	Vector of material properties (white/gray matter, etc.) in patient domain
Ω	Spatial domain
N_m	Number of advectible material properties
N_p	Parametrization size for initial tumor distribution
$\boldsymbol{m} \in \mathbb{R}^{N_m}$	Vector of material properties
$D = \operatorname{div} \mathbf{K}(\boldsymbol{m})\nabla$	Diffusion operator
$\mathbf{K}(\boldsymbol{m})$	Diffusion coefficient tensor (function of material properties)
K	Diffusion tensor
$R(c) = \rho c(1-c)$	Tumor reaction operator
ρ	Reaction coefficient
$\boldsymbol{p}$	Reconstruction initial condition parametrization
Φ	Reconstruction initial condition parametrization basis function
β_p	Regularization parameter for ℓ_2 norm of $\boldsymbol{p}$
β_v	Regularization parameter for registration velocity
α	Adjoint variable for tumor growth equations
λ^m	Adjoint variable for transport equation for material properties
λ^c	Adjoint variable for transport equation for material properties
$\mathcal{F}_{\boldsymbol{v}}$	Forward transport operator with velocity $\boldsymbol{v}$

Table 4.1: Basic notation used in this chapter

In the coupling framework, we seek a registration velocity $\boldsymbol{v}$, and a set of model parameters $\boldsymbol{p}$ for the tumor problem, such that the following objective functional is minimized:

$$\min_{\boldsymbol{v},\boldsymbol{p}} \mathcal{J}[\boldsymbol{v}, p] = \underbrace{\frac{1}{2}\|c_1^A - c_1^P\|_{L^2(\Omega)}^2 + \frac{1}{2}\|\boldsymbol{m}_1^A - \boldsymbol{m}_1^P\|_{L^2(\Omega)}^2}_{\text{tumor and geometry mismatch between patient and atlas}} + \underbrace{\frac{\beta_v}{2}\|\nabla\boldsymbol{v}\cdot\nabla\boldsymbol{v}\|_{L^2(\Omega)}^2 + \frac{\beta_p}{2}\|\Phi p\|_2^2}_{\text{Regularization}} \tag{4.1a}$$

subject to

$$\alpha: \quad \partial_t c^A - \nabla \cdot K \nabla c^A - \rho c^A (1 - c^A) = 0 \qquad \text{in } \Omega \times (0,1], \tag{4.1b}$$

$$\alpha_0: \quad c_0^A = \Phi p \qquad \text{in } \Omega \times \{0\}, \tag{4.1c}$$

$$\xi: \quad \boldsymbol{m}_1^A = (1 - c_1^A)\boldsymbol{m}_0^A \qquad \text{in } \Omega \times \{1\}, \tag{4.1d}$$

$$\lambda^m: \quad \partial_t \boldsymbol{m}^P + \nabla \boldsymbol{m}_P \cdot \boldsymbol{v} = 0 \qquad \text{in } \Omega \times (0,1], \tag{4.1e}$$

$$\lambda_0^m: \quad \boldsymbol{m}_0^P = \boldsymbol{m}_0^P(\boldsymbol{x}) \qquad \text{in } \Omega \times \{0\}, \tag{4.1f}$$

$$\lambda^c: \quad \partial_t c^P + \nabla c^P \cdot \boldsymbol{v} = 0 \qquad \text{in } \Omega \times (0,1], \tag{4.1g}$$

$$\lambda_0^c: \quad c_0^P = c_0^P(\boldsymbol{x}) \qquad \text{in } \Omega \times \{0\}, \tag{4.1h}$$

where the constraints are a diffusion-reaction tumor model (discussed in chapter 2), and transport equations for the image registration problem. The superscript A indicates atlas-related fields; and the superscript B indicates patient-related fields. Moreover, c_1^A is the concentration of tumor cells in the atlas after solving the forward problem (Eq. 4.1b) using c_0^A as initial condition (Eq. 4.1c). Furthermore, c_1^P is the patient's tumor concentration transported into the atlas domain by solving Eq. 4.1e using c_0^P as the initial distribution. We define the forward advection operator as $\mathcal{F}_v$, such that $c_1^P = \mathcal{F}_v(c_0^P)$. Notice that c_0^P is the actual tumor that we observe in the patient from imaging. Due to the ill-posedness of the inverse problem, we have to impose regularity constraints. Here $\nabla \boldsymbol{v} \cdot \nabla \boldsymbol{v}$ is the regularization operator for the velocity, and β_p is the regularization parameter for the tumor parameters. Additionally, $\boldsymbol{m}$ refers to the vector of material properties, which includes white matter, gray matter, CSF, and possibly other tissue types. We use a separate notation for the tumor concentration and denote it by c for clarity of the equations.

The variables in gray denote the corresponding adjoint variables for each of the

constraints. To solve the optimization problem, we form the Lagrangian as follows:

$$
\begin{aligned}
\mathcal{L}\left(c^P, c^A, \boldsymbol{m}^p, \boldsymbol{v}, \boldsymbol{p}, \alpha, \lambda_{\boldsymbol{m}}, \lambda_c, \boldsymbol{\xi}\right) = & \mathcal{J}[\boldsymbol{v}, p] \\
& + \int_T \quad \left\langle \alpha, \partial_t c^A - \nabla \cdot \left(K \nabla c^A\right) - \rho c^A \left(1 - c^A\right) \right\rangle_{L^2(\Omega)} dt \\
& + \quad \left\langle \alpha_0, (c_0^A - \Phi p) \right\rangle_{L^2(\Omega)} \\
& + \int_T \quad \left\langle \lambda_{\boldsymbol{m}}, \partial_t \boldsymbol{m}^P + \nabla \boldsymbol{m}^P \cdot \boldsymbol{v} \right\rangle_{L^2(\Omega)} dt \\
& + \quad \left\langle \lambda_0^{\boldsymbol{m}}, \boldsymbol{m}_0^P - \boldsymbol{m}_0^P(\boldsymbol{x}) \right\rangle_{L^2(\Omega)} \\
& + \int_T \quad \left\langle \lambda_c, \partial_t c^P + \nabla c^P \cdot \boldsymbol{v} \right\rangle_{L^2(\Omega)} dt \\
& + \quad \left\langle \lambda_0^c, c_0^P - c_0^P(\boldsymbol{x}) \right\rangle_{L^2(\Omega)} \\
& + \quad \left\langle \boldsymbol{\xi}, \boldsymbol{m}_1^A - (1 - c_1^A) \boldsymbol{m}_0^A \right\rangle_{L^2(\Omega)}
\end{aligned}
$$

The first order optimality conditions can be obtained by taking variations with respect to adjoint, state, and inversion variables. This leads to the following system of PDEs:

State, Adjoint and Inversion Equations for Tumor:

$$\delta_\alpha \mathcal{L} = 0: \qquad \partial_t c_A - \nabla \cdot (\nabla k c_A) - \rho c_A (1 - c_A) = 0 \qquad \text{in } \Omega \times (0, 1] \quad (4.2a)$$

$$\delta_{\alpha^0} \mathcal{L} = 0: \qquad c_A^0 - \Phi p = 0 \qquad \text{in } \Omega \times \{0\} \quad (4.2b)$$

$$\delta_{c_A} \mathcal{L} = 0: \qquad -\partial_t \alpha - \nabla \cdot (\nabla k \alpha) - \alpha \rho + 2 \alpha \rho c_A = 0 \qquad \text{in } \Omega \times [0, 1) \quad (4.2c)$$

$$\delta_{c_A^1} \mathcal{L} = 0: \qquad \alpha^1 = c_P^1 - c_A^1 - \sum_{i=1}^{3} (\boldsymbol{\xi})_i (\boldsymbol{m}_A^0)_i \qquad \text{in } \Omega \times \{1\} \quad (4.2d)$$

$$\delta_p \mathcal{L} = 0: \qquad \beta_p \Phi^\star \Phi p - \Phi^\star \alpha^0 = 0 \qquad \text{in } \mathcal{R}^{N_p} \quad (4.2e)$$

State, Adjoint and Inversion Equations for Registration:

$$\delta_{\lambda^m}\mathcal{L} = 0: \quad \partial_t \boldsymbol{m}^P + \nabla \boldsymbol{m}^P \cdot \boldsymbol{v} = 0 \quad \text{in } \Omega \times (0,1] \tag{4.3a}$$

$$\delta_{\lambda_1^m}\mathcal{L} = 0: \quad \boldsymbol{m}_0^P - \boldsymbol{m}_0^P(\boldsymbol{x}) = 0 \quad \text{in } \Omega \times \{0\} \tag{4.3b}$$

$$\delta_{\lambda^c}\mathcal{L} = 0: \quad \partial_t c^P + \nabla c^P \cdot \boldsymbol{v} = 0 \quad \text{in } \Omega \times (0,1] \tag{4.3c}$$

$$\delta_{\lambda_1^c}\mathcal{L} = 0: \quad c_0^P - c_0^P(\boldsymbol{x}) = 0 \quad \text{in } \Omega \times \{0\} \tag{4.3d}$$

$$\delta_{c^P}\mathcal{L} = 0: \quad -\partial_t \lambda^c - \nabla \cdot (\lambda^c \boldsymbol{v}) = 0 \quad \text{in } \Omega \times [0,1) \tag{4.3e}$$

$$\delta_{c_1^P}\mathcal{L} = 0: \quad \lambda_1^c = c_1^A - c_1^P \quad \text{in } \Omega \times \{1\} \tag{4.3f}$$

$$\delta_{\boldsymbol{m}^P}\mathcal{L} = 0: \quad -\partial_t \lambda^{\boldsymbol{m}} - \nabla \cdot (\lambda^{\boldsymbol{m}} \boldsymbol{v}) = 0 \quad \text{in } \Omega \times [0,1) \tag{4.3g}$$

$$\delta_{\boldsymbol{m}_1^P}\mathcal{L} = 0: \quad \lambda_1^{\boldsymbol{m}} = \boldsymbol{m}_1^A - \boldsymbol{m}_1^P \quad \text{in } \Omega \times \{1\} \tag{4.3h}$$

$$\delta_{\boldsymbol{v}}\mathcal{L} = 0: \quad -\beta_v \Delta \boldsymbol{v} + \int_0^1 \lambda \nabla m \, \mathrm{d}t \quad \text{in } \Omega, \tag{4.3i}$$

State and Adjoint Equation of Coupling:

$$\delta_{\boldsymbol{m}_A^1}\mathcal{L} = 0: \quad \boldsymbol{\xi} + \boldsymbol{m}_1^A - \boldsymbol{m}_1^P = 0 \tag{4.4a}$$

$$\delta_{\boldsymbol{m}_A^1}\mathcal{L} = 0: \quad \boldsymbol{m}_1^A = \boldsymbol{m}_0^A(1 - c_1^A) \tag{4.4b}$$

Similar to the inverse tumor or registration problems, we have to solve the optimization problem iteratively. Having the first-order optimality conditions, we can use variants of gradient descent or Newton methods. To compute the coupling gradient (i.e. the gradient of the objective functional w.r.t. $\boldsymbol{v}$ and $\boldsymbol{p}$), we need to start with an initial guess for the tumor parameters, $\boldsymbol{p}$, and an initial velocity, $\boldsymbol{v}$, for the inverse registration problem (both can be set to zero). We then solve the forward tumor problem in the atlas space (Eq. 4.1b) and obtain c_1^A. Then we have to transport the patient's material properties and tumor (i.e. $\boldsymbol{m}_0^P$ and c_0^P) into the

atlas domain by solving Eq. 4.1e and Eq. 4.1g to obtain $\boldsymbol{m}_1^P$ and c_1^P. Then from Eq. 4.4a and Eq. 4.4b, we can compute $\boldsymbol{\xi}$. Substituting $\boldsymbol{\xi}$ and c_1^P into Eq. 4.2d we obtain the terminal condition for α_1. Then we have to solve the adjoint equation Eq. 4.2c to obtain α_0. Substituting its value into Eq. 4.2e gives us the gradient with respect to $\boldsymbol{p}$. Furthermore, solving Eq. 4.3g and Eq. 4.3e and substituting the results into Eq. 4.3i gives us the gradient with respect to the velocity. To solve the inverse problem, we have to then update the initial guess for $\boldsymbol{p}$ and $\boldsymbol{v}$ and repeat this process.

4.2 Picard Iteration Scheme

Although a fully coupled method in which we use a gradient descent or a Newton method is the method of choice, the complexity of coupling the tumor-growth implementation and registration is quite high. So as a first step in testing the overall methodology, we propose a Picard iteration scheme, that corresponds to a nonlinear Gauss-Seidel scheme for the optimality conditions. We found that this scheme is quite effective. We describe the details in this section.

In the Picard scheme, we first solve the inverse registration problem and then transport the results into the atlas domain using the resulting velocity. Then we solve an inverse tumor problem using this transported tumor data, and update the tumor parameters. This process has to be repeated until convergence. In detail the Picard iteration is executed as follows:

We first start with an initial guess for the tumor model parameters, $\boldsymbol{p}$. Then we solve the forward tumor problem in the atlas domain using Eq. 4.1b, to obtain c_1^A. Afterwards, we solve an inverse registration problem to find correspondences between the atlas (c_1^A and $\boldsymbol{m}^A(1 - c_1^A)$) and the patient ($c^P$, $\boldsymbol{m}^P$). After finding

the registration velocity $\boldsymbol{v}$, we then transport the patient's tumor into the atlas domain using this velocity to obtain c_1^P. Then, we solve an inverse tumor problem to update the tumor model parameters $\boldsymbol{p}$, so that we can match this distribution (i.e. c_1^P). We repeat this process until convergence.

In the early iterations of the Picard scheme, we start with a large regularization parameter for the registration problem, and then reduce it. This is necessary since in the early iterations the atlas does not have the right tumor distribution and using a small regularization for the velocity would be equivalent to matching noise.

4.3 Numerical Experiments

We test the Picard scheme on several tests using synthetic data. In the first test, we consider a case where we create the patient's brain, $\boldsymbol{m}^P$, from the atlas's brain, $\boldsymbol{m}^A$, by using a synthetic velocity $\boldsymbol{v}^\star$. The synthetic velocity is computed by solving an inverse registration problem between two real images of healthy brains. We then use this velocity as ground truth, and advect $\boldsymbol{m}^A$ to obtain $\boldsymbol{m}^P$. We also create a synthetic tumor, by solving the forward tumor model and transporting the results using this velocity into the patient's domain.

Using this synthetically generated patient brain with tumor, we solve the coupled problem without using any test generation parameters. That is we do not use the tumor parameters that generated the patient tumor, or the ground truth velocity used to generate the brain.

To measure the coupling performance, we compute the registration error between the atlas and patient's brain without tumor, and then compare it with the registration error between the tumor-bearing patient image and the atlas, derived by solv-

ing the coupling problem. For the experiment we use $N = 128^3$, $dt = 0.1$, $N_t = 16$, $\rho = 15$, $k = 0$, $N_p = 125$, and $\beta_p = 2.5\text{E-}4$. For the registration regularization we use H2 seminorm.

After solving the registration problem between the atlas and the patient without tumor, we obtain a reconstruction velocity $\boldsymbol{v}$. We can then compute the relative mismatch between the two images which is equal to:

$$\frac{\|\boldsymbol{m}^A - \mathcal{F}_v(\boldsymbol{m}^P)\|}{\|\boldsymbol{m}^A\|} = 1.106\text{E-}1.$$

We expect the coupling framework to achieve an accuracy in this range. We first assume that we know the reaction and diffusion coefficients that was used to generate the test case, but not the initial distribution (we later relax this assumption). The results for the Picard iteration of the coupling is given in Table 4.2. The final reconstruction result (corresponding to iteration #14), gives a relative mismatch of 10.6% in the atlas domain, The final mismatch between the reconstructed and ground truth tumor distribution is also 9.11% and 5.18%, respectively. A visualization of the mismatch between the geometry ($\boldsymbol{m}^A - \mathcal{F}_v(\boldsymbol{m}^P)$), and the tumor $c_1^A - \mathcal{F}_v(c^P)$ is shown in Fig. 4.2. Notice that parts of this error is due to the information loss in the inversion, and also the advection error. To verify this, we computed advection error by performing a forward and backward advection solve on the ground truth tumor distribution. The resulting error is:

$$\frac{\|c_1^A - \mathcal{F}_{v^\star}\left(\mathcal{F}_{-v^\star}(c_1^A)\right)\|}{\|c_1^A\|} = 5.92\text{e}{-2}.$$

Now we relax the input to the coupling solve, and do not use the ground truth tumor model parameters. That is we do not use the reaction and diffusion coefficient

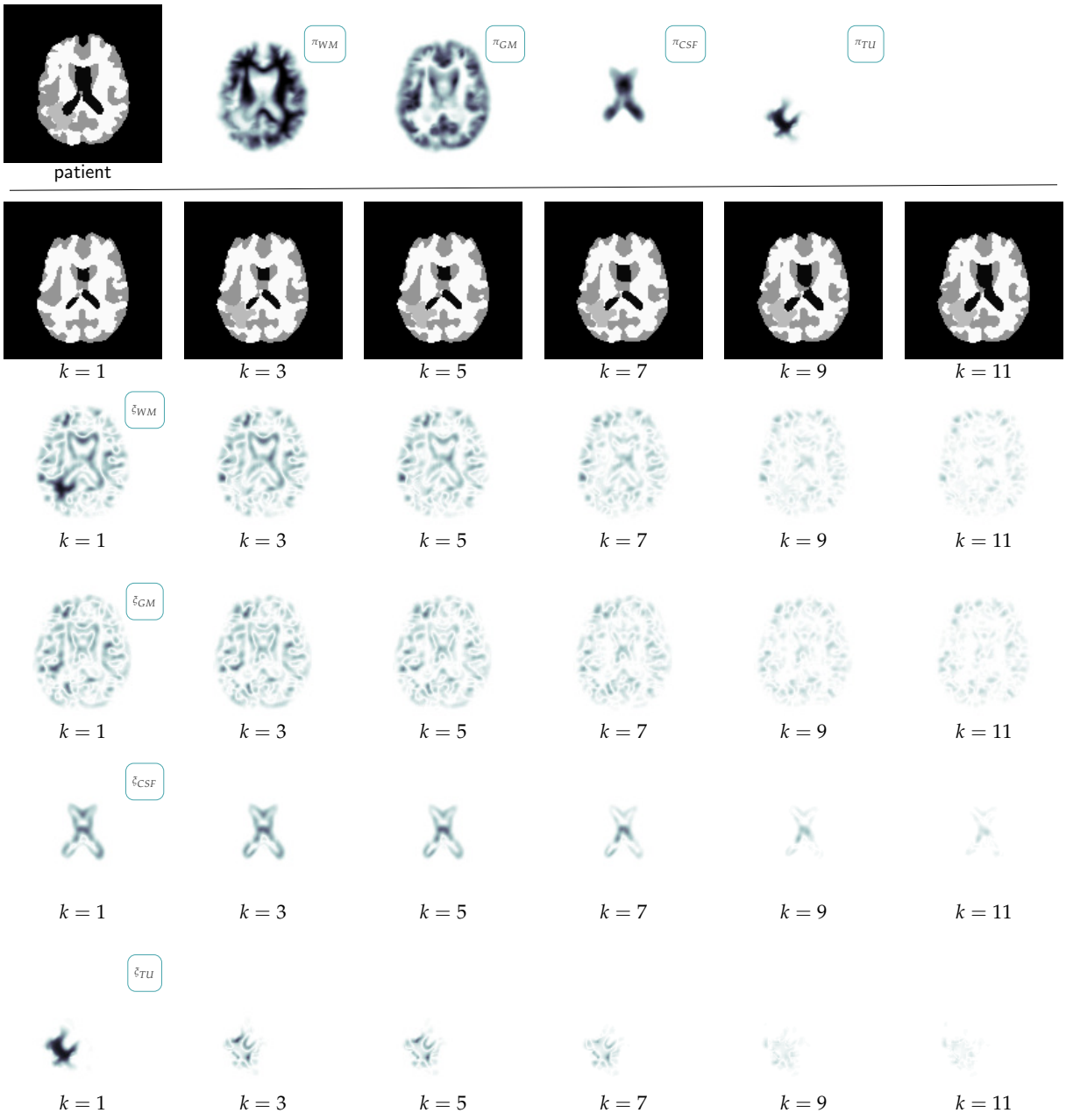


Figure 4.1: Visualization of the mismatch between the geometry and tumor distributions for different Picard iterations (denoted by k). The first row shows the patient's brain and its material properties. The second row shows the reconstructed geometry (i.e. atlas's brain transported into the patient's domain). The rest of the rows show the mismatch for different Picard iteration for white matter, gray matter, CSF, and tumor, respectively. Here darker areas show areas with larger mismatch and vice versa for white areas.

Table 4.2: The table shows the mismatch of geometries and tumor both in the atlas and in the patient domain, for different Picard iterations. The final reconstruction result (corresponding to iteration #14), gives a relative mismatch of 10.6% in the atlas domain, which is the same order as the registration error achieved above. The final mismatch between the reconstructed and ground truth tumor distribution is also 9.11%.

Itr	β_v	$\frac{\|\boldsymbol{m}^A - \mathcal{F}_v(\boldsymbol{m}^P)\|}{\|\mathcal{F}_v(\boldsymbol{m}^P)\|}$	$\frac{\|c_1^A - \mathcal{F}_v(c^P)\|}{\|\mathcal{F}_v(c^P)\|}$
0	1	4.19e−1	9.57e−1
1	1	3.76e−1	3.53e−1
2	1e−1	3.57e−1	3.37e−1
3	1e−1	3.57e−1	3.36e−1
4	1e−2	3.10e−1	2.97e−1
5	1e−2	3.10e−1	2.93e−1
6	1e−3	2.28e−1	2.27e−1
7	1e−3	2.28e−1	2.17e−1
8	1e−4	1.47e−1	1.42e−1
9	1e−4	1.46e−1	1.33e−1
10	1e−5	1.20e−1	1.07e−1
11	1e−5	1.15e−1	9.99e−2
12	1e−6	1.10e−1	9.59e−2
13	1e−6	1.08e−1	9.33e−2
14	1e−6	1.06e−1	9.11e−2

used to generate the test case, and test the coupling performance. The results are shown in Table 4.3. As one can see, we get very good reconstruction accuracy even for cases where the test data was generated with coefficients other than $\rho = 15$, and $k = 0$ which are used by the coupling. This is because the inverse solve changes the initial distribution of the tumor, so that it can match the patient's data.

4.4 Conclusions

In this chapter, we discussed our coupling formulation which can address a common limitation for the inverse tumor problem. This limitation stems from the fact that we usually do not have access to the healthy state of the patient. Without such information, it is not possible to perform the right forward solve. We

Table 4.3: The patient tumor is grown using different parameters $\rho \in \{5, 10, 15, 20\} \times k \in \{0, 10^{-2}, 10^{-3}\}$. The coupling is however always run with tumor reconstruction using $\rho = 15, k = 0$. The table shows the mismatch of geometries and tumor both in the atlas and in the patient domain.

test case setting	β_v	$\frac{\|\boldsymbol{m}^A - \mathcal{F}_v(\boldsymbol{m}^P)\|}{\|\mathcal{F}_v(\boldsymbol{m}^P)\|}$	$\frac{\|c_1^A - \mathcal{F}_v(c^P)\|}{\|\mathcal{F}_v(c^P)\|}$
$\rho = 5, k = 0$	1e−6	1.05e−1	1.39e−1
$\rho = 5, k = 10^{-3}$	1e−6	1.05e−1	1.46e−1
$\rho = 5, k = 10^{-2}$	1e−6	1.06e−1	1.89e−1
$\rho = 10, k = 0$	1e−6	1.05e−1	1.08e−1
$\rho = 10, k = 10^{-3}$	1e−6	1.05e−1	1.14e−1
$\rho = 10, k = 10^{-2}$	1e−6	1.06e−1	1.53e−1
$\rho = 15, k = 0$	1e−6	1.06e−1	9.11e−2
$\rho = 15, k = 10^{-3}$	1e−6	1.06e−1	9.92e−2
$\rho = 15, k = 10^{-2}$	1e−6	1.07e−1	1.33e−1
$\rho = 20, k = 0$	1e−6	1.07e−1	9.01e−2
$\rho = 20, k = 10^{-3}$	1e−6	1.07e−1	9.28e−2
$\rho = 20, k = 10^{-2}$	1e−6	1.08e−1	1.23e−1

discussed how the coupling framework addresses this limitation by solving the inverse tumor problem in the atlas domain and maps the data into the patient's domain through inverse registration problem.

We presented the optimality conditions and the Picard scheme for solving the corresponding optimization problem. We tested the performance of the method on synthetically generated tests, and showed how our solver performs even in cases where we do not have the ground truth tumor parameters.

Chapter 5

Tumor Model With Mass Effect

In Chapter 2, we considered a reaction-diffusion model for tumor growth. This is the simplest model of tumor growth behavior. It captures the two distinct behaviors of malignant tumor growth: proliferation (reaction) and infiltration (diffusion). However, even phenomenologically (that is in magnetic resonance images), gliomas exhibit a prominent characteristic not captured in the reaction-diffusion model: mass effect. "Mass effect" is a term-of-art used by clinicians to describe tissue displacement due to tumor growth forces. In this chapter, we discuss a more complex tumor model that attempts to capture mass effect. Essentially the model constitutes of a simple linear elastic model driven by a force that depends nonlinearly on the tumor concentration. The model has been introduced in the past in (81). Our main contribution here is the derivation of the adjoint equations, their numerical discretization, and the verification of the proposed scheme.

5.1 Linear Elasticity Model

Mass effect occurs due to the displacement imposed by tumor growth. This growth creates a deformation of the surrounding healthy tissues of the brain. We compute the displacement $\boldsymbol{u}$ by adding another constrain, the linear elasticity equation. Here we briefly derive the governing equations by following the seminal works of (65, 130).

Let $\boldsymbol{X}$ be position of the brain at the reference configuration, and $\boldsymbol{x}$ to be the deformed configuration at time t. Then the displacement $\boldsymbol{u}$ is defined as:

$$\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{X} = \boldsymbol{\phi}(\boldsymbol{X}, t) - \boldsymbol{x}, \tag{5.1}$$

where $\boldsymbol{\phi}$ is the deformation map. The elongation/contraction of an infinitesimal line segment of the tissue can be computed as:

c	Normalized tumor concentration
c_i	Normalized tumor concentration at $t = i$, $i \in \{0, 1\}$
Ω	Spatial domain
N_m	Number of advectible material properties
N_p	Parametrization size for initial tumor distribution
$\boldsymbol{m} \in \mathbb{R}^{\mathrm{N_m}}$	Vector of material properties
$D = \mathrm{div}\ \mathbf{K}(\boldsymbol{m})\nabla$	Diffusion operator
$\mathbf{K}(\boldsymbol{m})$	Diffusion coefficient tensor (function of material properties)
k_0	Inhomogeneous diffusion coefficient
k_f	Anisotropic diffusion coefficient
$\mathbf{T}$	Diffusion tensor
$R(c, \boldsymbol{m}) = \rho(\boldsymbol{m})c(1-c)$	Tumor reaction operator
$\rho(\boldsymbol{m})$	Reaction coefficient (function of material properties)
$L = \mu\Delta + (\lambda + \mu)\nabla\mathrm{div}$	Linear elasticity operator
$\mathbf{f}(\mathbf{c})$	Force function in linear elastic equation
d_i	Target tumor concentration at $t = i$, $i \in \{0, 1\}$
O_i	Observation operator at $t = i$, $i \in \{0, 1\}$
p	Initial condition parametrization
Φ	Initial condition parametrization basis function
β_p	Regularization parameter for ℓ_2 norm of $\boldsymbol{p}$
$\alpha,\ \alpha^\dagger$	Adjoint variable for tumor growth equations
$\boldsymbol{\psi},\ \boldsymbol{\psi}^\dagger$	Adjoint variable for transport equations

Table 5.1: Basic notation used in this chapter

$$d\boldsymbol{x}^T d\boldsymbol{x} - d\boldsymbol{X}^T \boldsymbol{X} = d\boldsymbol{X}^T (2\boldsymbol{E}) d\boldsymbol{X}, \tag{5.2}$$

Where $\boldsymbol{E}$ is the Green-Lagrangian (or Green-St. Venant) strain tensor given by:

$$\boldsymbol{E} = \frac{1}{2}(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T + \nabla \boldsymbol{u}^T \nabla \boldsymbol{u}) \tag{5.3}$$

The third nonlinear term in the strain tensor can be ignored by assuming that the displacement gradient is small. From the Hooke's law applied to a continuous media we have:

$$\boldsymbol{\sigma} = \boldsymbol{C}\boldsymbol{E}, \tag{5.4}$$

where $\boldsymbol{\sigma}$ is the stress tensor, and $\boldsymbol{C}$ is a fourth order stiffness tensor. For a general inhomogeneous, anisotropic medium the latter has 21 independent coefficients. Under isotropic assumption for the medium, $\boldsymbol{C}$ can be reduced to two Lamé coefficients of λ and μ, such that:

$$\boldsymbol{\sigma} = \lambda tr(\boldsymbol{E})\boldsymbol{I} + 2\mu\boldsymbol{E}, \tag{5.5}$$

Applying the balance of linear momentum for a non-accelerating segment, we have: [1]:

$$\text{div}\ \boldsymbol{\sigma} + \boldsymbol{f} = 0, \quad \text{in}\ \ \Omega, \tag{5.6}$$

where $\boldsymbol{f}$ is the total external and internal force exerted on the infinitesimal tissue. Using Eq. 5.5 and Eq. 5.6, we derive the linear elasticity equation for an isotropic medium:

$$\text{div}\ (\mu(\nabla\boldsymbol{u} + \nabla\boldsymbol{u}^T)) + \text{div}\ (\lambda\text{div}\ \boldsymbol{u}\boldsymbol{I}) + \boldsymbol{F} = \boldsymbol{0}\ \text{in}\ \ \Omega, \tag{5.7}$$

which is equivalent to:

[1]The velocity of the tumor and the acceleration of surrounding tissues that get deformed on completely negligible in the time scale of tumor growth.

$$\text{div}\,(\mu(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T)) + \nabla(\lambda \text{div}\,\boldsymbol{u}) + \boldsymbol{F} = \boldsymbol{0} \quad \text{in } \Omega, \tag{5.8}$$

We write the elasticity equation in a more compact form as:

$$\boldsymbol{L}\boldsymbol{u} = \boldsymbol{f}, \quad \text{in } \Omega, \tag{5.9}$$

where $\boldsymbol{L}$ is the linear elasticity operator. In our model the forcing term will depend on the tumor concentration. Once we have f, we can solve for u. The change of u in time defines a velocity u_t that we will use to advect the material properties. We describe the overall system in the next section.

5.2 Forward Reaction-Diffusion Model with Mass Effect

Adding the elasticity equation results in the following system of PDEs for the forward tumor problem:

$$c_t - D(\boldsymbol{m})c - R(\boldsymbol{m}, c) + \text{div}[c\boldsymbol{u}_t] = 0 \quad \text{in } \Omega \times (0,1] \tag{5.10a}$$

$$c_0 - \Phi \boldsymbol{p} = 0 \quad \text{in } \Omega \tag{5.10b}$$

$$\boldsymbol{L}\boldsymbol{u} - \boldsymbol{f}(c) = 0 \quad \text{in } \Omega \times (0,1] \tag{5.10c}$$

$$\boldsymbol{m}_{t,i} + \text{div}\,\boldsymbol{u}_t \boldsymbol{m}_i = 0 \quad \text{in } \Omega \times (0,1] \tag{5.10d}$$

$$\boldsymbol{m}_0 - \boldsymbol{m}_0(x) = 0 \quad \text{in } \Omega \tag{5.10e}$$

where c is the tumor concentration, $\boldsymbol{m} \in \mathbb{R}^{N_m}$ is the vector of material properties (White Matter, Grey Matter, etc.) and $\boldsymbol{m}_i$ refers to its i^{th} component and

$\boldsymbol{m}_{t,i}$ refers to the corresponding time derivative, $D(\boldsymbol{m})$ is the diffusion operator, $R(\boldsymbol{m}, c)$ is the reaction operator, $\boldsymbol{u}_t$ is the advection velocity, Ω is the spatial domain, c_0 is the initial distribution of the state variable at $t = 0$, Φ is the basis function for the initial distribution, $\boldsymbol{p} \in \mathbb{R}^{N_p}$ is the corresponding weights, and $\boldsymbol{f}(c)$ is the force function. Note that the diffusion and reaction operators now depend on the material properties that change in time. This is due to the evolution of the tumor growth, which in turn displaces the surrounding tissue. In order to derive the adjoint equations, it will be helpful to write the forward problem in a more compact form by eliminating Eq. 5.10 and substituting it in Eq. 5.10:

$$c_t - D(\boldsymbol{m})c - R(\boldsymbol{m}, c) + \operatorname{div} cL^{-1}\boldsymbol{f}(c)_t = 0 \quad \text{in } \Omega \times (0, 1] \tag{5.11a}$$

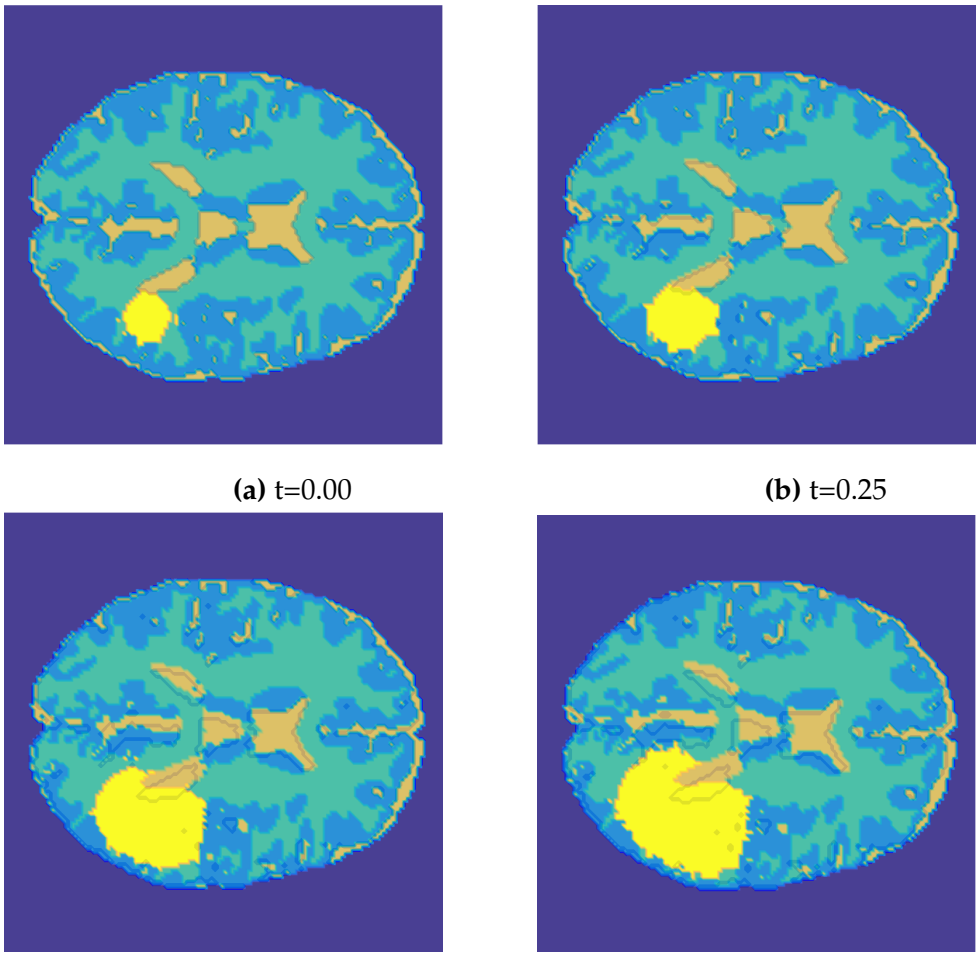
$$c_0 - \Phi\boldsymbol{p} = 0 \quad \text{in } \Omega \tag{5.11b}$$

$$\boldsymbol{m}_{t,i} + \operatorname{div} \boldsymbol{L}^{-1}(\boldsymbol{f}(c)_t)\boldsymbol{m}_i = 0 \quad \text{in } \Omega \times (0, 1] \tag{5.11c}$$

$$\boldsymbol{m}_0 - \boldsymbol{m}_0(x) = 0 \quad \text{in } \Omega \tag{5.11d}$$

An exemplary forward simulation is shown in Fig. 5.1. Here we show the segmentation of the brain at different times overlayyed with the initial CSF contour at time $t = 0$. We used $\boldsymbol{f}(c) = \beta_f tanh(c)\nabla c$, where β_f is a proportionality constant. The choice of the force function is not unique and other formulations are possible as well (81). Here we assume that the force exerted on the brain tissue is proportional to tumor concentration gradient, but in the opposite direction. The addition of the $tanh(c)$ term is to enforce small displacement force where the tumor concentration is small, and vice versa. The elasticity equations are solved with constant

Young modulus of $E = 1000$Pa and $\boldsymbol{u} = 0.45$ [2]. The simulation was performed using the BrainWeb atlas (spatial resolution: $1mm \times 1mm \times 1mm$) (30). The corresponding displacement field, $\boldsymbol{u}$, is shown in Fig. 5.2

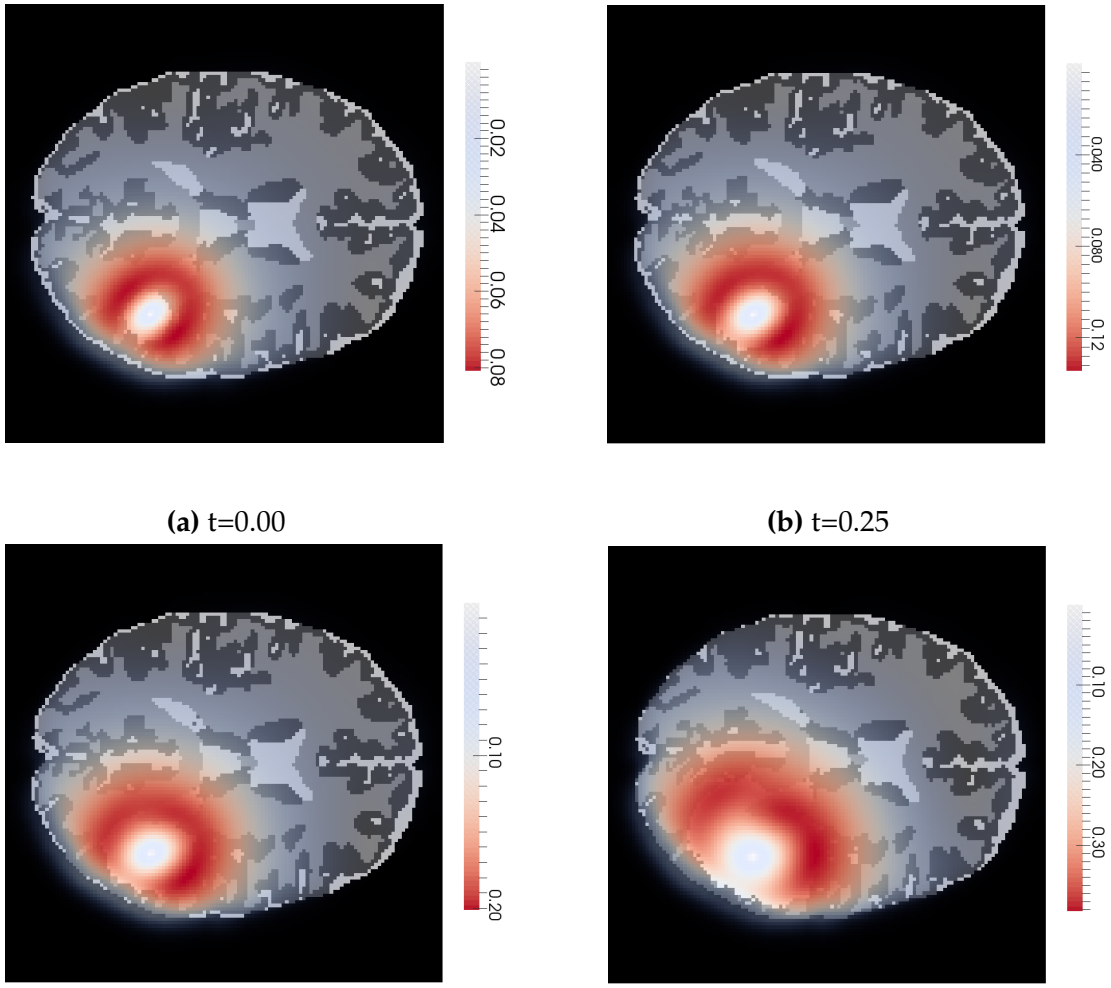


(a) t=0.00 **(b)** t=0.25

(c) t=0.75 **(d)** t=1.00

Figure 5.1: Exemplary forward solves with the mass effect model Eq. 5.11. The segmentation of the image into white matter, grey matter, CSF, and tumor is given for different times, overlayyed with the initial layout of the CSF at time $t = 0$ (gray contours). The segmentation is obtained by assigning the label with maximum probability (60).

[2]Other parameters include $dt = 0.005$, $N_t = 100$, $k = 0.1$, and $\rho = 8$.



(a) t=0.00 **(b)** t=0.25

(c) t=0.75 **(d)** t=1.00

Figure 5.2: The pointwise ℓ_2 norm of the displacement field, $\boldsymbol{u}$, is shown for the four times corresponding to Fig. 5.1.

5.3 Numerical Methods

We solve Eq. 5.11 a pseudo-spectral spatial discretization combined with a operator splitting method for time stepping. The exact numerical equations are given in the Discretize-Then-Optimize section (§5.4.2). In the first part of the time splitting method (5.26a), we solve for the reaction component using the analytical solution to the logistic growth. In the second part we solve the diffusion-advection component (5.26b). We use Crank-Nicolson (CN) for the diffusion terms and backward Euler for the advection. The transport equation for the material properties (5.26c) is solved similarly, using CN to discretize $\boldsymbol{m}_t$ and backwards Euler for the velocity term.

To compute the displacement $\boldsymbol{u}$, we have to solve Eq. 5.10. First, we consider the case of constant Lamé coefficients. Using the identity of $\text{div}\ (\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T) = \Delta \boldsymbol{u} + \nabla \text{div}\ \boldsymbol{u}$, we need to solve the following equation:

$$\boldsymbol{L}\boldsymbol{u} = \mu \Delta \boldsymbol{u} + (\lambda + \mu) \nabla \text{div}\ \boldsymbol{u} = \boldsymbol{f}, \tag{5.12}$$

Given the right hand side $\boldsymbol{f}$, we need to compute $\boldsymbol{u}$. Applying Fourier transform on both sides, we get the symbol of the differential operator:

$$\left(\mu \boldsymbol{\omega}^T \boldsymbol{\omega} \mathbf{I} + (\lambda + \mu) \boldsymbol{\omega} \boldsymbol{\omega}^{\mathbf{T}}\right) \widehat{\boldsymbol{u}} = \widehat{\boldsymbol{f}}, \tag{5.13}$$

where the hats denote the frequency domain, and $\boldsymbol{\omega}$ is the corresponding wave numbers. Afterwards we use the Sherman-Morrison formula to compute $\boldsymbol{u}$:

$$\boldsymbol{u} = \mathcal{F}^{-1}\Big(\big(\frac{1}{\mu \boldsymbol{\omega}^T \boldsymbol{\omega}} - \frac{1}{(\mu \boldsymbol{\omega}^T \boldsymbol{\omega})^2} \frac{(\lambda + \mu) \boldsymbol{\omega} \boldsymbol{\omega}^T}{1 + \frac{\lambda + \mu}{\mu}} \big) \widehat{\boldsymbol{f}} \Big), \tag{5.14}$$

where $\mathcal{F}^{-1}$ is the inverse Fourier transform. To enforce the boundaries we have to explicitly set the displacement to be zero in the boundaries followed by a Gaussian smoothing, since the constant coefficient case does not enforce correct boundary conditions. Note that this model is also very approximate. If we wanted to be more accurate we should have used variable coefficients, or even better unstructured meshes with correct boundary conditions that involve mixed boundary conditions (zero Dirichlet in the normal direction and zero Neumann in the tangential direction). But such an approach would be extremely expensive computationally, since it would require remeshing and significant complications for the adjoint problem. For the variable coefficient case, we use the Generalized Minimum Residual Method (GMRES) to iteratively minimize the residual in the Krylov subspace of Eq. 5.10. The above constant coefficient solver can be used to precondition the varialbe equation to increase the convergence rate. Solving the variable elasticity equation is one of the major bottlenecks of the algorithm. To solve a typical optimization problem we may need to apply the inverse elasticity operator $\mathcal{O}(\mathrm{E}{+}5)$ times. One remedy could be to use the constant coefficient operator in the first few Newton iterations of the inexact Newton solver, and only switch to the variable one when the gradient is small enough.

The computational cost of the forward problem includes the cost of solving the reaction splitting term, the diffusion-advection equation, the transport equation for the material peroperties, and the solution to the elasticity problem. The reaction part has a complexity of $\mathcal{O}(N)$ which is negligible compared to FFT. The implicit diffusion-advection equation has to be solved iteratively using GMRES. Here we report the complexity for solving the problem in 3D, in terms of 1D FFT (by assuming that the size of the problem in x,y, and z direction is the same). For each

time step, we first have to compute the right hand side of 5.26b, which requires 12 FFTs for the diffusion term and $18G_L + 24$ FFTs for the divergence term. Here G_L refers to the number of iterations needed to solve the elasticity equation using GMRES ($G_L = 1$ for the constant coefficient case). We have to solve 5.26b iteratively using GMRES which will take $G_{D,A}$ iterations. For each GMRES iteration we have to perform $18G_L + 24$ FFTs. The forward problem requires N_t time stepping and thus in total we need $((18G_L + 24) \times G_{D,A} + (18G_L + 24)) \times N_t$ FFTs for solving the diffusion-advection equation. Similarly, for solving the transport equation, we need $(6G_m + 6) \times N_t \times N_m$ FFTs. Here G_m is the number of GMRES iterations needed to solve 5.26c, and N_m is the number of material properties. Finally, we have to solve the elasticity equations which has a cost of $18G_L$ per time step. In total, solving the forward problem will have a cost of $((18G_L + 24) \times G_{D,A} + (18G_L + 24) + (6G_m + 6) \times N_m + 18G_L) \times N_t$.

5.4 Inverse Problem

In this section, we present the optimality conditions for the mass effect tumor model. First, we discuss the Optimize-Then-Discretize (OTD) approach and then present Discretize-Then-Optimize (DTO).

5.4.1 Optimize Then Discretize Approach

We seek to minimize the following objective functional:

$$\min_p \mathcal{J} = \frac{1}{2}\|O_1 c_1 - d_1\|_2^2 + \frac{\beta}{2}\|p\|_2^2 \tag{5.15}$$

subject to:

$$\alpha: \quad c_t - D(\boldsymbol{m})c - R(\boldsymbol{m}, c) + \text{div } cL^{-1}f(c)_t = 0 \qquad \text{in } \Omega \times (0,1] \qquad (5.16a)$$

$$\alpha^\dagger: \quad c_0 - \Phi \boldsymbol{p} = 0 \qquad \text{in } \Omega \qquad (5.16b)$$

$$\boldsymbol{\psi}: \quad \boldsymbol{m}_{t,i} + \text{div } \boldsymbol{L}^{-1}f(c)_t \boldsymbol{m}_i = 0 \qquad \text{in } \Omega \times (0,1] \qquad (5.16c)$$

$$\boldsymbol{\psi}^\dagger: \quad \boldsymbol{m}_{0,i} - \boldsymbol{m}_{0,i}(x) = 0 \qquad \text{in } \Omega \qquad (5.16d)$$

Where α, $\alpha^\dagger$, $\boldsymbol{\psi}$, and $\boldsymbol{\psi}^\dagger$ denote the adjoint variables. The corresponding Lagrangian for this problem is as follows:

$$\begin{aligned}\mathcal{L}(c, \boldsymbol{p}, \alpha, \alpha^\dagger, \boldsymbol{\psi}, \boldsymbol{\psi}^\dagger) = \mathcal{J}(c, \boldsymbol{p}) &+ \int_0^1 \int_\Omega \alpha(c_t - Dc - R(c) + \text{div } cL^{-1}f(c)_t)\, d\Omega dt \\ &+ \int_\Omega \alpha^\dagger(c_0 - \Phi \boldsymbol{p}) d\Omega \\ &+ \int_0^1 \int_\Omega \sum_i^{N_m} \boldsymbol{\psi}_i(\boldsymbol{m}_{t,i} + \text{div } \boldsymbol{L}^{-1}f(c)_t \boldsymbol{m}_i)\, d\Omega dt \\ &+ \int_\Omega \sum_i^{N_m} \boldsymbol{\psi}_i^\dagger(\boldsymbol{m}_{0,i} - \boldsymbol{m}_{0,i}(x))\, d\Omega\end{aligned}$$

To obtain the first order optimality conditions we need to find the stationary points of the Lagrangian:

$$
\begin{cases}
\dfrac{\partial \mathcal{L}}{\partial c}, \dfrac{\partial \mathcal{L}}{\partial \boldsymbol{m}} = 0 & \Rightarrow \quad \text{adjoint equations} \\
\dfrac{\partial \mathcal{L}}{\partial \boldsymbol{p}} = 0 & \Rightarrow \quad \text{inversion equation} \\
\dfrac{\partial \mathcal{L}}{\partial \alpha}, \dfrac{\partial \mathcal{L}}{\partial \boldsymbol{\psi}} = 0 & \Rightarrow \quad \text{state equations}
\end{cases}
$$

Therefore, to get the adjoint equation we have:

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial c}\hat{c} = \frac{\partial \mathcal{J}}{\partial c}\hat{c} &+ \int_0^1 \int_\Omega (-\alpha_t - D(\boldsymbol{m})\alpha - \frac{\partial R(\boldsymbol{m}, c)}{\partial c}^T \alpha)\hat{c} d\Omega dt \\
&+ \int_0^1 \int_\Omega (\alpha \text{div}\ \hat{c}\boldsymbol{L}^{-1} f \nabla c_t + \alpha \text{div}\ c\boldsymbol{L}^{-1} f(\hat{c})_t) d\Omega dt \\
&+ \int_\Omega \alpha_1 \hat{c}_1 - \alpha_0 \hat{c}_0 + \alpha^\dagger \hat{c}_0 d\Omega \\
&+ \int_0^1 \int_\Omega (-\sum_i^{N_m} \boldsymbol{\psi}_i \text{div}\ \boldsymbol{L}^{-1} f(\hat{c})_t \boldsymbol{m}_i) d\Omega dt = 0 \quad \forall\ \hat{c} \in \Omega \times [0,1]
\end{aligned}
$$

Integrating by parts:

$$\frac{\partial \mathcal{L}}{\partial c}\hat{c} = \frac{\partial \mathcal{J}}{\partial c}\hat{c} + \int_0^1 \int_\Omega (-\alpha_t - D(\boldsymbol{m})\alpha - \frac{\partial R(\boldsymbol{m},c)}{\partial c}^T \alpha)\hat{c} d\Omega dt$$

$$+ \int_\Omega \alpha_1 \hat{c}_1 - \alpha_0 \hat{c}_0 + \alpha^\dagger \hat{c}_0 d\Omega$$

$$+ \int_0^1 \int_\Omega (-\nabla\alpha \cdot \boldsymbol{L}^{-1} f \nabla c_t + \frac{\partial \mathbf{f}}{\partial c}^T \boldsymbol{L}^{-1}(c\nabla\alpha)_t)\hat{c} d\Omega dt$$

$$+ \int_\Omega \frac{\partial \mathbf{f}}{\partial c}^T \boldsymbol{L}^{-1} c_0 \nabla\alpha_0 \hat{c_0} - \frac{\partial \mathbf{f}}{\partial c}^T \boldsymbol{L}^{-1} c_1 \nabla\alpha_1 \hat{c_1} d\Omega$$

$$+ \int_0^1 \int_\Omega (\frac{\partial \mathbf{f}}{\partial c}^T \boldsymbol{L}^{-1} (\sum_i^{N_m} \nabla\boldsymbol{\psi}_i \boldsymbol{m}_i)_t)\hat{c} d\Omega dt$$

$$+ \int_\Omega \frac{\partial \mathbf{f}}{\partial c}^T \boldsymbol{L}^{-1} \sum_i^{N_m} \nabla\boldsymbol{\psi}_{0,i}\boldsymbol{m}_{0,i}\hat{c_0} - \frac{\partial \mathbf{f}}{\partial c}^T \boldsymbol{L}^{-1} \sum_i^{N_m} \nabla\boldsymbol{\psi}_{1,i}\boldsymbol{m}_{1,i}\hat{c_1} d\Omega = 0 \quad \forall\ \hat{c} \in \Omega \times [0,1]$$

Requiring $\hat{c} \neq 0,\ \hat{c_0} = 0\ \hat{c_1} = 0$:

$$-\alpha_t - D(\boldsymbol{m})\alpha - \frac{\partial R(\boldsymbol{m},c)}{\partial c}^T \alpha - \nabla\alpha \cdot L^{-1} f \nabla c_t + \frac{\partial \mathbf{f}}{\partial c}^T \boldsymbol{L}^{-1}(c\nabla\alpha + \sum_i^{N_m} \nabla\boldsymbol{\psi}_i \boldsymbol{m}_i)_t = 0 \quad \text{in } \Omega \times [0,1) \tag{5.17}$$

Requiring $\hat{c} = 0,\ \hat{c_0} \neq 0\ \hat{c_1} = 0$:

$$\alpha^\dagger - \alpha_0 + \frac{\partial \mathbf{f}}{\partial c}^T \boldsymbol{L}^{-1}(c_0 \nabla\alpha_0 + \sum_i^{N_m} \nabla\boldsymbol{\psi}_{0,i}\boldsymbol{m}_{0,i}) = 0 \tag{5.18}$$

And requiring $\hat{c} = 0,\ \hat{c_0} = 0\ \hat{c_1} \neq 0$:

$$\alpha_1 + O^T(Oc_1 - d_1) - \frac{\partial \mathbf{f}}{\partial c}^T \boldsymbol{L}^{-1}(c_1 \nabla\alpha_1 + \sum_i^{N_m} \nabla\boldsymbol{\psi}_{1,i}\boldsymbol{m}_{1,i}) = 0 \tag{5.19}$$

The second adjoint equation has to be computed with respect to material proper-

ties, $\boldsymbol{m}_i$:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_i}\hat{\boldsymbol{m}}_i = &\int_0^1 \int_\Omega (-\alpha \frac{\partial D(\boldsymbol{m})}{\partial \boldsymbol{m}_i}\hat{\boldsymbol{m}}_i c - \frac{\partial R(\boldsymbol{m},c)}{\partial \boldsymbol{m}_i}^T \alpha \hat{\boldsymbol{m}}_i)\hat{c} d\Omega dt \\ &+ \int_0^1 \int_\Omega ((-\boldsymbol{\psi}_{i,t} - \nabla \boldsymbol{\psi}_i \cdot \boldsymbol{L}^{-1} f(c)_t) \cdot \hat{\boldsymbol{m}}_i) d\Omega dt \\ &+ \int_\Omega \boldsymbol{\psi}_{i,1}\hat{\boldsymbol{m}}_{i,1} - \psi_{i,0}\hat{\boldsymbol{m}}_{i,0} d\Omega \\ &+ \int_\Omega \boldsymbol{\psi}_i^\dagger \hat{\boldsymbol{m}}_{i,0} d\Omega = 0 \quad \forall\ \hat{c} \in \Omega \times [0,1]\end{aligned}$$

Requiring $\hat{\boldsymbol{m}}_i \neq 0,\ \hat{\boldsymbol{m}}_0 = 0\ \hat{\boldsymbol{m}}_1 = 0$:

$$-\boldsymbol{\psi}_{i,t} - \nabla \boldsymbol{\psi}_i \cdot \boldsymbol{L}^{-1} f(c)_t - \alpha \frac{\partial D(\boldsymbol{m})}{\partial \boldsymbol{m}_i} c - \frac{\partial R(\boldsymbol{m},c)}{\partial \boldsymbol{m}_i}^T \alpha = 0 \tag{5.20}$$

Requiring $\hat{\boldsymbol{m}} = 0,\ \hat{\boldsymbol{m}}_0 \neq 0\ \hat{\boldsymbol{m}}_1 = 0$:

$$-\boldsymbol{\psi}_{i,0} + \boldsymbol{\psi}_i^\dagger = 0 \tag{5.21}$$

And requiring $\hat{\boldsymbol{m}} = 0,\ \hat{\boldsymbol{m}}_0 = 0\ \hat{\boldsymbol{m}}_1 \neq 0$:

$$\boldsymbol{\psi}_{i,1} = 0 \tag{5.22}$$

Similarly, the inversion equation can be derived as follows:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{p}}\hat{\boldsymbol{p}} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{p}}\hat{\boldsymbol{p}} + \int_\Omega -\Phi^T \alpha^\dagger \hat{\boldsymbol{p}} d\Omega = 0 \quad \forall\ \hat{\boldsymbol{p}} \in \mathcal{R}^{N_p}$$

By eliminating $\alpha^\dagger$ from Eq. 5.18 we get the reduced gradient:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{p}} = \beta \boldsymbol{p} - \Phi^T \alpha_0 + \Phi^T \frac{\partial \mathbf{f}}{\partial c}^T \boldsymbol{L}^{-1}(c_0 \nabla \alpha_0 + \sum_i^{N_m} \nabla \boldsymbol{\psi}_{0,i} \boldsymbol{m}_{0,i}) \tag{5.23}$$

This results in the following system of PDEs:

$$-\alpha_t - D(\boldsymbol{m})\alpha - \frac{\partial R(\boldsymbol{m}, c)}{\partial c}^T \alpha - \nabla \alpha \cdot L^{-1} f(c)_t + \frac{\partial \mathbf{f}}{\partial c}^T L^{-1}(c \nabla \alpha + \sum_i^{N_m} \nabla \boldsymbol{\psi}_i \boldsymbol{m}_i)_t = 0 \tag{5.24a}$$

$$\alpha_1 + O^T(O c_1 - d_1) - \frac{\partial \mathbf{f}}{\partial c}^T \boldsymbol{L}^{-1}(c_1 \nabla \alpha_1 + \sum_i^{N_m} \nabla \boldsymbol{\psi}_{1,i} \boldsymbol{m}_{1,i}) = 0 \tag{5.24b}$$

$$-\boldsymbol{\psi}_{i,t} - \nabla \boldsymbol{\psi}_i \cdot \boldsymbol{L}^{-1} f(c)_t - \alpha \frac{\partial D(\boldsymbol{m})}{\partial \boldsymbol{m}_i} c - \frac{\partial R(\boldsymbol{m}, c)}{\partial \boldsymbol{m}_i}^T \alpha = 0 \tag{5.24c}$$

$$\boldsymbol{\psi}_{i,1} = 0 \tag{5.24d}$$

$$\beta \boldsymbol{p} - \Phi^T \alpha_0 + \Phi^T \frac{\partial \mathbf{f}}{\partial c}^T \boldsymbol{L}^{-1}(c_0 \nabla \alpha_0 + \sum_i^{N_m} \nabla \boldsymbol{\psi}_{0,i} \boldsymbol{m}_{0,i}) = 0 \tag{5.24e}$$

where $i = 1, \cdots, N_m$ denotes the i^{th} material. To compute the gradient, we have to first solve the forward problem Eq. 5.11 and compute c and $\boldsymbol{m}$. Given c_1 and $\boldsymbol{m}_1$ we need to solve 5.24b iteratively to compute the terminal value for the first adjoint variable α_1. Finally, we have to solve the coupled adjoint equations of 5.24a and 5.24c to get α_0 and $\boldsymbol{\psi}_0$. Substituting these into 5.24e gives us the OTD gradient.

5.4.2 Discretize Then Optimize Approach

In the DTO method, we compute the gradient of the discretized objective functional. This is the exact gradient for the discretized system. Here we seek to minimize:

$$\min_p \mathcal{J} = \frac{1}{2}\|Oc_n - d\|_2^2 + \frac{\beta}{2}\|p\|_2^2 \tag{5.25}$$

subject to:

$$c_0 - \Phi \boldsymbol{p} = 0$$

$$\boldsymbol{m}_0 - \boldsymbol{m}_0(x) = 0$$

$$------------------------------------$$

$$c_1^\star - R_0(\boldsymbol{m}_0, c_0) = 0$$

$$c_1 - c_1^\star - \frac{\Delta t}{2}D(\boldsymbol{m}_0)c_1 - \frac{\Delta t}{2}D(\boldsymbol{m}_0)c_1^\star + \text{div } c_1^\star \boldsymbol{L}^{-1}(f(c_1) - f(c_1^\star)) = 0$$

$$\boldsymbol{m}_{1,i} - \boldsymbol{m}_{0,i} + 0.5\text{div } \boldsymbol{L}^{-1}(f(c_1) - f(c_0))(\boldsymbol{m}_{0,i} + \boldsymbol{m}_{1,i}) = 0$$

$$------------------------------------$$

$$\vdots$$

$$------------------------------------$$

$$c_n^\star - R_{n-1}(\boldsymbol{m}_n, c_{n-1}) = 0 \tag{5.26a}$$

$$c_n - c_n^\star - \frac{\Delta t}{2}D(\boldsymbol{m}_{n-1})c_n - \frac{\Delta t}{2}D(\boldsymbol{m}_{n-1})c_n^\star + \text{div } c_n^\star \boldsymbol{L}^{-1}(f(c_n) - f(c_n^\star)) = 0 \tag{5.26b}$$

$$\boldsymbol{m}_{n,i} - \boldsymbol{m}_{n-1,i} + 0.5\text{div } \boldsymbol{L}^{-1}(f(c_n) - f(c_{n-1}))(\boldsymbol{m}_{n-1,i} + \boldsymbol{m}_{n,i}) = 0, \tag{5.26c}$$

where $\boldsymbol{m}_i$ is the numerical solution for the forward advection equation (Eq. 5.16), $c_i, c_i^\star$ are the intermediate solutions to Eq. 5.16 at time points $t = i\Delta t$, and n denotes the n^{th} time step. Here the dashed line indicate each time step in the forward solve. The corresponding Lagrangian is[3]:

[3]For clarity we are misusing the integral notation instead of a Riemann sum

$$
\begin{aligned}
&\mathcal{L}(c, \boldsymbol{p}, \alpha, \alpha^{\dagger}, \boldsymbol{\psi}, \boldsymbol{\psi}^{\dagger}) = \mathcal{J}(c, \boldsymbol{p}) \\
&+ \int_{\Omega} \alpha_0 (c_0 - \Phi \boldsymbol{p}) d\Omega \\
&+ \int_{\Omega} \boldsymbol{\psi}_0 \cdot (\boldsymbol{m}_0 - \boldsymbol{m}_0(x)) d\Omega \\
&\\
&+ \int_{\Omega} \alpha_1^{\star} (c_1^{\star} - R_0(\boldsymbol{m}_0, c_0)) d\Omega \\
&+ \int_{\Omega} \alpha_1 (c_1 - c_1^{\star} - \frac{\Delta t}{2} D(\boldsymbol{m}_0) c_1 - \frac{\Delta t}{2} D(\boldsymbol{m}_0) c_1^{\star} + \operatorname{div} c_1^{\star} \boldsymbol{L}^{-1} (f(c_1) - f(c_1^{\star}))) d\Omega \\
&+ \int_{\Omega} \sum_{i}^{N_m} \boldsymbol{\psi}_{1,i} (\boldsymbol{m}_{1,i} - \boldsymbol{m}_{0,i} + 0.5 \operatorname{div} \boldsymbol{L}^{-1} (f(c_1) - f(c_0)) (\boldsymbol{m}_{0,i} + \boldsymbol{m}_{1,i})) d\Omega \\
&\\
&\vdots \\
&+ \int_{\Omega} \alpha_n^{\star} (c_n^{\star} - R_{n-1}(\boldsymbol{m}_{n-1}, c_{n-1})) d\Omega \\
&+ \int_{\Omega} \alpha_n (c_n - c_n^{\star} - \frac{\Delta t}{2} D(\boldsymbol{m}_{n-1}) c_n - \frac{\Delta t}{2} D(\boldsymbol{m}_{n-1}) c_n^{\star} + \operatorname{div} c_n^{\star} \boldsymbol{L}^{-1} (f(c_n) - f(c_n^{\star}))) d\Omega \\
&+ \int_{\Omega} \sum_{i}^{N_m} \boldsymbol{\psi}_{n,i} (\boldsymbol{m}_{n,i} - \boldsymbol{m}_{n-1,i} + 0.5 \operatorname{div} \boldsymbol{L}^{-1} (f(c_n) - f(c_{n-1})) (\boldsymbol{m}_{n-1,i} + \boldsymbol{m}_{n,i})) d\Omega
\end{aligned}
$$

Similar to the OTD case we can derive the first order optimality conditions by taking variations w.r.t. adjoint, state and inversion parameters:

$$
\begin{cases}
\dfrac{\partial \mathcal{L}}{\partial c_i}, \dfrac{\partial \mathcal{L}}{\partial c_i^{\star}}, \dfrac{\partial \mathcal{L}}{\partial \boldsymbol{m}_i} = 0 \quad \Rightarrow \quad \text{adjoint equations} \\
\dfrac{\partial \mathcal{L}}{\partial \boldsymbol{p}} = 0 \quad \Rightarrow \quad \text{inversion equation} \\
\dfrac{\partial \mathcal{L}}{\partial \alpha_i}, \dfrac{\partial \mathcal{L}}{\partial \alpha_i^{\star}}, \dfrac{\partial \mathcal{L}}{\partial \boldsymbol{\psi}_{j,i}} = 0 \quad \Rightarrow \quad \text{state equations}
\end{cases}
$$

This leads to the following system of PDEs for the adjoint equation:

$$\alpha_n - \frac{\Delta t}{2} D\alpha_n - \frac{\partial \mathbf{f}(c_3)}{\partial c_3}^T \boldsymbol{L}^{-1} c_n^\star \nabla \alpha_n + O^T(Oc_n - d) = 0$$

$$\boldsymbol{\psi}_{n,i} = 0$$

$$- -$$

$$\alpha_n^\star - \alpha_n - \frac{\Delta t}{2} D(\boldsymbol{m}_2)\alpha_n - \nabla \alpha_n \cdot \boldsymbol{L}^{-1}(f(c_n) - f(c_n^\star)) + \frac{\partial \mathbf{f}(c_3^\star)}{\partial c_3^\star}^T \boldsymbol{L}^{-1} c_n^\star \nabla \alpha_n = 0$$

$$\begin{aligned}
&\boldsymbol{\psi}_{n-1,i} - \boldsymbol{\psi}_{n,i} - 0.5\nabla\boldsymbol{\psi}_{n-1,i} \cdot \boldsymbol{L}^{-1}(f(c_{n-1}) - f(c_{n-2})) - 0.5\nabla\boldsymbol{\psi}_{n,i} \cdot \boldsymbol{L}^{-1}(f(c_n) - f(c_{n-1})) \\
&\qquad + \frac{\Delta t}{2} \nabla \alpha_n \cdot \frac{\mathrm{d}K(\boldsymbol{m}_{n-1}, c_{n-1})}{\mathrm{d}\boldsymbol{m}_{n-1,i}} \nabla(c_n + c_n^\star) - \frac{\mathrm{d}R_{n-1}(\boldsymbol{m}_{n-1}, c_{n-1})}{\mathrm{d}\boldsymbol{m}_{n-1,i}}^T \alpha_n^\star = 0 \\
&\alpha_{n-1} - \frac{\mathrm{d}R_{n-1}(\boldsymbol{m}_{n-1}, c_{n-1})}{\mathrm{d}c_{n-1}}^T \alpha_n^\star - \frac{\Delta t}{2} D(\boldsymbol{m}_{n-2})\alpha_{n-1} - \frac{\partial \mathbf{f}(c_2)}{\partial c_2}^T \boldsymbol{L}^{-1} c_{n-1}^\star \nabla \alpha_{n-1} \\
&\qquad - 0.5 \sum_i \frac{\partial \mathbf{f}(c_2)}{\partial c_2}^T \boldsymbol{L}^{-1}(\boldsymbol{m}_{n-2,i} + \boldsymbol{m}_{n-1,i}) \nabla \boldsymbol{\psi}_{n-1,i} \\
&\qquad + 0.5 \sum_i \frac{\partial \mathbf{f}(c_2)}{\partial c_2}^T \boldsymbol{L}^{-1}(\boldsymbol{m}_{n-1,i} + \boldsymbol{m}_{n,i}) \nabla \boldsymbol{\psi}_{n,i} = 0
\end{aligned}$$

$$- -$$

$$\vdots$$

$$- -$$

$$\alpha_1^\star - \alpha_1 - \frac{\Delta t}{2} D(\boldsymbol{m}_0)\alpha_1 - \nabla \alpha_1 \cdot \boldsymbol{L}^{-1}(f(c_1) - f(c_1^\star)) + \frac{\partial \mathbf{f}(c_1^\star)}{\partial c_1^\star}^T \boldsymbol{L}^{-1} c_1^\star \nabla \alpha_1 = 0$$

$$\begin{aligned}
&\boldsymbol{\psi}_{0,i} - \boldsymbol{\psi}_{1,i} - 0.5\nabla\boldsymbol{\psi}_{1,i} \cdot \boldsymbol{L}^{-1}(f(c_1) - f(c_0)) \\
&\qquad - \frac{\mathrm{d}R_0(\boldsymbol{m}_0, c_0)}{\mathrm{d}\boldsymbol{m}_{0,i}}^T \alpha_1^\star + \frac{\Delta t}{2} \nabla \alpha_1 \cdot \frac{\mathrm{d}K(\boldsymbol{m}_0, c_0)}{\mathrm{d}\boldsymbol{m}_{0,i}} \nabla(c_1 + c_1^\star) = 0 \\
&\alpha_0 - \frac{\mathrm{d}R_0(\boldsymbol{m}_0, c_0)}{\mathrm{d}c_0}^T \alpha_1^\star + 0.5 \frac{\partial \mathbf{f}(c_0)}{\partial c_0}^T \boldsymbol{L}^{-1} \sum_i (\boldsymbol{m}_{0,i} + \boldsymbol{m}_{1,i)} \nabla \psi_{1,i} = 0
\end{aligned}$$

Here $i = 0, \cdots, N_m$ refers to i^{th} material type (i.e. white matter, grey matter, etc.).

Similarly, the DTO's inversion equation is:

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{p}} = \beta \boldsymbol{p} - \Phi^T \alpha_0 \quad \text{in } \mathcal{R}^{N_p}$$

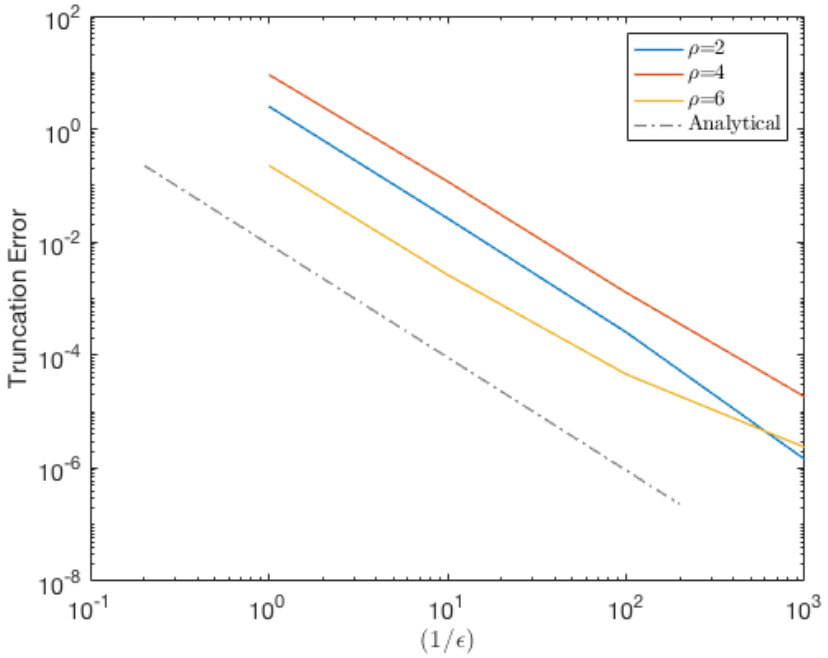


Figure 5.3: The gradient convergence test. We numerically compute the truncation error given by the gradient and compare it with the expected second-order analytical result (dotted line). As one can see, the numerical gradient gives the convergence rate up to the single precision accuracy of the forward solve.

To compute the gradient at a point given by the set of parameters $\boldsymbol{p}$, we should first solve the forward problem (Eq. 5.26) to compute $c, c^\star$, and $\boldsymbol{m}$. Given these values we have to solve the adjoint equation (Eq. 5.4.2) to compute $\alpha, \alpha^\star$ and $\boldsymbol{\psi}$. Substituting the results into Eq. 5.4.2 gives us the gradient. We test the gradient by studying the convergence rate of the Taylor series expansion of the objective functional defined in Eq. 5.25, around $\boldsymbol{p}$:

$$\mathcal{J}(\boldsymbol{p}+\epsilon\tilde{\boldsymbol{p}}) = \mathcal{J}(\boldsymbol{p}) + \epsilon\frac{\partial\mathcal{J}}{\partial\boldsymbol{p}}\tilde{p} + \mathcal{O}(\epsilon^2), \tag{5.27}$$

where $\tilde{\boldsymbol{p}}$ is a random variation in the parameters, and ϵ is a scalar. The gradient correction should result in second-order truncation error. We compute the left hand side and the right hand side and compute the truncation order numerically. The results are shown in Fig. 5.3 for different values of ρ.

5.5 Conclusions

In this chapter we discussed how the reaction-diffusion tumor model should be modified to capture mass-effect, an important characteristic of high grade gliomas. Even though the mass-effect can capture more visible characteristics of the gliomas, but it adds more patient specific unknowns, which themselves have to be approximated through inverse problem. These additional degrees of freedom can easily cause overfitting, if there is not enough data. This is a major limitation of using more complex tumors. However, we discussed how the inverse problem algorithm discussed in chapter 2, has to be changed in order to solve for the parameters of interest, assuming that those unknowns are available. We derived the first order optimality conditions for the Optimize-Then-Discretize(OTD) and Discretize-Then-Optimize (DTO) approach and tested the accuracy of the gradient by studying its convergence rate.

Chapter 6

Parallel Spectral Operators: Accelerated FFT Library

In this chapter we discuss how we evaluate the spectral operators in parallel using our in-house Accelerated FFT (AccFFT) library. There are two phases in evaluating a spectral operator: forward/backward FFTs in parallel and Hadamard products in the frequency domain. The operation is limited by the FFTs as the Hadamard product can be done locally and has a linear complexity. In this chapter we will discuss the design of AccFFT library and its performance optimizations that can provide considerable speedups for computing spectral operators such as gradient and divergence.

AccFFT is not specific to image analysis and it is now being used by many groups for other applications. It has support for parallel 3D FFT in single/double precision for CPU and GPUs as well as optimized kernels for many spectral operators used in practice.

6.1 Literature Review and Background

Fast Fourier Transform is one of the most fundamental algorithms in computational science and engineering. It is used in turbulence simulations (97), computational chemistry and biology (37), gravitational interactions (2), cardiac electrophysiology (21), cardiac mechanics (115), acoustic, seismic and electromagnetic

This chapter is based on existing publication: *A. Gholami, J. Hill, D. Malhotra, and G. Biros. AccFFT: A library for distributed-memory FFT on CPU and GPU architectures, arxiv:1506.07933*
The author designed the library and implemented the algorithms. Dhairya helped with the GPU kernels. Judith helped with benchmarking on Titan system. Prof. Biros aided in editing the paper and supervised the work.

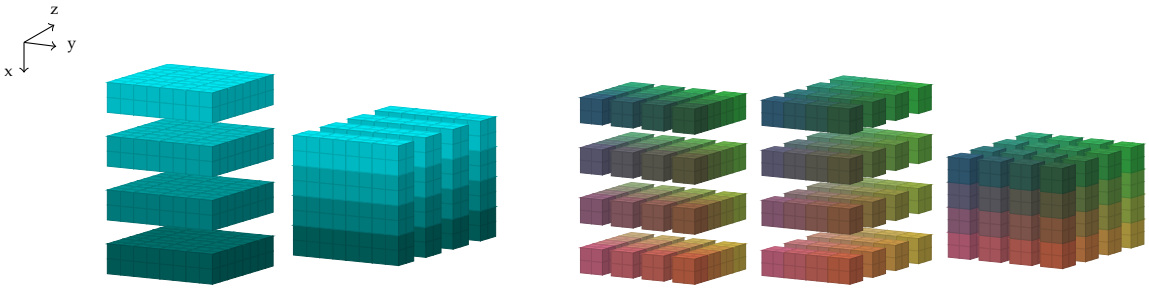


Figure 6.1: Decomposition of the input/output array in different stages of the forward FFT algorithm. Left:slab decomposition. Right: pencil decomposition

scattering (20, 137), materials science (119), molecular docking (91) and many other areas.

Due to its wide range of applications and the need for scalability and performance, the design of FFT algorithms remains an active area of research. Highly optimized single-node FFT algorithms have been implemented by all major hardware vendors, including Intel's MKL library (182), IBM's ESSL library (45), NVIDIA's CUFFT (129) library, and the new AMD's clFFT library (1). A thorough review of the key challenges to get good performance from single node FFT implementations can be found in (48). In the realm of open-source software, one of the most widely used libraries is the FFTW (50, 51). Getting good single-node performance from FFT that works optimally for all platforms is challenging. Therefore, libraries such as FFTW or SPIRAL use auto-tuning or search and learn techniques to find an optimal algorithm for a given platform (143). Single-node implementations of these libraries have been extended to distributed memory versions either by the original developers or by other research groups. A large number of distributed memory libraries is currently available for CPUs.

Related work

There is a vast literature on algorithms for FFTs. Our discussion is by no means exhaustive. We limit it on the work that is most closely related to ours. Introductory material on distributed memory FFT can be found in (63). Excellent discussions on complexity and performance analysis for 3-D FFTs can be found in (53) and (33).

• (*Libraries for CPU architectures.*) One of the most widely used packages for FFTs is the FFTW (50) library. FFTW supports MPI using slab decomposition and hybrid parallelism using OpenMP. However, the scalability of slab decompositions is limited. Furthermore, FFTW does not support GPUs. P3DFFT (135) extends the single-node FFTW (or ESSL) and supports both slab and pencil decompositions. Another recent library is PFFT (140). The library is built on top of FFTW and uses its transpose functions to perform the communication phase. It has recently been extended to nonequispaced FFTs (141). It supports distributed multidimensional FFTs, as well as features such as ghost points or pruned transforms. PFFT has an auto-tuning function for finding an optimal communication pattern. A very similar code to P3DFFT and PFFT is 2DECOMP (99) and OpenFFT (38). These are all very well written libraries that have been used extensively.

A multithreaded code (not open source) is described in (97) in the context of turbulence simulations. This code is based on FFTW and employs single-node optimizations. To our knowledge, this code is one of the most scalable 3-D FFTs. The authors report results on up to 786,432 cores on an IBM Blue Gene machine. However, the authors observe lack of scalability of the transpose for large core counts. On Stampede they start losing scalability at 4,096 nodes. In (158) the authors propose pencil decomposition optimizations that deliver 1.8× speed-up over

FFTW. The main idea is the use of non-blocking MPI all-to-all operations that allow overlapping computation and communication. However the method does not address the scalability issues of FFTs. The authors compare FFTW, P3DFFT and 2DECOMP with their scheme. Other works that study 3-D FFTs on x86 platforms include (7, 132).

Algorithm 1: Forward and backward FFT algorithm for pencil decomposition.

Input : Data in spatial domain.
Layout: $N_0/P_0 \times N_1/P_1 \times N_2$
Output: Data in frequency domain.
Layout: $\widehat{N}_0 \times \widehat{N}_1/P_0 \times \widehat{N}_2/P_1$

$N_0/P_0 \times N_1/P_1 \times \widehat{N}_2 \xleftarrow{FFT} N_0/P_0 \times N_1/P_1 \times N_2$;
$N_0/P_0 \times N_1 \times \widehat{N}_2/P_1 \xleftarrow{T} N_0/P_0 \times N_1/P_1 \times \widehat{N}_2$;
$N_0/P_0 \times \widehat{N}_1 \times \widehat{N}_2/P_1 \xleftarrow{FFT} N_0/P_0 \times N_1 \times \widehat{N}_2/P_1$;
$N_0 \times \widehat{N}_1/P_0 \times \widehat{N}_2/P_1 \xleftarrow{T} N_0/P_0 \times \widehat{N}_1 \times \widehat{N}_2/P_1$;
$\widehat{N}_0 \times \widehat{N}_1/P_0 \times \widehat{N}_2/P_1 \xleftarrow{FFT} N_0 \times \widehat{N}_1/P_0 \times \widehat{N}_2/P_1$;

Input : Data in frequency domain.
Layout: $\widehat{N}_0 \times \widehat{N}_1/P_0 \times \widehat{N}_2/P_1$
Output: Data in spatial domain.
Layout: $N_0/P_0 \times N_1/P_1 \times N_2$

$N_0 \times \widehat{N}_1/P_0 \times \widehat{N}_2/P_1 \xleftarrow{IFFT} \widehat{N}_0 \times \widehat{N}_1/P_0 \times \widehat{N}_2/P_1$;
$N_0/P_0 \times \widehat{N}_1 \times \widehat{N}_2/P_1 \xleftarrow{T} N_0 \times \widehat{N}_1/P_0 \times \widehat{N}_2/P_1$;
$N_0/P_0 \times N_1 \times \widehat{N}_2/P_1 \xleftarrow{IFFT} N_0/P_0 \times \widehat{N}_1 \times \widehat{N}_2/P_1$;
$N_0/P_0 \times N_1/P_1 \times \widehat{N}_2 \xleftarrow{T} N_0/P_0 \times N_1 \times \widehat{N}_2/P_1$;
$N_0/P_0 \times N_1/P_1 \times N_2 \xleftarrow{IFFT} N_0/P_0 \times N_1/P_1 \times \widehat{N}_2$;

• (*Libraries for distributed-memory GPUs.*) The work presented in (128) is, to our knowledge, one the most efficient and more scalable distributed GPU implementations. It only supports slab decomposition so it cannot be scaled to large core counts. The scaling results presented in the paper are up to 768 GPUs. The authors employ special techniques to improve the complexity of the transpose and use an in-house CUDA FFT implementation. Their optimizations are specific to

the infiniband-interconnect using the IBverbs library and thus is not portable. In the largest run, they observed 4.8TFLOPS for a 2048^3 problem on 786 M2050 Fermi GPUs (double precision, complex-to-complex), which is roughly 1.2% of the peak performance.

The ohter recent GPU library is digpuFFT (33) which is a modification of P3DFFT in which the intranode computations are replaced by CUFFT. They achieve about 0.7% of the peak. However, digpuFFT code was not designed for production but for experimental validation of the theoretical analysis of the complexity of 3-D FFTs, and its implications to the design of exascale architectures.

• (*1D FFT and single-node libraries.*) Other works that analyze scalability of the FFT codes include the FFTE code (173), which is part of the HPCC benchmark. It includes several optimizations but has support for GPU only via PGI compiler directives. In (64, 186), the authors propose schemes for single node 3-D FFTs. In (125), shared-memory multiple GPU algorithms are discussed. More specialized and somewhat machine-dependent codes are (87) and (49).

A very interesting set of papers proposes a different FFT algorithm that has lower global communication constants. It requires one all-to-all communications as opposed to three, and can be made up to 2× faster by introducing an approximation error in the numerical calculations. The algorithm was introduced in (174) and its parallel implementation discussed in (134). Now it is part of the MKL library. It currently supports 1D FFT transforms only.

Contributions

We present AccFFT, a library that given an $N_0 \times N_1 \times N_2 \times \cdots$ matrix of values computes its Fourier Transform. The library supports the following features:

Algorithm 2: Forward and backward FFT algorithms for general $d-1$ dimensional decomposition.

Input : Data in spatial domain.
Layout: $N_0/P_0 \times \cdots N_{d-1}/P_{d-1} \times N_d$
Output: Data in frequency domain.
Layout: $\widehat{N}_0 \times \cdots \widehat{N}_{d-1}/P_{d-2} \times \widehat{N}_d/P_{d-1}$
$h = N_0/P_0 \times \cdots \times N_{d-1}/P_{d-1}$;
$h' = 1$;
for $i = d, ..., 1$ **do**
  $h \times \widehat{N}_i \times h' \xleftarrow{FFT} h \times N_i \times h'$;
  $H = h/(N_{i-1}/P_{i-1})$;
  $H' = h' * (\widehat{N}_i/P_{i-1})$;
  $H \times N_{i-1} \times H' \xleftarrow{T} h \times \widehat{N}_i \times h'$;
  $h = H$; $h' = H'$;
$h \times \widehat{N}_0 \times h' \xleftarrow{FFT} h \times N_0 \times h'$;
Input : Data in frequency domain.
Layout: $\widehat{N}_0 \times \cdots \widehat{N}_{d-1}/P_{d-2} \times \widehat{N}_d/P_{d-1}$
Output: Data in spatial domain.
Layout: $N_0/P_0 \times \cdots N_{d-1}/P_{d-1} \times N_d$
$h = 1$;
$h' = \widehat{N}_1/P_0 \times \cdots \times \widehat{N}_d/P_{d-1}$;
for $i = 0, ..., d-1$ **do**
  $h \times N_i \times h' \xleftarrow{IFFT} h \times \widehat{N}_i \times h'$;
  $H = h * (N_i/P_i)$;
  $H' = h'/(\widehat{N}_{i+1}/P_i)$;
  $H \times \widehat{N}_{i+1} \times H' \xleftarrow{T} h \times \widehat{N}_i \times h'$;
  $h = H$; $h' = H'$;
$h \times N_d \times h' \xleftarrow{IFFT} h \times \widehat{N}_d \times h'$;

- hybrid MPI and CUDA parallelism,
- An overlapping all-to-all that reduces the PCIe overhead
- slab (1D) and pencil (2D) decomposition, and
- support for real-to-complex (R2C), complex-to-complex (C2C), and complex-to-real (C2R) transforms
- Single and double precision support
- Fast spectral operators that can provide significant speedup

AccFFT uses CUFFT and FFTW for the FFT computations. It extends the single-node version of these two libraries to pencil decomposition for distributed memory FFTs. Both decompositions are necessary in order to ensure good performance across a range of MPI ranks. Slab decomposition limits the number of MPI ranks, P, to be less or equal to $N_0 = \max\{N_0, N_1, N_2\}$, and thus does not scale as well as the pencil decomposition [1]. To the best of our knowledge, AccFFT is the only open source code for distributed GPU transforms. The only other library that is currently maintained is FFTE, which has very limited features. It only supports complex-to-complex transforms and requires commercial PGI compiler. However, our code is open source and does not require such compilers. Furthermore, AccFFT supports more transforms and our experimental comparisons show that our GPU code is faster than FFTE.

We present scaling results on Maverick (with K40 GPUs) at TACC, and on Titan at ORNL (with K20 GPUs). We use overlapping communication algorithms for GPUs to hide the overhead of moving data forth and back from the CPU. The

[1] Typical values for N_0 range from 100s to 10,000s

Algorithm 3: Forward and backward FFT algorithm for slab decomposition.

Input : Data in spatial domain.
Layout: $N_0/P \times N_1 \times N_2$
Output: Data in frequency domain.
Layout: $\widehat{N}_0 \times \widehat{N}_1/P \times \widehat{N}_2$
$N_0/P \times \widehat{N}_1 \times \widehat{N}_2 \xleftarrow{FFT} N_0/P \times N_1 \times N_2$;
$N_0 \times \widehat{N}_1/P \times \widehat{N}_2 \xleftarrow{T} N_0/P \times \widehat{N}_1 \times \widehat{N}_2$;
$\widehat{N}_0 \times \widehat{N}_1/P \times \widehat{N}_2 \xleftarrow{FFT} N_0 \times \widehat{N}_1/P \times \widehat{N}_2$;
Input : Data in frequency domain.
Layout: $\widehat{N}_0 \times \widehat{N}_1/P_0 \times \widehat{N}_2$
Output: Data in spatial domain.
Layout: $N_0/P_0 \times N_1 \times N_2$
$N_0 \times \widehat{N}_1/P_0 \times \widehat{N}_2 \xleftarrow{IFFT} \widehat{N}_0 \times \widehat{N}_1/P_0 \times \widehat{N}_2$;
$N_0/P_0 \times \widehat{N}_1 \times \widehat{N}_2 \xleftarrow{T} N_0 \times \widehat{N}_1/P_0 \times \widehat{N}_2$;
$N_0/P_0 \times N_1 \times N_2 \xleftarrow{IFFT} N_0/P_0 \times \widehat{N}_1 \times \widehat{N}_2$;

library is open source and available for download (56) under GNU GPL version 2 license.

Limitations

There are several limitations in our library. We are not using non-blocking collective communications. The authors of (158) demonstrated that such an asynchronous approach can yield further speedups. Also Currently we do not support pruned FFTs or additional features such as Chebyshev approximations. Our implementation does not support inexact FFTs. Currently there is no support for hybrid floating point computation, but for larger FFTs it may be necessary.

Outline of this chapter

In Section 6.2, we summarize our algorithms for CPU and GPU platforms, and discuss details of our optimizations. In Section 6.3, we will discuss the algorithm for evaluating spectral operators and the related optimization. In Section 6.4,

we present results of the numerical experiments.

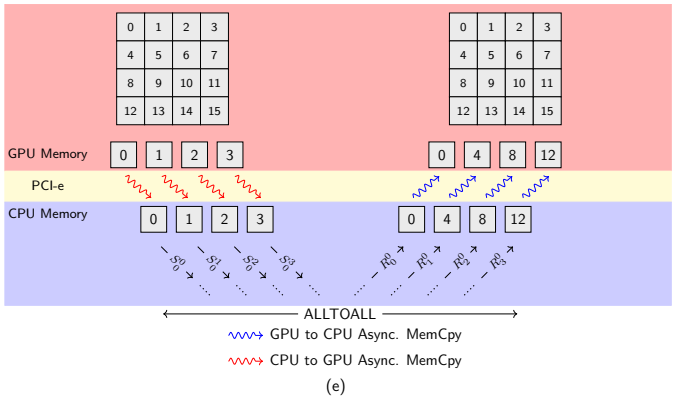


Figure 6.2: Demonstration of the GPU all-to-all. We interleave PCI-e transfers with send/recv operations to hide the overhead of sending data back/forth from the CPU. Here we are showing the alltoall process for the case of $P = 4$. Each element in the 4×4 matrix, represents a super element in the memory. S_i^j denotes a send directive to process j from process i. Similarly, R_i^j denotes receive message at process j sent from process i.

6.2 Algorithm

First let us introduce some basic notation: f is input array, $\widehat{f}$ is its Fourier transform, P is the total number of MPI tasks, N_0, N_1, N_2 denotes the size of f in x,y, and z direction, and $N = N_0 \times N_1 \times N_2$.

The discrete 3-D Fourier transform corresponds to a dense matrix-vector multiplication. However, the computational complexity can be reduced by using Cooley-Tukey algorithm to:

$$5N_0N_1N_2(\log(N_0N_1N_2)).$$

The forward FFT maps space to frequency domain and the inverse FFT maps the frequency to space domain. The algorithms are the same up to a scaling factor, and

have the same computational complexity[2].

For many scientific applications, f does not fit into a single node and therefore the data needs to be distributed across several nodes. Two such distributions for a 3D array are the slab decomposition, in which the data is distributed in slabs and the pencil decomposition where each task gets a pencil of the data, as shown in Figure 6.1. To compute the FFT, each task has to compute its portion of FFTs, and then exchange data with other tasks. One can either use a binary exchange or a transpose (all-to-all) algorithm (63). In this chapter we focus on the latter.

First we discuss the slab-decomposition, which is outlined in Algorithm 3. The input data is distributed in the first dimension over P tasks, i.e. $N_0/P \times N_1 \times N_2$, which is referred to as a slab. Without loss of generality, let us assume that $N_0 = \max\{N_0, N_1, N_2\}$. In the limit of $P = N_0$, each task will just get a 2D slice of f locally. If $P > N_0$ the slab decomposition cannot be used. In the forward algorithm, each task computes a N_0/P-batch of 2-D FFTs, each one having a size of $N_1 \times N_2$. Then an all-to-all exchange takes place to redistribute the data (indicated by the second step marked as T in Algorithm 3). After this each task gets a slab of size $N_0 \times \widehat{N}_1/P \times \widehat{N}_2$, where the hats denote that the Fourier Transform has been computed across the last two dimensions. To complete the transform, each task can then compute a batch of $\widehat{N}_1/P \times \widehat{N}_2$ 1-D FFTs of length N_0. The inverse FFT can be computed in a similar fashion by reversing these steps. One advantage of our implementation is that the memory layout of the data in frequency space is the same as in the spatial one. This is different from, e.g., PFFT or FFTW's implementation, where the default (and faster) option, changes the memory layout from xyz to yxz. They

[2]Note that typically FFT libraries do not apply scaling when forward FFT is computed and instead a full normalization is done when inverse FFT is performed. Therefore the complexity of the inverse would be slightly different than forward FFT, but the asymptotic behaviour would be the same.

provide an option which brings back the memory to the original xyz format, but at the cost of a local transpose [3].

Slab decomposition can be modified by decomposing the second dimension as well, which is known as pencil decomposition (Algorithm 1). In this approach each task gets a batch of 1-D pencils, local in the last dimension, with a data layout of $N_0/P_0 \times N_1/P_1 \times N_2$. The MPI tasks are mapped to a 2D matrix with P_0 rows and P_1 columns such that $P = P_0 \times P_1$. To compute the forward FFT, each task first computes a $N_0/P_0 \times N_1/P_1$ batch of 1-D FFTs of length N_2. This is followed by a block row-wise all-to-all communication step in which all the tasks in the same row call exchange to collect the second dimension of the array locally. In this step, one needs to redistribute a batch of N_0/P_0 matrices of size $N_1/P_1 \times N_2$. A naive implementation of this phase would lead to costly cache misses. As a result, we first perform a local packing of the data, so that all the batched noncontiguous data, are grouped together in a contiguous buffer. Then the all-to-all operation is performed, followed by an unpacking operation to get the data back into its correct format. Another approach is to use MPI data types to exchange non contiguous data. However, the latter would depend on how well the MPI compiler handles non-contiguous data. For consistency we do the packing and unpacking before the all-to-all calls.

After the first all-to-all exchange, each task computes a batched 1-D FFT of size N_1 of its local data, which is now in the form of $N_0/P_0 \times N_1 \times \widehat{N}_2/P_1$. This is followed by an the all-to-all operation performed on a $N_0/P_0 \times N_1$ matrix with super-elements of size $\widehat{N}_2/P_1$ complex numbers. In this step, the data is indeed

[3]Note that this is different from global transposes. Here we are referring to the case where the global data layout is transposed. That is after passing the equivalent of TRANSPOSED_OUT flag for each library, by default they change the memory footprint of the data in frequency domain to yxz instead of xyz.

contiguous and no packing/unpacking is required. However, we do perform a local transpose, to change the memory layout from $N_1/P_0 \times N_0 \times N_2/P_1$ to $N_0 \times N_1/P_0 \times N_2/P_1$. This is done to have a consistent memory access pattern in both the spatial and frequency domain, which is desirable from a user's standpoint. To complete the forward FFT, a final batched 1-D FFT of length N_0 should be computed after this step. Now in the frequency space, each task owns the x pencil locally, while in spatial domain it owns the z dimension locally. This can be changed by performing two additional all-to-all exchanges, but it is typically avoided as it can be done while the inverse FFT is being computed. The pencil decomposition algorithm can be extended to support n-dimensional tensors as shown in Algorithm 2.

It is well known that the most expensive part of distributed FFT is the communication phase (33), which adversely affects the scaling at large core counts. This has been verified in large scale runs on Stampede and Blue Waters (97). This phase involves all-to-all exchanges, which is essentially transpose operation between a subgroup of tasks. As mentioned earlier, this exchange should be wrapped around a packing/unpacking phase to make the data contiguous in the memory. This phase can be performed either by reshuffling of the data as done in P3DFFT, a local transpose as implemented in FFTW library, or eliminated by using MPI Data types (152). However, the communication time dominates the cost of distributed FFT. The situation is even worse for the GPU, since the data has to be transferred forth and back to the CPU through PCIe, which is as expensive as the communication phase.

Recently, NVIDIA has introduced GPUDirect technology where GPUs on different nodes can communicate directly through the PCIe bus and avoid the CPU altogether. However, this feature requires special hardware support as well

as a compatible OFED (OpenFabrics Enterprise Distribution). In the absence of GPUDirect, one option is to perform a blocking memcpy from GPU to CPU, use the transpose functions that are already implemented in the CPU code, and then copy back the results to the GPU. The packing and unpacking phases can still be performed on the GPU, as it can perform the reshuffling/local transposes much faster than on the CPU.

However, it is possible to hide the extra cost of memcpy by interleaving it into send and receive operations. Instead of copying all the data at once and then sending it, we divide the memcpy into chunks of the same size that each process has to send to other tasks. Each chunk is copied to a CPU buffer at a time, followed by an asynchronous send instruction. In this manner, the communication part of the CPU can start while the rest of the chunks are being copied. Since we post the asynchronous receive instructions beforehand, the receive operation can also happen with the device to host memcpy. Each received chunk can then be copied asynchronously back to the GPU Fig. 6.2.

For local FFT computations on GPUs, we use the CUFFT library from NVIDIA. One development was to implement a local transpose since the transpose on the NVIDIA's SDK libraries is not appropriate for the 3-D (or higher dimensional) FFTs since it doesn't support the correct stride and n_tuples. The second and the main contribution was to work around the limited bandwidth between the host CPU and GPU, and hide its overhead.

Complexity Analysis

The communication cost is $\mathcal{O}(\frac{N}{\sigma(p)})$, where $\sigma(p)$ is the bisection bandwidth of the network (for a hypercube it is $p/2$ (135)). The total execution time for an FFT

of size N on a hypercube can be approximated by:

$$T_{\mathrm{FFT}} = \mathcal{O}\left(\frac{N \log N}{P}\right) + \mathcal{O}\left(\frac{N}{P}\right).$$

The first term represents the computation and the second the memory and communication costs. For a 3-D torus topology (such as the one used on Titan) the complexity becomes:

$$T_{\mathrm{FFT}} = \mathcal{O}\left(\frac{N \log N}{P}\right) + \mathcal{O}\left(\frac{N}{P^{2/3}}\right).$$

For the GPU version this should also include the device-host communication costs. In (33) the authors give a detailed analysis in which cache effects and the local and remote memory bandwidth for GPUs and CPUs is taken into account. The basic point is that in strong scaling, the computation part becomes negligible and the overall wall-clock time will be dominated by the communication costs.

6.3 Fast Spectral Operators

Spatial differential operators such as gradient, divergence, Laplace, etc. can be computed by first transforming the input field into the frequency domain (FFT), followed by a Hadamard transform and an inverse FFT. Exemplarily, we consider computing the x-derivative of a scalar field f:

$$f_x = \mathcal{F}^{-1}(-i\omega_x \mathcal{F}(f)), \tag{6.1}$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote 3D forward/backward FFTs. However, the extra FFTs in y and z directions are unnecessary as the derivative can be equivalently computed

as:

$$f_x = \mathcal{F}_x^{-1}(-i\omega_x \mathcal{F}_x(f)), \tag{6.2}$$

where $\mathcal{F}_x$ denotes the FFT transform in x direction. Note that, here we only need a batched 1D forward/inverse FFT instead of a 3D FFT and only one global transpose to establish the pencil decomposition in x direction. This saves a large amount of communication since no repartitioning of data for the pencil decompositions in y direction is required. After the transpose is performed, we compute the forward FFT transform in x-direction, followed by a Hadamard product, a local inverse FFT, and another global transpose to redistribute the data.

Another commonly used Parallel FFT library (P3DFFT (135)), allow the user to specify if a local forward/inverse FFT is needed only in the Z direction and they currently do not support custom global transposes which limits us to use Eq. 6.1. We have enhanced AccFFT and extended the global transpose support so that we can use Eq. 6.2 when appropriate. For the x-derivative this reduces 4 global transposes to just 2 (Algorithm 4). Overall this reduces the total global transposes to 4 for the gradient/divergence operators as opposed to 8, which reduces the communication volume by a factor of 2. Moreover, we avoid unnecessary local FFTs which leads to further improvement.

The library currently supports the following type of spectral transforms in single/double precision for both CPU and GPU:

- Gradient (∇)
- Divergence ($\nabla\cdot$)
- Laplace (Δ) and inverse Laplace (Δ^{-1})

- Biharmonic (Δ^2), and inverse Biharmonic (Δ^{-2})

Algorithm 4: Fast algorithm for computing x derivative, which only requires two global transposes as opposed to four.

Input : Data in spatial domain.
Layout: $N_0/P_0 \times N_1/P_1 \times N_2$
Output: x derivative
Layout: $\widehat{N}_0/P_0 \times N_1/P_1 \times N_2$

$N_0 \times N_1/P_0 \times N_2/P_1 \xleftarrow{T} N_0/P_0 \times N_1/P_1 \times N_2$; // input data

$\widehat{N}_0 \times N_1/P_0 \times N_2/P_1 \xleftarrow{FFT} N_0 \times N_1/P_0 \times N_2/P_1$; // Hadamard and FFT

$\widehat{N}_0/P_0 \times N_1 \times N_2/P_1 \xleftarrow{T} \widehat{N}_0 \times N_1/P_0 \times N_2/P_1$; // x derivative

6.4 Numerical experiments

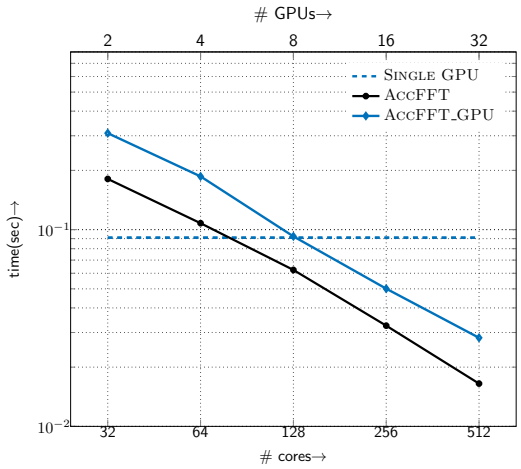
In this section we report the performance of AccFFT and give details regarding our implementation and the different problem sizes used for evaluating the library.

Computing Platforms

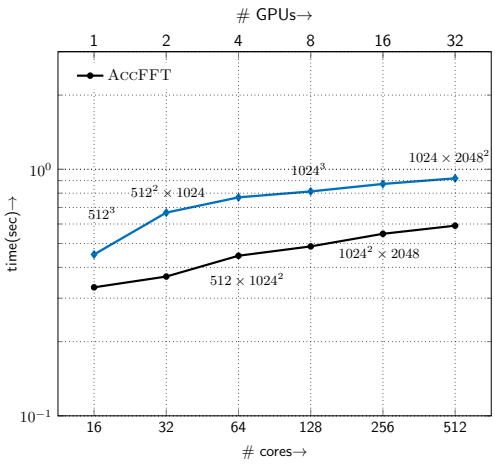
The tests are performed on the following platforms:

- The **Maverick** system at TACC is a Linux cluster with 132 compute nodes, each with dual 10-core 2.8GHz Intel Xeon E5 (Ivy Bridge) processors with 13GB/core of memory equipped with FDR Mellanox InfiniBand network. Each of its 132 nodes is equipped with a K40 GPU.

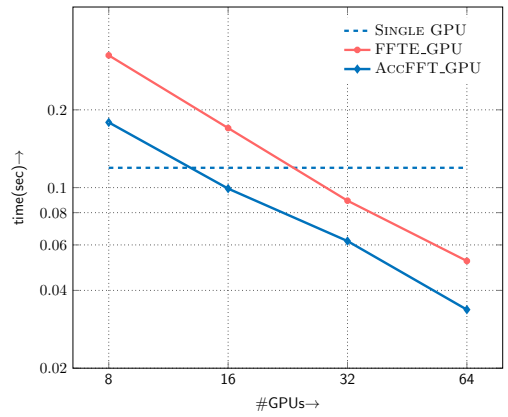
- The **Titan** system is a Cray XK7 supercomputer at ORNL. Titan has a total of 18,688 nodes consisting of a single 16-core AMD Opteron 6200 series processor, for a total of 299,008 cores. Each node has 32GB of memory. It is also equipped with a Gemini interconnect. In addition, all of Titan's 18,688 compute nodes contain an NVIDIA Tesla K20 GPU/



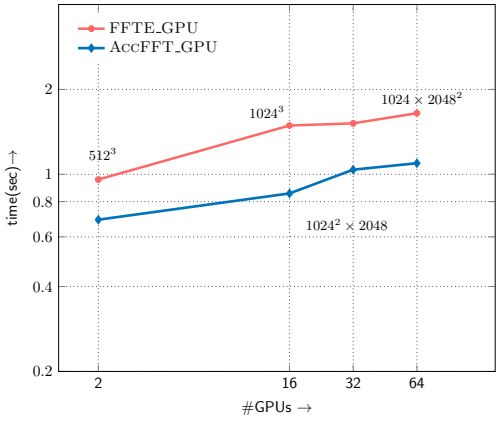
(a) R2C strong scaling for $N = 256 \times 512 \times 1024$



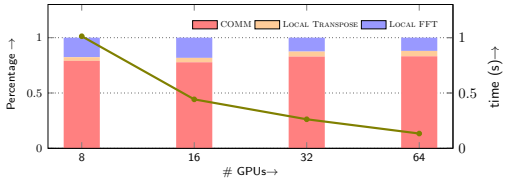
(b) R2C weak scaling



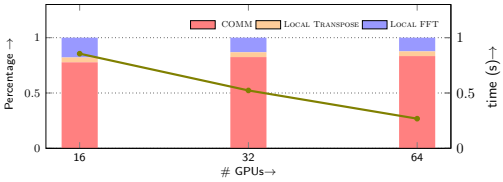
(c) C2C strong scaling for $N = 256 \times 512 \times 1024$



(d) C2C weak scaling

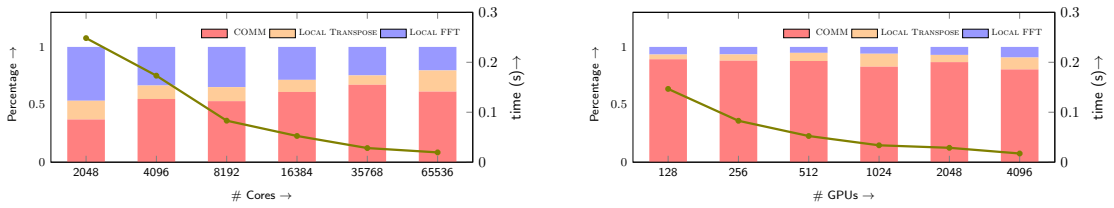


(e) R2C time break down for $N = 1024^3$



(f) C2C time break down for $N = 1024^3$

Figure 6.3: *Scaling results of AccFFT performed on Maverick. (a) strong scaling result for a non-structured R2C transform of size $N = 256 \times 512 \times 1024$, (b) R2C weak scaling, (c) C2C Strong scaling for $N = 256 \times 512 \times 1024$ and comparison with FFTE's GPU code, (d) C2C weak scaling, (e) breakdown of timings for R2C transform of size $N = 1024^3$, (f) breakdown of timings for C2C transform of size $N = 1024^3$. Dashed line in (a) and (c) show the timing for a 3D cuFFT transform without using the pencil decomposition (i.e. direct call to a single 3D FFT kernel).*



(a) **(b)**

Figure 6.4: *Strong scaling results for AccFFT performed on Titan. Timings in seconds are given for R2C transform of size $N = 1024^3$ on CPU and GPU (left/right). The breakdown of the total time in terms of local FFT computations, packing and unpacking (local transpose), and the communication times is given for different core counts.*

- The **Stampede** system at TACC is a Linux cluster consisting of 6400 compute nodes, each with dual, eight-core processors for a total of 102,400 available CPU-cores. Each node has two eight-core 2.7GHz Intel Xeon E5 (Sandy Bridge) processors with 2GB/core of memory and a three-level cache. Stampede has a 56GB/S FDR Mellanox InfiniBand network connected in a fat tree configuration.

- The **Lonestar5** is a system at the *Texas Advanced Computing Center* in Austin, TX, US (system specifications: dual socket Xeon® E5-2690 v3 (Haswell) with 12 cores per socket (24 cores/node) at 2.6GHz with 64 GB memory per node).

Implementation Details

All algorithms described in this work were implemented using C++, OpenMP and MPI. The only external libraries used were the MPI, FFTW, and CUFFT. On Titan, we used the GCC compiler and the CRAY-MPICH libraries. On Stampede and Maverick we used the Intel compilers and the Intel MPI library We compare our GPU code with FFTE library (version 6.0). The GPU code for FFTE library is written in Fortran and requires the commercial PGI compiler, and cuFFT for its FFT computations on the GPU. All the libraries were compiled with the MEASURE

planner flag where applicable. This flag is used to tune the libraries to the machine used. All results were computed in double precision and with pencil decomposition.

Parameters in the Experiments The parameters in our runs are the problem size N_0, N_1, N_2 and the number of tasks P. In most of the tests, we use $N_0 = N_1 = N_2$. The exception are two tests in which we test the library with non well-structured matrices, and a 4D test case. Except otherwise indicated, we use 16 MPI tasks per node for CPU runs and 2 MPI tasks per node for GPU tests. We use R2C to denote real-to-complex FFTs and C2C to denote complex to complex. Roughly speaking, the C2C transform has double the computation and communication compared to R2C transform. All timings are in seconds.

Experiments First we examine the performance of our code on the TACC systems, and then we discuss the results on Titan.

• In the first experiment we present scaling tests on Maverick. The strong scaling of the CPU and GPU code for a size of $N = 256 \times 512 \times 1024$ is shown in (Fig. 6.3 (a)). The GPU code scales similarly to the CPU code, however it is about $2\times$ slower. One reason for this is that we are comparing 16 CPU cores vs 1 GPU at each node. Local FFT computations scale almost perfectly as the cores are increased, so the advantage that the GPU has for its fast FFT computation would become negligible. The second point, is that part of the CPU communication occurs inside the same node, which is much faster than two GPUs communicating from different nodes. Weak scaling analysis shows the same trend, as shown in Fig. 6.3 (b).

There is currently no open source GPU code that supports pencil decomposition, other than digpufft and FFTE. The digpufft library, which is built on top of P3DFFT, is no longer maintained and our attempts to run it were not successful.

However, FFTE library is maintained and supports C2C transforms with pencil decomposition (172). Both our library as well as FFTE use cuFFT for local FFT computations. However, FFTE relies on the commercial PGI complier for the communication phase and does not do any optimizations. The two libraries are compared in Fig. 6.3 (c-d). AccFFT is consistently faster in both the strong and weak scaling tests. Moreover, AccFFT supports other transforms and is not just limited to C2C.

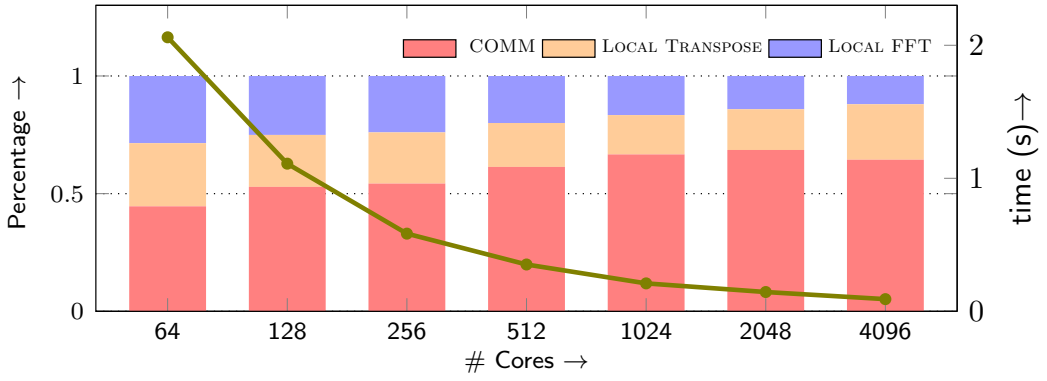
The break down of the timings for R2C and C2C transforms for $N = 1024^3$ is shown in figures 6.3 (e-f). Again the communication phase dominates the cost.

• Now we switch to Titan, where we consider the strong scaling of GPU versions of the code. We also present CPU results on Titan with 2 MPI tasks per node (1 tasks per NUMA). The goal is to compare the codes where the communication pattern between the CPU code and the GPU code is similar (that is there is no intra node acceleration that the GPU does not have). This allows us to see how effective does the PCIe overlapping works. The results are shown in Fig. 6.4 for up to 4096 GPUs and 65K cores. The efficiency for the largest CPU run is 40% for a $32\times$ increase in the core count (2,048 to 65K cores). The GPU code achieves 26% efficiency for a $32\times$ increase in the number of GPUs. The GPU code compared to 2 CPU cores is obviously faster as expected. An interesting observation is that the communication times are comparable which is because of the overlapping of PCIe exchanges. Although we did not have a chance to test 16 MPI tasks per node for the CPU case on Titan, but the CPU time is expected to become faster compared to the GPU code similar to the Maverick results.

• As a proof of concept we show how the code can be used for high dimensional transforms. In certain applications such as in signal processing, one

Table 6.1: Comparison of the fast algorithm for computing x-derivative for different problem sizes of $N = 1024^3, 512^3, 256^3$ performed on Lonestar5 machine, using 16 MPI tasks per node. The speedup achieved for the communication time as well as the total speedup is reported in the right-most columns.

		Fast Algorithm				**Slow Algorithm**				**Comm.**	**Total.**
	Procs	Total	FFT	Comm	Pack	Total	FFT	Comm	Pack	Speedup	Speedup
$N = 1024^3$	4096	2.3e−2	6.5e−3	1.3e−2	1.0e−2	4.7e−2	1.2e−2	4.3e−2	5.5e−3	**2.06 x**	**3.36 x**
	2048	5.3e−2	1.4e−2	3.0e−2	2.1e−2	8.8e−2	2.9e−2	7.9e−2	1.2e−2	**1.67 x**	**2.65 x**
	1024	1.1e−1	2.8e−2	6.0e−2	5.4e−2	1.6e−1	5.1e−2	1.4e−1	2.7e−2	**1.4 x**	**2.35 x**
	512	2.3e−1	3.4e−2	1e−1	9.0e−2	4.1e−1	7.2e−2	3.4e−1	4.1e−2	**1.77 x**	**3.41 x**
	256	4.2e−1	6.9e−2	1.5e−1	2.1e−1	5.8e−1	1.4e−1	4.2e−1	8.0e−2	**1.38 x**	**2.79 x**
	128	7.6e−1	1.4e−1	1.9e−1	4.4e−1	1	2.7e−1	6.1e−1	1.6e−1	**1.32 x**	**3.13 x**
	64	1.6	2.8e−1	3.6e−1	9.3e−1	1.8	5.0e−1	9.9e−1	3.3e−1	**1.15 x**	**2.8 x**
	32	3.3	5.9e−1	7.5e−1	1.9	2.8	1.1	1.0	6.5e−1	**0.85 x**	**1.38 x**
$N = 512^3$	4096	2.8e−3	3.4e−4	2.1e−3	9.5e−4	7.6e−3	1.2e−3	7.3e−3	4.3e−4	**3.42 x**	**2.77 x**
	2048	5.1e−3	1.3e−3	3.5e−3	2.0e−3	9.2e−3	2.3e−3	8.7e−3	8.5e−4	**2.51 x**	**1.82 x**
	1024	1.1e−2	3.1e−3	5.4e−3	5.0e−3	1.9e−2	5.4e−3	1.8e−2	2.0e−3	**3.23 x**	**1.82 x**
	512	1.9e−2	4.1e−3	7.6e−3	7.3e−3	2.6e−2	6.8e−3	2.1e−2	5.4e−3	**2.72 x**	**1.36 x**
	256	4.1e−2	9.3e−3	1.6e−2	1.4e−2	5.6e−2	1.5e−2	4.6e−2	1.2e−2	**2.86 x**	**1.37 x**
	128	8.4e−2	1.8e−2	3.1e−2	3.1e−2	1.2e−1	3.3e−2	8.1e−2	2.1e−2	**2.66 x**	**1.39 x**
	64	1.6e−1	3.6e−2	4.0e−2	7.9e−2	2.0e−1	6.4e−2	1.3e−1	3.9e−2	**3.14 x**	**1.26 x**
	32	3.3e−1	6.7e−2	4.6e−2	2.0e−1	3.3e−1	1.3e−1	1.2e−1	7.9e−2	**2.65 x**	**1.01 x**
$N = 256^3$	4096	8.3e−4	4.0e−5	7.6e−4	1.0e−4	1.1e−3	1.5e−4	1.0e−3	0	**1.38 x**	**1.32 x**
	2048	8.4e−4	7.9e−5	6.9e−4	2.0e−4	1.3e−3	2.5e−4	1.3e−3	0	**1.84 x**	**1.6 x**
	1024	1.1e−3	1.6e−4	8.1e−4	4.0e−4	2.2e−3	5.3e−4	2.1e−3	1.0e−4	**2.54 x**	**2 x**
	512	4.7e−3	1.9e−4	4.1e−3	4.1e−4	7.3e−3	5.2e−4	7.1e−3	2.6e−4	**1.74 x**	**1.57 x**
	256	4.6e−3	7.1e−4	2.9e−3	1.2e−3	5.9e−3	1.1e−3	5.4e−3	5.5e−4	**1.89 x**	**1.29 x**
	128	8.4e−3	1.8e−3	3.1e−3	3.3e−3	1.0e−2	2.6e−3	7.7e−3	1.1e−3	**2.47 x**	**1.24 x**
	64	1.7e−2	4.1e−3	5.9e−3	6.7e−3	2.1e−2	5.6e−3	1.4e−2	4.0e−3	**2.34 x**	**1.23 x**
	32	3.2e−2	9.8e−3	6.1e−3	1.5e−2	3.6e−2	1.4e−2	1.7e−2	1.0e−2	**2.71 x**	**1.13 x**



(a)

Figure 6.5: *CPU strong scaling results of AccFFT using the pencil decomposition algorithm for a four dimensional problem size of* $512 \times 256 \times 128 \times 64$ *on Stampede. Wall-clock time is given for different core counts for a forward C2C FFT.*

Table 6.2: Comparison of the fast algorithm for computing y-derivative for different problem sizes of $N = 1024^3, 512^3, 256^3$ performed on Lonestar5 machine, using 16 MPI tasks per node. The speedup achieved for the communication time as well as the total speedup is reported in the rightmost columns. The noticeable communication speedup is because all or part of the exchanges occur in the NUMA node and the MPI library does not have to send messages through the network.

		Fast Algorithm				**Slow Algorithm**				**Comm.**	**Total.**
	Procs	Total	FFT	Comm	Pack	Total	FFT	Comm	Pack	Speedup	Speedup
$N = 1024^3$	4096	2.9e−2	6.9e−3	1.8e−2	1.2e−2	4.7e−2	1.2e−2	4.2e−2	5.5e−3	**2.32 x**	**1.59 x**
	2048	5.2e−2	1.4e−2	2.9e−2	2.3e−2	8.8e−2	2.9e−2	7.9e−2	1.2e−2	**2.72 x**	**1.7 x**
	1024	1.1e−1	2.9e−2	5.8e−2	4.7e−2	1.6e−1	5.1e−2	1.4e−1	2.7e−2	**2.44 x**	**1.49 x**
	512	1.4e−1	3.6e−2	2.1e−2	6.8e−2	3.9e−1	7.0e−2	3.3e−1	4.1e−2	**15.65 x**	**2.92 x**
	256	2.7e−1	7.6e−2	3.8e−2	1.4e−1	5.8e−1	1.3e−1	4.2e−1	8.1e−2	**11.17 x**	**2.14 x**
	128	5.7e−1	1.5e−1	7.6e−2	3.0e−1	1.0	2.6e−1	6.2e−1	1.6e−1	**8.14 x**	**1.78 x**
	64	1.1	3.0e−1	1.5e−1	6.0e−1	1.8	5.0e−1	9.7e−1	3.3e−1	**6.35 x**	**1.57 x**
	32	2.4	6.0e−1	2.6e−1	1.4	2.8	1.1	1.0	6.5e−1	**3.92 x**	**1.16 x**
$N = 512^3$	4096	1.1e−2	4.1e−4	9.7e−3	1.2e−3	7.4e−3	1.2e−3	7.1e−3	4.3e−4	**0.73 x**	**0.7 x**
	2048	5.6e−3	1.4e−3	3.7e−3	2.5e−3	9.1e−3	2.3e−3	8.5e−3	8.5e−4	**2.29 x**	**1.6 x**
	1024	1.2e−2	3.1e−3	7.3e−3	5.6e−3	1.9e−2	5.5e−3	1.8e−2	2.0e−3	**2.4 x**	**1.61 x**
	512	1.4e−2	5.0e−3	3.7e−3	7.8e−3	2.5e−2	6.6e−3	2.0e−2	4.7e−3	**5.56 x**	**1.82 x**
	256	2.9e−2	9.7e−3	6.1e−3	1.4e−2	5.6e−2	1.6e−2	4.6e−2	1.3e−2	**7.42 x**	**1.93 x**
	128	6.3e−2	2.0e−2	1.5e−2	3.2e−2	1.2e−1	3.3e−2	8.6e−2	1.9e−2	**5.86 x**	**1.95 x**
	64	1.3e−1	3.8e−2	3.0e−2	6.4e−2	2.1e−1	6.7e−2	1.3e−1	3.9e−2	**4.41 x**	**1.65 x**
	32	2.8e−1	6.9e−2	3.3e−2	1.5e−1	3.3e−1	1.2e−1	1.3e−1	7.9e−2	**3.74 x**	**1.2 x**
$N = 256^3$	4096	6.0e−4	7.2e−5	5.0e−4	1.2e−4	4.1e−3	1.5e−4	4e−3	0	**8 x**	**6.76 x**
	2048	7.2e−4	1.4e−4	5.3e−4	2.4e−4	1.5e−3	2.5e−4	1.4e−3	0	**2.59 x**	**2.03 x**
	1024	1.2e−3	2.2e−4	8.6e−4	4.9e−4	2.1e−3	5.3e−4	1.9e−3	1.0e−4	**2.26 x**	**1.66 x**
	512	1.8e−3	1.6e−4	9.9e−4	5.8e−4	7.5e−3	5.4e−4	7.3e−3	2.6e−4	**7.34 x**	**4.3 x**
	256	2.5e−3	7.2e−4	6.1e−4	1.3e−3	6.0e−3	1.1e−3	5.4e−3	5.3e−4	**8.93 x**	**2.36 x**
	128	5.1e−3	1.9e−3	8.5e−4	2.4e−3	1.1e−2	2.8e−3	7.8e−3	1.1e−3	**9.12 x**	**2.06 x**
	64	1.2e−2	4.5e−3	2.2e−3	5.5e−3	2.1e−2	5.8e−3	1.4e−2	3.9e−3	**6.14 x**	**1.69 x**
	32	2.7e−2	9.6e−3	4.2e−3	1.1e−2	3.9e−2	1.5e−2	1.9e−2	1.0e−2	**4.57 x**	**1.44 x**

needs to compute FFT of a 4D array, where the last dimension corresponds to time. Figure 6.5 shows the strong scaling of the CPU code for a C2C transform of size $N = 512 \times 256 \times 128 \times 64$ [4].

Spectral operators:

As discussed above, the performance of the gradient and divergence operators can be improved substantially by using Algorithm 4. The results for computing x and y derivative on Lonestar5 system are shown in Tables 6.1 and 6.2, respectively. The results original and fast algorithms (Eq. 6.1 vs Eq. 6.2) are compared for different problem sizes of $N = 1024^3, 512^3, 256^3$ and the speedup is given for the communication and total evaluation times. The performance boost is significant especially for large core counts, where the major bottleneck is the communication as opposed to local computations (such as local FFTs or packing and unpacking operations). In particular, the communication speedup achieved for the y-derivative is quite considerable compared to the x-derivative. This is due to the fact that the majority of the communication needed for transposing the data in y-direction can occur via NUMA transfers as opposed to transfers via the network interconnect. This is a direct consequence of using a pencil decomposition for the MPI tasks. However, when transposing the array in x-direction we have to perform intra-node communication which is considerably more expensive.

6.5 Conclusions

We presented AccFFT, a library for distributed memory FFTs with several unique features: distributed GPU calculations, communication overlap using pipelin-

[4] This feature has not been added to the public repo the library yet.

ing for the GPU version, and support for real and complex transforms.

The performance of the library was tested on three different machines: The Stampede and Maverick systems at TACC, as well as the Titan system at ORNL. The largest test corresponds to 65K cores as well as 4,096 GPUs of Titan. To the best of our knowledge, the latter is the only open source library supporting pencil decomposition for GPUs. One of the observations of this work is that parallel GPU is not faster when compared to parallel CPU, in cases where there are multiple CPU cores per node. The main reason for this is that local FFT computations scale almost perfectly. So using more MPI tasks per node would significantly reduce the local FFT time for CPU. Moreover, the communication phase of the CPU would benefit from fast intra-node exchanges through shared memory. This is not the case for machines which habe one GPU per node. In that case two GPUs have to communicate through the network which is much costlier.

This work is by no means complete with these results. Using non-blocking collectives can further accelerate the calculations, if the user interleave other computations while the FFT communication is completing. Other possibilities include distributing the data to both the CPU and GPU on each node. This has been shown to be an effective strategy on single node computations (186). Another possibility is to extend the method to Xeon Phi accelerators.

Chapter 7

FFT, FMM, or MultiGrid?

From molecular dynamics and quantum chemistry, to plasma physics and computational astrophysics, Poisson solvers in the unit cube are used in many applications in computational science and engineering. In this work, we benchmark and discuss the performance of the scalable methods for the Poisson problem which are used widely in practice: the Fast Fourier Transform (FFT), the Fast Multipole Method (FMM), the geometric multigrid (GMG) and algebraic multigrid (AMG). Our focus is on solvers support high-order, highly non-uniform discretizations, but for reference we compare with solvers specialized for problems on regular grids. So, we include FFT, since it is a very popular algorithm for several practical applications, and the finite element variant of HPGMG, a high-performance geometric multigrid benchmark. In total we compare five different codes, three of which are developed in our group. Our FFT, GMG and FMM are parallel solvers that use high-order approximation schemes for Poisson problems with continuous forcing functions (the source or right-hand side). Our FFT code is based on the FFTW for single node parallelism. The AMG code is from the Trilinos library from the Sandia National Laboratory. Our geometric multigrid and our FMM support octree based mesh refinement, variable coefficients, and enable highly non-

This chapter is based on existing publication: . *Gholami, D. Malhotra, H. Sundar and G. Biros. FFT, FMM, or Multigrid? A comparative Study of State-Of-the-Art Poisson Solvers for Uniform and Nonuniform Grids in the Unit Cube. SIAM Journal on Scientific Computing, Vol. 38 (3), 2016.*

The author contributed in designing the benchmarks and implemeted parallel FFT algorithm and performed the benchmarking of high performance multigrid and FFT. Dhairya implemented the FMM section and Hari implementd the multigrid algorithm. Prof. Biros contributed in the algorithms and supervised the work.

uniform discretizations. The GMG, actually also supports complex (non-cubic) geometries using a forest of octrees.

We examine and report results for weak scaling, strong scaling, and time to solution for uniform and highly refined grids. We present results on the Stampede system at the Texas Advanced Computing Center and on the Titan system at the Oak Ridge National Laboratory. In our largest test case, we solved a problem with 600 billion unknowns on 229,379 cores of Titan. Overall, all methods scale quite well to these problem sizes. We have tested all of the methods with different source functions (the right hand side in the Poisson problem). Our results indicate that FFT is the method of choice for smooth source functions that require uniform resolution. However, FFT loses its performance advantage when the source function has highly localized features like internal sharp layers. FMM and GMG considerably outperform FFT for those cases. The distinction between FMM and GMG is less pronounced and is sensitive to the quality (from a performance point of view) of the underlying implementations. In most cases, high-order accurate versions of GMG and FMM significantly outperform their low-order accurate counterparts.

7.1 Introduction

The need for large scale parallel solvers for elliptic partial differential equations (PDEs) pervades across a spectrum of problems with resolution requirements that cannot be accommodated on current systems. Several research groups are working on technologies that scale to trillions of unknowns and billions of cores. To illustrate some of the issues in scaling such solvers and to provide a (non-exhaustive) snapshot of current technologies, we conduct an experimental study of solving a simple model elliptic PDE and compare several state of the art method-

ologies.

We restrict our attention to the following model problem: given f, a smooth and periodic function in the unit cube, we wish to find u (also smooth and periodic in the unit cube) such that

$$-\Delta u = f, \tag{7.1}$$

where Δ is the Laplace operator. This is also known as the constant-coefficient Poisson problem. It encapsulates many of the difficulties in solving elliptic partial differential equations (PDEs). We chose this problem because all four methods can address it in an algorithmically optimal way. Algorithms for solving this problem, also known as "*Poisson solvers*" find applications in astrophysics, chemistry, mechanics, electromagnetics, statistics, and image processing, to name a few. Vendors like Intel and NVIDIA provide Poisson solvers in their math libraries. Examples of scientific computing libraries that provide Poisson solvers include PETSc (9), Trilinos (73), deal.II (10), and MATLAB.

Poisson solvers must scale to trillions of unknowns. Example of methods that scale well are the FFT (based on spectral discretizations)[1], the Fast Multipole Method (based on discretizing the integral equation reformulation of (7.1), and multigrid methods (for stencil-based discretizations). Other scalable methods include domain decomposition and wavelet transforms, which will not be discussed here. Despite the existence of many different Poisson solvers there has been little work in *directly* benchmarking the computational efficiency of these methods, in particular for the case of non-uniform discretizations. Such benchmarking is quite typical in other scientific computing areas (e.g., sorting, matrix computations, and

[1]FFT can be used also to diagonalize and invert stencil discretizations on uniform grids. We are not discussing this case here.

graph partitioning).

Methodology and contributions

In this chapter we benchmark five state-of-the-art algorithms and implementations, three from our group and two from different groups:

The first solver is parallel FFT using the AccFFT which has been recently developed in our group (58). AccFFT, built on top of FFTW, uses MPI and OpenMP, as well as novel communication schemes that makes it faster than similar libraries. We report comparisons with PFFT (140) and P3DFFT (135).

The second solver is the ML algebraic multigrid solver which is part of the Trilinos library developed and maintained by the U.S. Department of Energy (54, 73). In our runs, we use ML with MPI. It is one of the most scalable, general purpose codes available. It can handle much more complex problems than the one we consider here.

The third solver is an in-house Geometric Multigrid scheme that uses continuous Galerkin discretizations on octree meshes. The low-order version of the code appeared in (163), but the scalable high-order results we report here are new, as well as the outline of the algorithm. The sequential algorithm is described in (165). The library uses MPI and more details about the new algorithm will be presented in §7.2.

The fourth code is PvFMM, also an in-house novel parallel volume FMM that supports continuous as well as particle sources (f). It uses octree discretization using Chebyshev polynomials at each leaf node to represent f. PvFMM uses MPI and OpenMP (106).

The fifth code is HPGMG (3) and it is a high-performance computing bench-

mark code for regular grids with finite-element and finite-volume implementations. The finite-element implementation is more general as it supports variable coefficients and coordinate transformation Jacobians that resemble overset grids and non-uniform grids. For this reason we include only the finite-element version in our tests.

We compare these five methods on two different architectures, Stampede and Titan, and we discuss two main questions:

- Which method is faster? What matters the most is the wall-clock time to solution. That is, given f, we would like to evaluate u (typically at a given number of points) to a specified accuracy. We consider two main cases, highly oscillatory fields, for which a regular grid is necessary, and highly localized fields for which adaptively refined meshes are expected to be more effective.
- How does the cost per unknown compare for the different methods given a fixed algebraic accuracy? This test focuses on constant in the complexity estimates which are functions of the problem size, the number of processors, the approximation order, and of course the implementation. We perform weak and strong scaling studies to directly compare the complexity estimates on specific architectures using the same problem size. We have scaled our runs up to 229,376 cores on Titan for problems with up to 600 billion unknowns. Our goal here is not to fit a detailed performance model to the runs, but rather to identify whether there are order-of-magnitude differences in the performance between these methods.

To our knowledge, such a benchmark at these problem sizes and number of cores is the first of its kind. We view it as a companion to existing theoretical complexity analysis. In addition to work complexity given a problem size, we

consider the issue of work complexity given a target accuracy using both uniform and non-uniform grids—the "right" problem size is *not known* a priori. One reason such a study has not taken place is that the underlying technologies have not been available at this scale. Indeed, we are not aware of any other distributed-memory FMM codes that allow f to be an arbitrary function (most existing codes only support sums of delta functions, also known as point-FMMs methods). Also, the only other scalable, high-order, multigrid scheme we know is that of Paul Fischer's group (103). Both of our GMG and FMM codes support arbitrary order discretizations. In summary, we test weak and strong scalability of all of these methods and report time to solution, and setup time for different test cases.

Qualitatively, the results of our study for solving (7.1) can be summarized as follows: FFT is the method of choice for uniform discretizations even at large-core count problems. FMM and MG are the methods of choice in the presence of strongly-localized features. AMG scales well, but it is significantly slower, especially when including setup costs. Uniform grid, second-order discretizations (e.g. the 7-point Laplacian) end up being $1000\times$ or more, slower than high-order schemes and the FFT for high-accuracy solutions. Even for low-accuracy solutions in non-uniform grids, second-order methods suffer. Third or higher order can offer significant speed-ups. Of course, these conclusions are valid only when the solution is smooth.

Related work

There is rich literature discussing the accuracy and scalability of FFT, FMM and multigrid but to our knowledge, little work has been done on directly comparing the efficiency of these schemes. In (5), the authors compare FMM, FFT,

and GMG for particle summation (point sources) with periodic conditions. This is different from what we look here, which is a Poisson solver with continuous right-hand side, not point sources. For the particle mesh variant of the codes tested, there is a continuous source but it is uniform. Furthermore they only consider nearly uniform distributions of charges. This is fine for molecular dynamics, but it is of rather narrow scope. Here we consider highly non-uniform sources. The results FFT and GMG solvers are not considered separately but only as part of particle-mesh solvers that include many additional components that complicate the interpretation of the results for other applications. In (53), theoretical complexity estimates for the scaling of FFT and multigrid (for uniform grids) are provided and their implications towards the design of exascale architectures is discussed. In (33), a similar study is carried for the FFT, along with experimental results on both CPU and hybrid systems. An interesting performance model is introduced that accounts for both intra-node and inter-node communication costs. In (14), the authors consider complexity estimates that account for low-level hardware details and consider the viability of different applications including FFT, and matrix vector multiplications, and molecular dynamics simulations. In (52), the authors discuss the scalability of algebraic multigrid (on uniform grids), provide performance models, and conduct an experimental scalability study on up to 65,536 cores. A perspective on scalability is given in (180). In (190), the authors discuss the scalability of a point FMM code to exascale architectures and provide scalability results up to 32,768 cores and 40 billion unknowns. In our group we have worked on scalable geometric multigrid methods and their comparison to algebraic multigrid schemes (163) (but only for low-order discretizations), as well as parallel FMM schemes based on the kernel-independent variant of the FMM kernels (93, 189).

All of these studies are critical in understanding the scalability of the schemes and helping co-design the next architectures. We consider our study as a companion to these works as it provides experimental data that can be further analyzed using performance models. Also, except for the work in our group, others have only considered uniformly refined grids (and low-order discretizations for the multigrid). Here we consider all cases: uniform and refined grids, low and high-order discretizations, and four major algorithms.

Other scalable approaches to solving the Poisson problem include hybrid domain decomposition methods (103). A very efficient Poisson solver is based on a non-iterative domain decomposition method (114) using a low-order approximation scheme. In (92), that solver was compared with a high order volume FMM. The FMM solver required $4\times$–$100\times$ fewer unknowns. Other works based on FFTs, tree codes and multigrid that are highly scalable (albeit for low-order, or point FMM only) include (37, 128, 138, 139, 151).

Limitations

Our study is limited to problems with constant coefficients on the unit cube, with periodic boundary conditions, and smooth solutions. AMG is the only method that is directly applicable to general geometries and problems with complex coefficients. GMG methods also can handle certain types of complex geometries. FMM can be used for constant-coefficient problems on arbitrary geometries but can also be extended to variable coefficients using volume integral equations. Complex geometries are also possible but the technology for such problems is not as developed as for algebraic multigrid. In our tests we only use periodic boundary conditions but the results for the FMM and GMG apply to Neumann and Dirichlet problems

on the unit cube. For our high-order geometric multigrid we use a smoother that heuristically works well but we do not have supporting theory that shows that we have made the best possible choice (see §7.2.3). The solve phase for these codes has been optimized significantly but it doesn't mean that it can be further improved. In §7.4, we compare our GMG with the stencil-based HPGMG library (3) to give a measure of its performance relatively to a highly optimized (but less general code). Another limitation with respect performance is the use of heterogeneous architectures.

Comparing highly specialized codes with much general purpose codes (like the AMG solvers we use here) is problematic but, we think, informative as it provides some data for future developments of software that must be as efficient as possible if it is to be used in the million and billion-core systems. As we mentioned other methods like domain decomposition or hybrids like particle-in-cell methods are not discussed. One salient disadvantage of high-order methods is that when used with explicit time stepping schemes (for example, due to coupling to a transport equation) they can lead to extremely small time steps. Finally, FFTs, FMM, and Multigrid can be also discussed in other contexts (e.g., Ewald sums, particle-in-cell methods, signal analysis), which we do not do here.

Outline of the chapter

In section §7.2, we summarize complexity estimates for FFT, Multigrid, and FMM. For FMM and geometric multigrid we provide some more detail since some components of the underlying algorithms are new. In §7.3, we summarize the experimental setup, platforms, and the choice of the right hand sides. In §7.4, we present and discuss the results of our experiments.

Notation: We use p to denote the number of cores q to denote the order of polynomial approximation for f and u, m the FMM approximation order for the far field, and N the total number of unknowns.

7.2 Methods

Here we describe the basic algorithmic components of each method, and their overall complexity (setup and solve) and the solve complexity ($T_{\text{FMM}}, T_{\text{FFT}}, T_{\text{GMG}}$). In all of our results we assume that $N/p \gg 1$ and the complexity estimates are stated for an uncongested hypercube topology and uniform grids.

7.2.1 The Fast Fourier Transform

The Fast Fourier Transform is an algorithm for computing the Discrete Fourier Transform (DFT) of a signal in $\mathcal{O}(N \log N)$. Several efficient implementations of single-node FFT exist (for example, FFTW (50), Intel MKL's FFT (182), IBM's ESSL library(45), etc). In addition, several FFT algorithms have been proposed for distributed machines (for example see (47, 99, 140, 170, 171)). In computing 3D FFTs, the key challenge is in dividing the data across the processes. One option is slab decomposition in which the data is partitioned into p slabs or slices, each containing N/p samples. Such an approach however limits the maximum number of process to the number of slices in the input data ($p \leq \mathcal{O}(N^{1/3})$).

This bound can be increased to $\mathcal{O}(N^{2/3})$ by using a 2D decomposition, which is also referred to as *pencil decomposition*. Consider a 3D function of size $N = N_x \times N_y \times N_z$ whose FFT we wish to compute on p processes. AccFFT maps the processes into a 2D grid of size $p_x \times p_y = p$, each having a block (or *"pencil"*) of size

$N_x/p_x \times N_y/p_y \times N_z$ (or an equivalent partition). In the limit of $p = N_x \times N_y = N^{2/3}$ processes, each process will get a one dimensional pencil of the data (of length N_z). With the pencil decomposition, each process has all the data in one direction (e.g. z direction), and partial data of the other two directions (x and y). Consequently, the 1D FFT along the z direction can be performed independent of other processes.

Following the local 1D FFT (along the z-direction), each process exchanges its data using all-to-all communication with other processes in the same row (p_x) to form the transpose. After a process receives the data from other processes, it computes the 1D FFT along the next direction. This process is once again repeated in the column direction (p_y). The data layout in each stage is shown schematically in Figure 7.1.

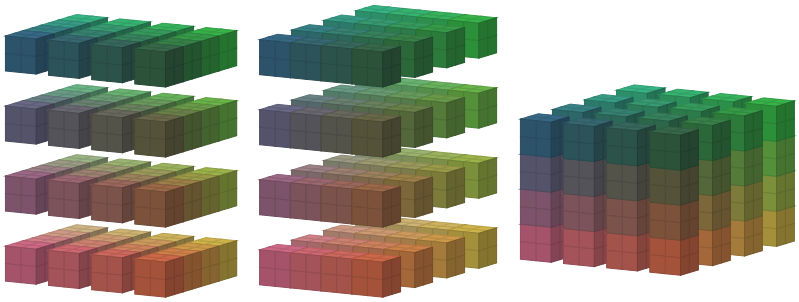


Figure 7.1: Data layout in different stages of computing a forward/inverse FFT using pencil decomposition for $p_x = p_y = 4$.

The communication cost of distributed FFT is given by $\mathcal{O}(\frac{N}{\sigma(p)})$, where $\sigma(p)$ is the bisection bandwidth of the network; for a hypercube it is $p/2$ (135). The total execution time of one 3D FFT can be approximated by

$$T_{\text{FFT}} = \mathcal{O}\left(\frac{N \log N}{p}\right) + \mathcal{O}\left(\frac{N}{p}\right).$$

Because we are comparing different solvers, it is important that each of the implementations are optimal. To address this concern, we compare AccFFT's per-

formance with two other libraries in Table 7.1. The first library is P3DFFT, written by Dmitry Pekurovsky (135). P3DFFT is a robust parallel FFT library that has been successfully used in different applications and shown to have excelelent scalability. The second library is PFFT which has been recently released by Michael Pippig (140). PFFT supports high dimensional FFTs as well as optimized pruned FFTs (with respect to time). It is designed to allow the user to compute transforms of more general data layouts. It has been tested up to 200K cores of BlueGene/Q, and shown to be scalable. Our AccFFT supports slab and pencil decompositions for CPU and GPU architectures, for real-to-complex, complex-to-complex, and complex-to-real transforms. It uses a series of novel schemes and in our tests results in an almost 2× speedup over P3DFFT and 3× speedup over PFFT as shown in Table 7.1. Details of all of the performance optimizations of AccFFT are beyond the scope of this chapter (please see chapter 6). We refer to (58) for details on the method. We would like to remark that P3DFFT and PFFT have been designed to allow for more general features that are currently not available in AccFFT. In our tests we have used both P3DFFT and AccFFT.[2]

To solve the Poisson problem we compute the FFT of f in (7.1), scale it by the corresponding inverse diagonal form of the Laplace operator (using a Hadamard product), and compute the inverse Fourier transform of the result. The overall complexity of solving the Poisson problem with FFT is the same order as T_{FFT}.

[2]We have compiled all libraries with performance tuning on. This significantly increases the setup time. However, this cost can be amortized across multiple solves, so for most applications it is negligible.

	512 Cores		2048 Cores		8192 Cores	
	FFT	*Setup*	*FFT*	*Setup*	*FFT*	*Setup*
AccFFT	0.256	10.8	0.070	3.6	0.024	1.5
P3DFFT	0.468	10.0	0.118	12.9	0.040	4.1
PFFT	0.848	19.4	0.257	6.9	0.073	2.8

Table 7.1: *Comparison of total time and setup time of AccFFT, P3DFFT, and PFFT libraries. We report timings for a single threaded, in-place real-to-complex FFT of size 1024^3 on 512, 2048, and 8192 cores of TACC's Stampede platform. All the libraries were tested using* `MEASURE` *flag as well as other library specific options to achieve the best performance.*

7.2.2 The Fast Multipole Method

The FMM was originally developed to speedup the solution of particle N-body problems by reducing the complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$ (24, 26). Solving Poisson's equation by computing volume potential is similar to an N-body problem, except that the summation over source points is replaced by an integral over the continuous source density. The work (41, 92) shows how one needs to modify the particle-FMM to obtain a volume FMM and this is the approach used in our implementation. Our contribution is that we have focused on the efficient implementation of this algorithm by considering efficient blocking and vectorization strategies for per core performance. We further optimized this method by adding support for the Intel Phi accelerator and extended it to a distributed memory method based on our hypercube communication algorithm (93). For results in this chapter, we do not use Phi since none of the other solvers can use it. Below we summarize the main components of our algorithm, which is described in full detail in (106).

We evaluate the solution to (7.1) as a convolution of the source density function f with $K(x) = -(4\pi|x|)^{-1}$ (free space Green's function for Laplace equation). We accelerate computation of this convolution using FMM. The basic idea behind FMM is to construct a hierarchical decomposition of the computational domain

using an octree. Then, the solution at each point x can be evaluated by summing over contributions from all octants in the octree. This summation is split into near and far interactions:

$$u(x) = \sum_{B \in \text{NEAR}(x)} \int_B K(x-y) f(y) + \sum_{B \in \text{FAR}(x)} \int_B K(x-y) f(y)$$

The near interactions (from $B \in \text{NEAR}(x)$) are computed through direct integration. The far interactions (from $B \in \text{FAR}(x)$) are low-rank and can be approximated. In the following sections, we describe the tree construction and the different interactions in more detail.

Octree Construction

We partition the computational domain using an octree data structure. For each leaf octant B, we approximate the source density f by Chebyshev polynomials of order q

$$f(y) = \sum_{i,j,k \geq 0}^{i+j+k<q} \alpha_{i,j,k} T_i(y_1) T_j(y_2) T_k(y_3)$$

where, T_i is the Chebyshev polynomial of degree i and y is a point in the octant B. The absolute sum of the highest order coefficients is used as an estimate of the truncation error. This error estimate is used to refine adaptively until a specified error tolerance is achieved.

We also apply 2:1-balance constraint on our octree, that is we constrain the difference in depth of adjacent leaf octants to be at most one. To do this, we need to further subdivide some octants and is called 2:1-balance refinement (164).

Far Interactions

We define a source and a target octant to be well separated (or far) if they are at the same depth in the octree and are not adjacent. To compute far interactions we use two building blocks: multipole expansions and local expansions.[3] The multipole expansion approximates the potential of an octant far away from it. The local expansion approximates the potential within an octant due to sources far away from it. The interactions are then approximated by computing a multipole expansion (source-to-multipole, multipole-to-multipole) for the source octant, multipole-to-local (V-list) translation and then evaluating the local expansion (local-to-local, local-to-target) at the target octant. We use the kernel-independent variant of FMM in our implementation. The form of the multipole and local expansions and the V-list translation operator for this variant are discussed in detail in (189).

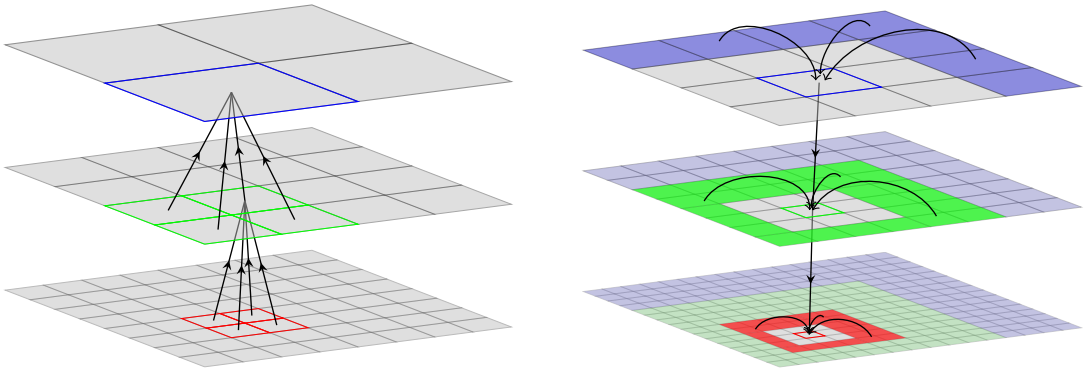
Near Interactions

The near interactions (source-to-target or U-list), between pairs of adjacent source and target octants, are computed using exact integration. Since these are singular and near-singular integrals (because the kernel has a singularity), it is not feasible to do this on the fly. We have to precompute integrals over Chebyshev basis functions, then the integral can be represented as a linear transformation:

$$u(x) = \int_B K(x-y)f(y) = \int_B K(x-y) \sum_{i,j,k} \alpha_{i,j,k} T_{i,j,k}(y) = \sum_{i,j,k} \alpha_{i,j,k} I_x$$

[3] We refer to multipole expansions and we use the term *"multipole order"* m to denote the accuracy of the far field approximation, following the analytic FMM. However, in our implementation we use the kernel independent FMM and m refers to the square root of the number of equivalent points. We used the terminology from the original FMM for the readers that are not familiar with the kernel independent FMM (189).

We precompute these integrals for each target point x, and for each interaction direction then construct interaction matrices. These interactions can then be evaluated through matrix-vector products. To limit the number of possible interactions directions and make this precomputation possible, we need the 2:1-balance constraint (discussed above).



(a) Upward Pass **(b)** Downward Pass

Figure 7.2: (a) Upward Pass: constructing multipole expansions. (b) Downward Pass: constructing local expansions, evaluating near interactions.

Summary of Volume FMM

The overall algorithm for volume FMM can be summarized as follows:

- *Tree Construction*: Construct a piecewise Chebyshev approximation of the source density using octree based domain decomposition and perform 2:1-balance refinement.
- *Upward Pass:* For all leaf nodes apply source-to-multipole (S2M) translation to construct the multipole expansion from the Chebyshev approximation. For all non-leaf nodes apply multipole-to-multipole (M2M) translations in bottom-up order, to construct multipole expansion from that of its children, as shown in Fig. 7.2.

- *Communication:* In the distributed memory implementation, we communicate the source density and the multipole expansions for ghost octants.
- *Downward Pass:* For all octree nodes, apply V-list and source-to-local (X-list) translations to construct the local expansion of each octant. In top-down order apply the local-to-local (L2L) translation to all nodes (Fig. 7.2). For all leaf octants, apply local-to-target (L2T), multipole-to-target (W-list) and source-to-target (U-list) translations to construct the final target potential as piecewise Chebyshev interpolation.

In our results, we report tree construction as the setup phase. The upward-pass, communication and downward-pass together constitute the evaluation phase.

Parallel Fast Multipole Method

Here we list the important features of the intra-node parallelism (accelerators, multithreading and vectorization) and distributed memory parallelism. A detailed discussion of these optimizations is beyond the scope of this chapter. We refer the interested readers to (106) for a detailed discussion of these concepts.

U,W,X-List Optimizations: We group similar interactions, those with the same interaction matrix or related by a spatial symmetry relation into a single matrix-matrix product, evaluated efficiently through DGEMM.

V-List Optimizations: The V-list interactions involve computation of the Hadamard products, which has low computational intensity and is therefore bandwidth bound. We rearrange data and use spatial locality of V-list interactions to optimize cache utilization. This along with use of AVX and SSE vector intrinsics and OpenMP allowed us to achieve over 50% of peak performance for this operation on the Intel Sandy Bridge architecture.

Distributed 2:1 Balance Refinement: We developed a new distributed memory algorithm for 2:1 balance refinement, which is more robust for highly non-uniform octrees than our earlier implementation.

Distributed Memory Parallelism: We use Morton ordering to partition octants across processors during the tree construction. In the FMM evaluation, after the upward pass, we need to construct the local essential tree through a reduce-broadcast communication operation. For this, we use the hypercube communication scheme of (93).

Complexity

The cost of FMM evaluation is given by the number of interactions between the octree nodes weighted by the cost of each translation. The cost of each interaction depends on the multipole order m and the order of Chebyshev polynomials q. Let N_{oct} be the local octree nodes, N_{leaf} the number of local leaf nodes. Also, let N_U, N_V, N_W, $N_X (= N_W)$ denote the number of interactions of each type U,V,W and X-list respectively. The overall cost is summarized in Table 7.2.

Interaction Type	Computational Cost
S2M, L2T	$\mathcal{O}(N_{leaf} \times q^3 \times m^2)$
M2M, L2L	$\mathcal{O}(N_{oct} \times m^2 \times m^2)$
W-list, X-list	$\mathcal{O}(N_W \times m^2 \times q^3)$
U-list	$\mathcal{O}(N_U \times q^3 \times q^3)$
V-list	$\mathcal{O}(N_V \times m^3 + N_{oct} \times m^3 \log m)$

Table 7.2: Computational cost for each interaction type.

The communication cost for the hypercube communication scheme is discussed in detail in (93). For an uncongested network, that work provides a worst case complexity which scales as $\mathcal{O}(N_s(q^3 + m^2)\sqrt{p})$, where N_s is the maximum number of shared octants per processor. However, assuming that the messages are

evenly distributed across processors in every stage of the hypercube communication process, we get a cost of $\mathcal{O}(N_s(q^3+m^2)\log p)$. In our experiments with uniform octrees, the observed complexity agrees with this estimate. Also, since shared octants are near the boundary of the processor domains, we have $N_s \sim (N_{oct}/p)^{2/3}$, where N_{oct} is total number of octants.

In the uniform octree case, there are $N_{oct} = N/q^3$ total octants, $N_U = 27N_{oct}$ and $N_V = 189N_{oct}$. Due to the large constant factors, the cost of U-list and V-list interactions dominate over other interactions and the overall cost is:

$$T_{\text{FMM}} = \mathcal{O}\left(q^3\frac{N}{p}\right) + \mathcal{O}\left(\frac{m^3}{q^3}\frac{N}{p}\right) + \mathcal{O}\left(\left(\frac{N}{p}\right)^{2/3} q\log p\right).$$

In our results, we also report the time for setup (tree construction and 2:1 balance refinement). For tree construction, the cost of Chebyshev approximation for $\mathcal{O}(N/q^3)$ octants, distributed across p processors is $\mathcal{O}(qN/p)$. During the adaptive refinement, we may also need to repeatedly redistribute octants across processors. This communication cost is data dependent and is difficult to analyze. For the 2:1 balance refinement, the cost is $\mathcal{O}\left(qN/p + (N\log N)/(pq^3)\right)$ assuming a hypercube interconnect.

7.2.3 Geometric and Algebraic Multigrid

Multigrid (19, 66, 179) is one of the most effective solvers for elliptic operators. It is algorithmically optimal and easy to implement and parallelize for uniform grids. Multigrid consists of two complimentary stages: *smoothing* and *coarse grid correction*. Smoothing involves the application of a (typically stationary) iterative solver to reduce (oscillatory) high-frequency errors. Coarse-grid correc-

tion involves transferring information to a coarser grid through *restriction*, solving a coarse-grid system of equations, and then transferring the solution back to the original grid through *prolongation* (interpolation). The coarse-grid correction eliminates (smooth) low-frequency errors. This approach can be applied recursively to obtain a multilevel system consisting of progressively coarser meshes. The multigrid method for the discretization of (7.1) $A_h u_h = f_h$ amounts to the recursive application of the well-known V-cycle multigrid scheme (Algorithm 5). Here, S is the

Algorithm 5: Multigrid V-Cycle

Pre-smooth:	u_k	$\leftarrow$	$S_k(u_k, f_k, A_k)$
Compute Residual:	r_k	$\leftarrow$	$f_k - A_k u_k$
Restrict:	r_{k-1}	$\leftarrow$	$R_k r_k$
Recurse:	e_{k-1}	$\leftarrow$	$A_{k-1}^{-1} r_{k-1}$
Correct:	u_k	$\leftarrow$	$u_k + P_k e_{k-1}$
Post-smooth:	u_k	$\leftarrow$	$S_k(u_k, f_k, A_k)$

smoother and k denotes the multigrid level. The solve at the coarsest level ($k = 0$), is done using a direct solver.

Multigrid methods can be classified into two categories, geometric and algebraic. The primary difference is that GMG methods rely on the underlying mesh connectivity for constructing coarser multigrid levels, whereas AMG methods are mesh-agnostic and work directly on the fine-grid matrix. The advantages of AMG are that it can be used as a black-box algorithm and does not require geometry or mesh information (only the assembled matrix). Its disadvantages are the communication-intense setup and the increased memory requirement compared to matrix-free geometric multigrid. Advantages of GMG are that it can be used in a matrix-free fashion and that it has low memory overhead. Moreover, operators can easily be modified at different levels, which can be necessary to accommodate

certain boundary conditions. The disadvantages of GMG are that it requires a hierarchy of meshes and that it cannot be used in a black-box fashion.

Our GMG code is an extension of our previous work (163) to support high-order discretizations. For the AMG comparison, we use *ML* (54) from the *Trilinos* Project (73). *ML* implements smoothed aggregation, a variant of AMG, and has shown excellent robustness and scalability. All experiments reported in this chapter for multigrid (AMG and GMG) were performed using a single multigrid V-cycle as a preconditioner for the conjugate gradient (CG) method (175).

For the AMG and GMG scalability experiments, we report times (in seconds) for the following stages:

- *Setup*: setting up the multigrid hierarchy and computing the diagonal of the operator for the Jacobi smoother,
- *Eval*: applying the smoother and in the coarse grid solve, and
- *Comm*: performing restriction and prolongation.
- *Solve*: *Eval* + *Comm*.

In the rest of this section we provide additional details of our GMG implementation, in particular where it differs from (163).

Meshing

Our parallel geometric multigrid framework is based on hexahedral meshes derived from adaptive octrees (22, 149, 163). As the multigrid hierarchy is independent of the discretization order, the construction of the grid hierarchy is identical to (163). Multigrid requires the construction of a hierarchy of meshes, such that every element at level k, is either present at the coarser level $k-1$ or is replaced by an coarser(larger) element. The main steps in building the grid hierarchy are,

Coarsening:

Since we use a Morton-ordered linear representation for the octree, all eight sibling-octants, if present, will occur together. While partitioning the octants across processors, we ensure that all eight sibling-octants are owned by the same processor. Because the partitioning guarantees a single level of coarsening, the actual coarsening algorithm is embarrassingly parallel with $\mathcal{O}(N/p)$ parallel time complexity where N is the total number of elements in the finer octree and p is the number of processes. 2:1 balancing is enforced following the coarsening operation.

Partition:

Coarsening and the subsequent 2:1-balancing of the octree can result in a non-uniform distribution of coarse elements across the processes, leading to load imbalance. The Morton ordering enables us to equipartition the elements by performing a parallel scan on the number of elements on each process followed by point-to-point communication to redistribute the elements.

The reduction in the problem size at successive grids creates some problems from the perspective of load balancing. Since we partition each grid separately to ensure that all processes have an equal number of elements, we can end up with a very small number of elements per core (partitioning 2000 elements across 1000 cores) or even impossible cases (2000 elements across 10,000 cores). The surrogate mesh is generated (at every coarsening step) to facilitate the parallelization of intergrid transfers. While partitioning the surrogate mesh, we dynamically reduce the number of processes that are active at the coarser grid in order to ensure that communication does not dominate the computation. Additional details regarding

our load balancing approach are discussed in (163).

Meshing:

By meshing we refer to the construction of the (numerical) data structures required for finite element computations from the (topological) octree data. In addition to the mesh extracted on the fine grid that is relevant for the simulation as a whole, we extract two meshes per multigrid level. First, we extract a *surrogate* mesh after coarsening and 2:1-balance of the fine mesh. Second, we extract the coarse mesh after repartitioning the surrogate mesh. This way we separate the processor-local numerical restriction (computation phase) from the parallel partition (communication phase). The fine and coarse meshes always contain all information for applying the elliptic operator A_k in addition to the encoding of the partition. The surrogate mesh only contains the encoding of the partition and sizes of the elements. The use of the surrogate meshes for intergrid transfers is further detailed in (163).

Recursion:

To build the next multigrid level in the hierarchy, we repeat the previous steps using the coarse mesh as the new fine mesh. The recursion is stopped when either the required number of multigrid levels have been created or when no further coarsening of the mesh is possible.

Discretization

We use high-order discretizations based on Legendre-Gauss-Lobatto (LGL) nodal basis functions for polynomial orders $1 \leq q \leq 16$. For tensorized nodal basis

functions on hexahedral meshes, the application of elemental matrices to vectors can be implemented efficiently by exploiting the tensor structure of the basis functions, as is common for spectral elements, e.g., (35). This results in a computational complexity of $\mathcal{O}(q^4)$ for the element MatVec instead of $\mathcal{O}(q^6)$ if the element matrices were assembled. Globally, across p processors, using tensor products the work complexity for a MatVec is $\mathcal{O}(Nq/p)$, requiring $\mathcal{O}(N)$ storage[4].

Smoother

We use damped Jacobi smoothing with $\omega = \frac{2}{3}$ for all runs reported in this chapter. Although the performance of the Jacobi smoother deteriorates rapidly at $q > 4$ for variable coefficient problems, it performs well for the constant coefficient Poisson problem. For variable coefficient problems high-order Multigrid, Chebyshev-accelerated Jacobi smoother (19) provides good multigrid convergence up to $q = 16$. The cost of applying the Chebyshev-accelerated Jacobi smoother is similar to the Jacobi smoother, although it does require estimation of the maximum eigenvalues of the system matrix, which has to be done during the setup. For our experiments, we estimated the largest eigenvalue using 10 iterations of the Arnoldi algorithm. This increases the setup cost for multigrid.

Restriction & Prolongation Operators

Since the coarse-grid vector space is a subspace of the fine-grid vector space, any coarse-grid vector v can be expanded independently in terms of the fine and coarse basis vectors,

$$v = \sum_i v_{i,k-1}\phi_i^{k-1} = \sum_j v_{j,k}\phi_j^k, \tag{7.2}$$

[4] as opposed to $\mathcal{O}(Nq^3/p)$ work and $\mathcal{O}(Nq^3)$ storage

where, $v_{i,k}$ and $v_{i,k-1}$ are the coefficients in the basis expansion for v on the fine and coarse grids respectively.

The prolongation operator can be represented as a matrix-vector product (MatVec) with the input vector as the coarse grid nodal values and the output as the fine grid nodal values. The matrix entries are the coarse grid shape functions evaluated at the fine-grid vertices, p_i,

$$P(i,j) = \phi_j^{k-1}(p_i). \tag{7.3}$$

The proof for (7.3) can be found in (149). Similar to matrix-free applications of the system matrix, all operations required for the intergrid operations are done at the element level. As in (149), the restriction operator is the transpose of the prolongation operator.

Complexity

For a q-order discretization with N unknowns, the number of elements in the mesh is $\mathcal{O}(N/q^3)$. Given p processes, the time complexity of building the multigrid hierarchy is $\mathcal{O}(N/(pq^3) + \log p)$. The $\mathcal{O}(N/(pq^3))$ corresponds to the coarsening and meshing operations, which are linear in the number of elements $(N/(pq^3))$. The second term accounts for the creation of the $\log p$ multigrid levels. The complexity of enforcing 2:1-balance is $\mathcal{O}(N/(pq^3)\log N/(pq^3))$ (23). The cost of a `MatVec` is $\mathcal{O}(Nq/p)$. Assuming a uniform grid, we can estimate the communication costs as well: Each coarsening, balancing, and partition-correction call requires additional $\mathcal{O}(\log p)$ time to `MPI_Allgather` the local element count (one long integer). For partitioning and transferring, all elements of a process can po-

tentially be communicated to $\mathcal{O}(1)$ processes. These transfers are implemented using non-blocking point-to-point communications, therefore the communication complexity per process is $\mathcal{O}(N/p)$. Thus the complexity for the GMG solve (without setup) is

$$T_{\text{GMG}} = \mathcal{O}\left(\frac{Nq}{p}\right) + \mathcal{O}\left(\log p\right).$$

Caveats

Although our GMG code is capable of handling complex geometries, the version used in this comparison is optimized for a unit cube domain with periodic boundary conditions. The setup phase, specifically the computation of the diagonal will be significantly more expensive for other domains. In addition, the smoother will also be more expensive due the need to compute the Jacobian of the mapping (this can be traded for pre-computation and additional storage) from the reference element to each element.

7.3 Experimental setup

In this section, we give details on the experimental setup we used to test the methods.

Hardware

The hardware employed for the runtime experiments carried out is the Stampede system at TACC and Titan at ORNL. Stampede entered production in January 2013 and is a high-performance Linux cluster consisting of 6,400 compute nodes, each with dual, eight-core processors for a total of 102,400

CPU-cores. The dual-CPUs in each host are Intel Xeon E5-2680 2.7GHz (Sandy Bridge) with 2GB/core of memory and a three-level cache. The nodes also feature the new Intel Xeon Phi co-processors. Stampede has a 56GB/S FDR Mellanox InfiniBand network connected in a fat tree configuration which carries all high-speed traffic (including both MPI and parallel file-system data). Titan is a Cray XK7 with a total of 18,688 nodes consisting of a single 16-core AMD Opteron 6274 processor, for a total of 299,008 cores. Each node has 32GB of memory. It is also equipped with a Gemini interconnect and 600 terabytes of memory across all nodes.

Experiments

We study cost per unknown and cost per accuracy for two sets of experiments:

- In the first set of tests we consider the ***cost per unknown for a fixed algebraic accuracy*** for all five different methods, both in terms of computation and communication. This is ***not*** equivalent to which method is faster. It just reveals the constants in the complexity estimates, and the effect of communication and overheads. For example, high-order methods have better locality but higher work per unknown, so it is not entirely clear what the effect is in the work per unknown. Also, we only test regular grids, but *GMG and FMM have overheads since they use pointer-based data-structure to support octree discretizations.*

  In Figures 7.4 and 7.5 and Tables 7.3, 7.4, 7.5, and 7.6, we consider weak and strong scalability of the solvers with regard only the application of the

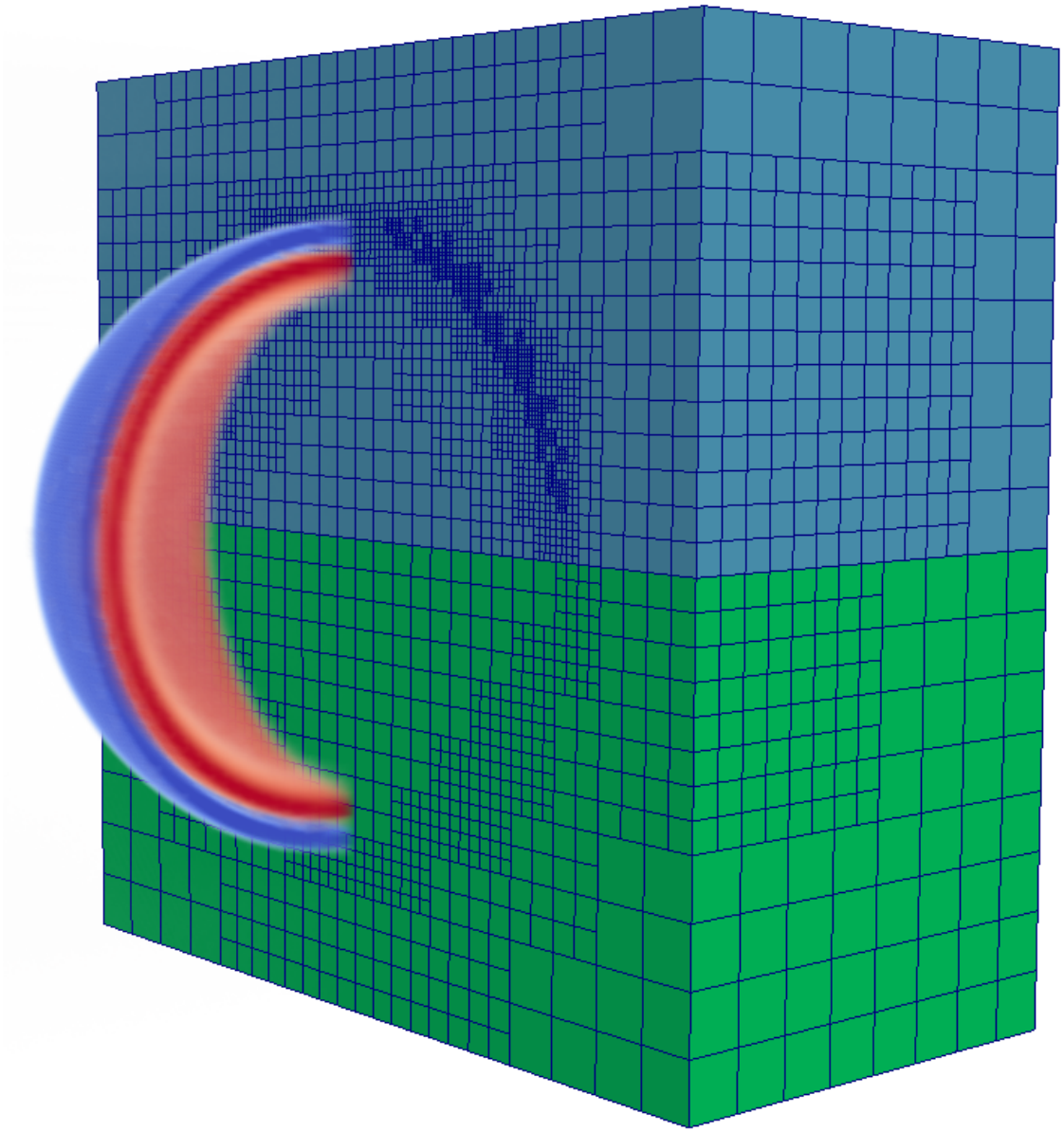


Figure 7.3: Adaptive mesh structure for the forcing term, f, in test case 2. The green cube (bottom quadrant) is the adaptive mesh using 6th order elements and the blue cube (top quadrant) is that of using 14th order elements. The number of unknowns is significantly reduced.

"inverse". For all cases we use an arbitrary right-hand side.

For the FFT, it means an 3D FFT , diagonal scaling to apply the inverse of the Laplacian in the spectral domain, and then another 3DFFT. For the FMM, is a an application of the scheme to the right hand side. For the AMG and GMG runs, we use CG preconditioned with multigrid and we drive the relative residual down to 1E-13. We use the V-cycle (using a full cycle makes little difference). In these runs, on average GMG-1 takes 6 iterations, GMG-4 takes 8 iterations, GMG-8 takes 20 iterations, and GMG-16 takes 28 iterations. These are CG iterations in which the preconditioner is the corresponding multigrid method. For the hpGMG runs we just used the default settings and just controlled the tolerance.

To give an estimate of the relative constants and connect to the complexity estimates we compile statistics for all runs on Stampede and we report the results in Table 7.11.

- In the second set of experiments, ***we examine the time to solution for the different methods***, irrespective of order, or grid. We just set the number of processors and the target accuracy. We report timings for solution error that is 1E-7 or less. We report timings for two different exact solutions u, one that requires uniform discretization (Table 7.7) and one that requires a highly refined discretization (Table 7.8). For the latter, we also consider lower accuracy (Table 7.9) and we also compare with the AMG solver (Table 7.10).

  The first exact u is an oscillatory field given by:

$$u(x_1, x_2, x_3) = \sin(2\pi k x_1)\sin(2\pi k x_2)\sin(2\pi k x_3).$$

This field requires uniform discretization. FFT can resolve u to machine precision using $2k$ points per dimension. First-order methods (for example the 7-point Laplacian or linear finite elements) require roughly $4k$ points per digit of accuracy. Roughly speaking, for two digits of accuracy in 3D, first-order methods require 64× more unknowns than FFT, and for seven digits of accuracy require $2048^3 k$ more unknowns than FFT. Higher-order methods, reduce the number of unknowns significantly but are still suboptimal compared to FFT for this particular u.

The second exact u is $\exp(-(r/R)^\alpha)$, which has a sharp internal layer at the surface of a sphere of radius R, where r is the distance from the point we evaluate u to the center of the sphere. As α increases u develops very sharp gradients around $r = R$ and is almost constant everywhere else. Roughly speaking, every time we double α we increase the spectrum of u by a factor of two and therefore in 3D the number of unknowns for FFT increases by a factor of eight. On the other hand, an adaptive method only refines around the area of the internal layer and can resolve the solution for large α. Figure 7.3 shows an example of an adaptive mesh used by FMM. As a result, it is expected that either FMM or GMG/AMG will be the optimal method for this test case.

For both time-to-solution test cases, the source function is set to the closed form Laplacian of the exact solution, that is $f = -\Delta u$. This f is then used to compute the discretized solution u_N. We report $\|u - u_N\|_{\ell_\infty}$, where we abuse the notation and use u for the exact solution evaluated at the discretization points.

Software parameters

For all our runs FFT means we used AccFFT. For some of the larger Titan runs we have used P3DFFT. For the FMM we have two settings, high accuracy ($q = 14$, $m = 10^2$) and low accuracy $q = 6$ and $m = 4^2$. For FMM the adaptivity criterion is based on the decay of the tail of the Chebyshev expansion of f at every leaf node. For GMG and AMG the adaptivity is based on the gradient of f. For the multigrid case the smoothing is done using a pointwise Jacobi scheme with two pre-smoothing steps and one post-smoothing step. The relative algebraic residual is driven (using CG) to 1E-13 in all runs. The coarse solves were done using sparse parallel LU factorization (SuperLU (100)).[5]

7.4 Results

We are evaluating our geometric multigrid (**GMG**) with $q = 1, 4, 8, 10, 14$, our fast multipole method (**FMM**) with $q = 6, 14$, the **hpGMG** ($q = 1$ and $q = 2$), and finally the (**FFT**) using the AccFFT library. As a reference we also perform some runs with Trilinos's ML algebraic multigrid (**AMG**) library. When using AMG, we are using matrices that are generated by our library using first and fourth order polynomials. AMG is used as a reference point to show the difference between using a state of the art general-purpose solver over a more problem-specific solver. All solvers use only CPU cores; the Phi coprocessors were not used.

We use the following notation to denote the different variants of the methods we are testing: "X-q" indicates method "X" ran with q-th order polynomials. For example "GMG-4" indicates geometric multigrid with fourth order polynomi-

[5] The main parameters for ML are aggregation threshold 0.2, aggregation type : uncoupled, smoother Gauss Seidel (three iterations), and the coarse grid solver is KLU.

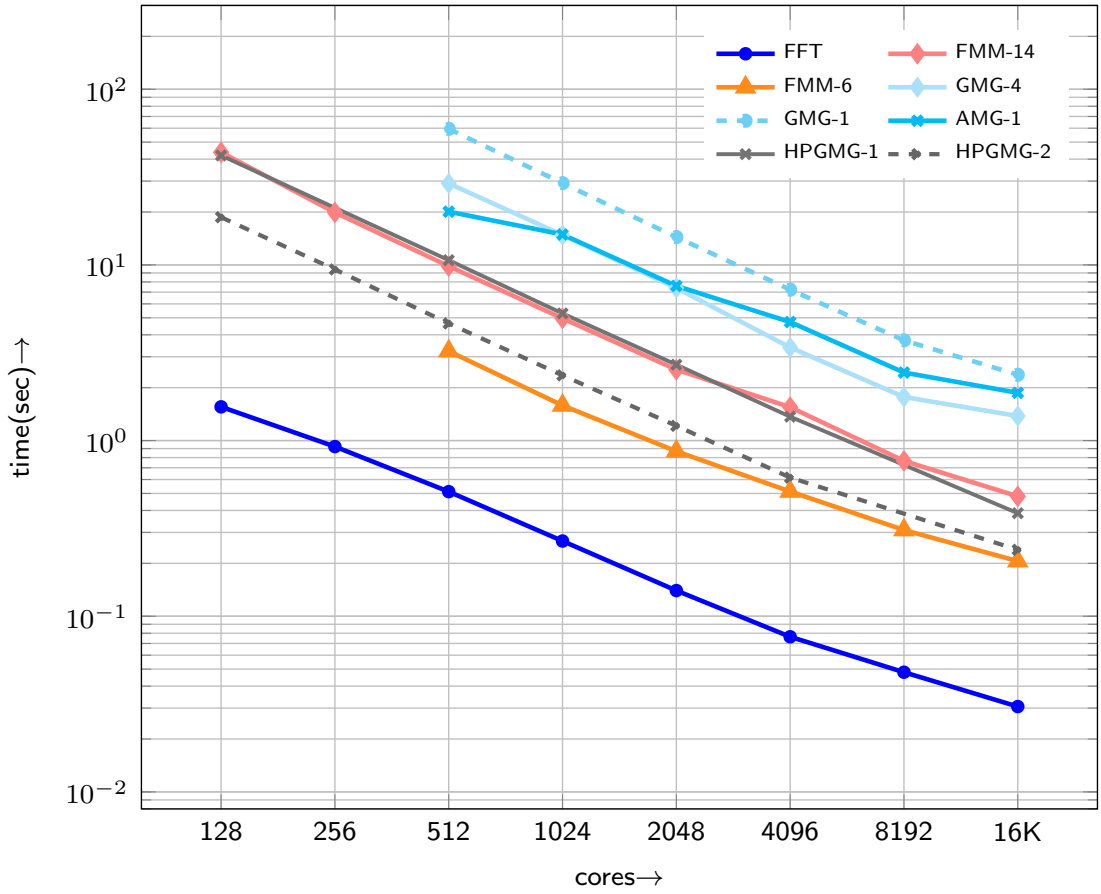


Figure 7.4: Here we report strong scaling results on Stampede that capture the cost per unknown for the different methods. These are only the "Solve" times compared to Table 7.3. For the FFT and FMM this involves just a single evaluation. For the GMG and AMG methods it involves several multigrid sweeps to bring the relative algebraic residual to zero (up to double precision). All times are in seconds for a problem with $N = 1024^3$. The parallel efficiency for the different methods is: FFT (52%), FMM-14 (64%), AMG-1 (33%), GMG-4 (67%), HPGMG-1 (96%), and HPGMG-2(94%). In absolute terms FFT has always the lowest constant per unknown. We can observe orders of magnitude differences between the different variants. Since FMM involves just an evaluation lower order has a smaller constant than higher. Notice also that for our implementations, FMM with high order is significantly faster than GMG with low order. Although, we don't report the results in detail here, we also tested AMG-4 and it was 2× slower than AMG-1. Note that HPGMG is specifically optimized for uniform discretization, and thus is faster compared to our GMG library which can also handle adaptive high-order multigrid.

als; "AMG-1", indicates algebraic multigrid with linear polynomials.

Our first test is a weak and strong scaling test on Stampede and Titan to assess the cost per unknown as a function of the method and the problem size without regard to the accuracy. The strong scaling results are reported in Tables 7.3 (Stampede) and 7.5 (Titan) and the weak scaling results are reported in Tables 7.4 (Stampede) and 7.6 (Titan). All these results where done using uniform discretization so that we could compare with FFT. FMM and GMG have overhead costs since they are not optimized specifically for regular grids.

The second set of experiments examine the time to solution, which are of course the ones that fully characterize the performance of these solvers. The results for the oscillatory u are reported in Table 7.7 and the results for the non-uniform u are reported in Table 7.8.

We also report wall-clock times of "*Setup*", "*Comm*" and "*Solve*". The "*Setup*" involves costs that incur while building data-structures whenever a change in the number of unknowns or their spatial distribution takes place. For the GMG, AMG, building the grid hierarchy, 2:1 balancing it, computing coefficients for the Chebyshev smoother, and setting-up the sparse direct solver for the coarse solve. For the FMM, it involves loading precomputed near-interaction translation operators from the disk to main memory, computing far-field interaction operators, allocating memory buffers, building the octree and its 2:1 balance refinement. For right-hand sides that require the same number of unknowns but different trees due to different distribution of spatial features, the setup has to be called again. For the FFT the setup phase involves parameter tuning to optimize CPU performance. "*Solve*" is the time required to evaluate the solution u whenever a new right-hand side f is specified (without changing the resolution); and "*Comm*" is the distributed

p	phase	AMG-1	GMG-1	GMG-4	HPGMG-1	HPGMG-2	FMM-14	FFT
1024	Setup	5.4	1.5	0.3	-	-	5.2	5.9
	Comm	-	2.8	2.9	-	-	0.2	0.1
	Solve	15	26.2	11.8	5.3	2.3	5.0	0.3
4096	Setup	4.1	0.7	0.3	-	-	3.9	2.5
	Comm	-	0.9	0.9	-	-	0.1	0.05
	Solve	4.7	6.3	2.5	1.4	0.6	1.5	0.08
16384	Setup	15	0.5	0.4	-	-	5.2	1.4
	Comm	-	0.9	0.8	-	-	0.1	0.02
	Solve	1.9	1.6	0.6	0.4	0.2	0.5	0.03

Table 7.3: Here we report strong scaling results on Stampede. We report representative breakdowns of the timings (in seconds) for uniform grids with $N = 1024^3$. GMG has insignificant setup costs. AMG has a higher cost as it needs to build the multigrid hierarchy algebraically using graph coarsening. Finally, FMM has setup costs that are in the order of the solve time; these costs are related to tree construction, loading (from the disk) precomputed tables, 2:1 balancing, and constructing the interaction lists. The precomputed tables are generic and valid for both uniform and non-uniform grids. The setup phase for FMM has fixed size overheads associated with loading and computing interaction operators and therefore, the setup time does not scale as we increase the number of processor cores. Note that HPGMG is specifically optimized for uniform discretization, and thus is faster compared to our GMG library which can also handle adaptive high-order multigrid.

memory communication costs during the "*Solve*" phase. For AMG library we do not report communication costs, as we could not find an easy way to measure it (as of version ML 5.0). As we mentioned in the introduction more detailed analysis of the three algorithms (along with discussion on single-core performance) can be found in (106, 135, 163).

Strong scaling analysis on Stampede, uniform grid

To demonstrate the overhead and performance characteristics of the methods, we consider strong scaling for a problem with one billion unknowns (1024^3). We report "Solve" time in Figure 7.4; and the "Setup" and "Comm" times in Table 7.3. We report "Solve" time separately since in many cases setup time is amortized across nonlinear iterations or time-stepping. As one can see, FFT is clearly the fastest method, although FMM is not far behind. Here AMG-1, despite hav-

p	phase	AMG-1	GMG-1	GMG-4	HPGMG-1	HPGMG-2	FMM-14	FFT
128	Setup	4.9	1.7	0.1	-	-	4.1	5.0
	Comm	–	2.0	2.1	-	-	0.2	0.1
	Solve	14.9	32.9	15.8	5.4	2.4	4.9	0.2
1024	Setup	5.4	1.5	0.3	-	-	4.5	5.9
	Comm	–	2.8	3.0	-	-	0.2	0.16
	Solve	14.9	32.2	16.9	5.3	2.3	5.1	0.26
8192	Setup	6.5	1.7	0.5	-	-	5.6	8.0
	Comm	–	4.3	4.6	-	-	0.3	0.22
	Solve	15.2	35.7	17.9	7.6	3.4	5.3	0.34

Table 7.4: Here we report weak scaling results on the Stampede system. In this test, we keep the number of unknowns per process fixed as we increase p. The problem sizes that used are $N = 512^3,\ 1024^3,\ 2048^3$ for $p = 128,\ 1024,\ 2048$ respectively. For the AMG and HPGMG runs, we could not directly measure the communication costs.

p	**phase**	**FMM-14**	**P3DFFT**
$57,344$	Setup	105	—
	Comm	14.4	—
	Solve	230	44.2
$114,688$	Setup	44.4	—
	Comm	6.3	—
	Solve	105	19.1
$229,376$	Setup	30	—
	Comm	4.2	—
	Solve	58	11.3

Table 7.5: Strong scaling on Titan (for FMM only). We report wall-clock time in seconds for FMM with 14th order polynomials for a problem with 600 billion unknowns. The efficiency is 99% for the FMM "Solve" phase. The communication costs for the FMM are order-of-magnitude smaller than the "Solve" and "Setup" phases.

ing less computation per unknown is somewhat slower due to the cost per V-cycle and the number of sweeps. All the methods scale quite well. Also notice that GMG-4 is faster than GMG-1 due to better compute intensity. Same is true for HPGMG-2 compared to HPGMG-1. Note that HPGMG is specifically optimized for uniform discretization, and thus is faster compared to our GMG library which can also handle adaptive high-order multigrid. The FFT setup time only depend on the problem size and number of processors and thus is done just once during the course of a calculation. In most applications it is unusual to change resolution

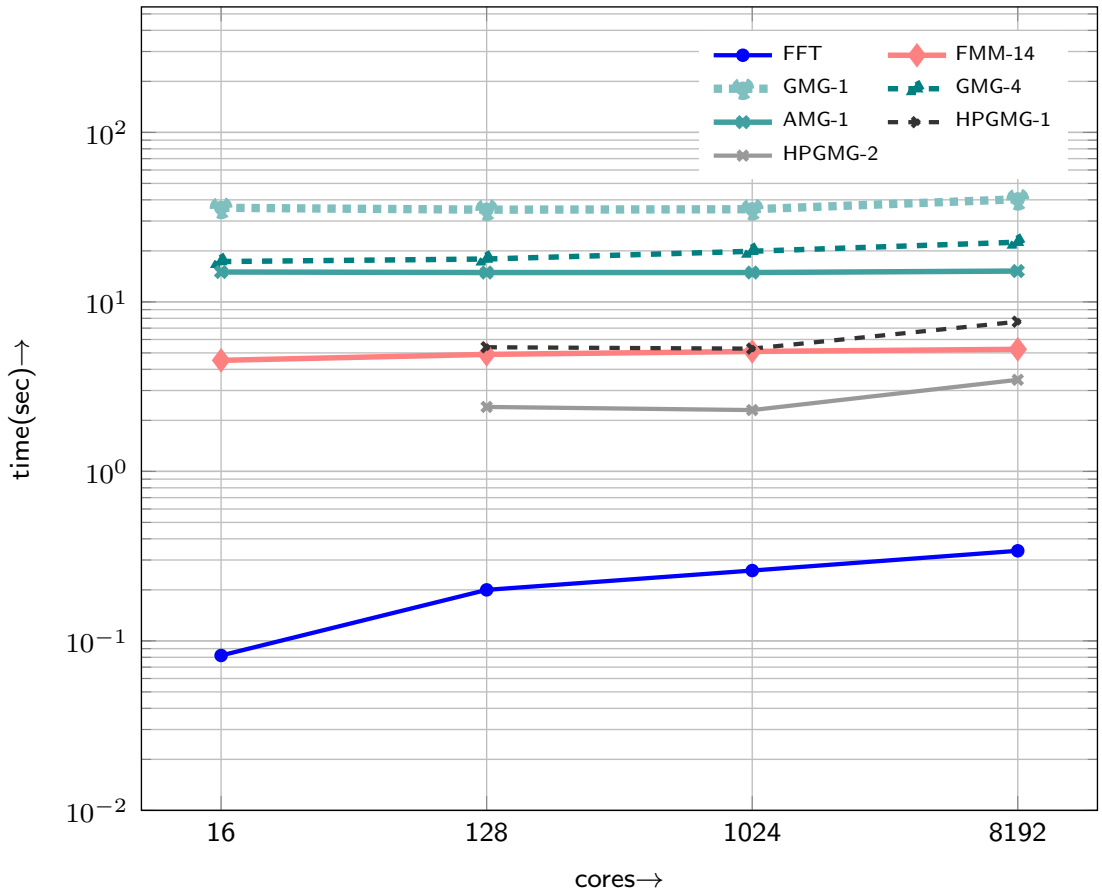


Figure 7.5: Weak scaling on Stampede: We report "Solve" time in seconds of different methods for a problem with a grain size of one million unknowns per core; the largest problem on Stampede has 8.5 billion unknowns on 8,192 cores. All the methods scale quite well. FFT is the fastest and the difference is over 30× over GMG-1.

dynamically when using FFTs.

Weak scaling on Stampede

To illustrate the constant factors in the complexity estimates for the different methods, we consider a weak scaling test in which we keep the number of unknowns per core fixed to roughly one million and we increase N and p, keeping their ratio fixed Figure 7.5. The observed end-to-end efficiencies for the different methods are as follows: FFT(59%), FMM-14(95%), GMG-1(92%), GMG-4(88%), AMG-1 (98%), HPGMG-1 (69%), and HPGMG-2 (70%). FFT does not scale as well as other methods but it is still quite faster. It is worth noting that although GMG-4 does not scale as well as GMG-1, but it is about two times faster. This is due to the higher order scheme used in GMG-4.

Strong and weak scaling on Titan

On Titan we have performed weak scaling with FFT, and FMM-14. The low-order versions for these methods have already scaled to 260K cores on Jaguar (163).)

The strong scaling results are summarized in Table 7.5 for a problem with 600 billion unknowns for the FMM.

The weak scaling results for FFT and FMM are shown in Table 7.6. We observe that the FMM is almost 25× slower than the FFT. This is because FFT has much less work per unknown. Also it is "more" memory bound and less sensitive to lower per-node peak performance (which is the case for Titan CPUs compared to Stampede CPUs) so FMM runs slower on Titan whereas FFT is not affected as much.

N	p	phase	FMM-14	AccFFT	P3DFFT
$1,024^3$	448	Setup	16.0	12.0	—
		Comm	2.5	1.3	—
		Solve	51.8	2.2	4.5
$2,048^3$	3,584	Setup	16.7	15.0	—
		Comm	2.7	1.9	—
		Solve	52.5	2.6	6.0
$4,096^3$	28,672	Setup	17.6	—	—
		Comm	3.6	—	—
		Solve	53.1	—	8.0
$8,192^3$	229,376	Setup	19.0	—	—
		Comm	4.2	—	—
		Solve	57.8	—	11.3

Table 7.6: Weak scaling results on Titan. We report wall-clock time in seconds for the FFT and FMM with 14th order polynomials. The largest problem size corresponds to half a trillion unknowns. FFT is significantly faster than FMM.

Time to solution weak scaling

So as we saw in the previous paragraphs, one can test these solvers using the same number of unknowns across solvers, where significant differences were observed. However, what matters most is time to solution. Once this is factored in, the differences between the solvers become more pronounced.

We start by looking at test case 1 (the oscillatory synthetic problem), a problem that requires uniform refinement. We report the "Solve" time to reach single precision accuracy in Table 7.7. From this first experiment, it is clear that FFT is extremely efficient for this problem. For example, for an $8,192$-core run, if we retain the same grid and we change k from 128 to 256, the errors for FMM and GMG drop to three digits of accuracy whereas FFT can still resolve it to machine precision (as a reminder k is the frequency of the f). This is of course expected given the spectral accuracy of the FFT and the very low work per unknown required by the algorithm. The purpose of the experiment is to highlight the performance gap even if we use very high order FMM or Galerkin methods.

		GMG-14			FMM-14			FFT	
p	k	T	ℓ_∞	L	T	ℓ_∞	L	T	L
16	16	10	1E-8	4	4.5	3E-7	5	<0.01	6
128	32	9.8	4E-8	5	4.9	3E-7	6	<0.01	7
1,024	64	10	1E-7	6	4.9	4E-7	7	<0.01	8
8,192	128	10	3E-7	7	5.1	5E-7	8	<0.01	9

Table 7.7: Time-to-solution for test case 1 (oscillatory field) performed on Stampede: Here we report the number of cores p, the effective resolution (wavenumber) k, the relative discrete infinity norm ℓ_∞ of the error in u, "Solve" time "T" and the uniform refinement level "L", required to achieve single-precision (seven digits) accuracy or better. For the FFT we do not report errors since it resolves the solution to machine accuracy. FFT dramatically outperforms FMM and GMG, since it can resolve the problem with considerably fewer unknowns. FFT requires 8^L unknowns to resolve f. FMM requires $8^L q^3/6$ (and GMG $8^L q^3$) or roughly 9 billion unknowns to resolve a u with $k = 128$ to single precision; FFT can resolve $k = 256$ to machine precision with nearly 70× fewer points. Also note that the target precision doesn't change the conclusions, it just determines the overall problem size.

		GMG-5		FMM-14		FFT	
p	α	T	N	T	N	T	N
32	10	4.4	4.4E+6	0.5	2.9E+6	0.1	1.7E+7
512	40	5.0	1.8E+7	0.7	6.4E+7	0.5	1.1E+9
8,192	160	6.0	2.5E+8	0.8	1.1E+9	2.6	6.9E+10
32,768	320	6.6	9.8E+9	1.8	4.3E+9	4.9	5.5E+11

Table 7.8: Time-to-solution for test case 2 performed on Stampede: Here α is a measure of the difficulty of resolving u. The higher the α value the sharper the derivatives. We report "Solve" time "T" and the required number of unknowns N to achieve single precision accuracy or higher. In all of these runs and for all three methods the relative error ℓ_∞ is less than $4E-7$. We used GMG-5, because in our test it was the fastest variant of GMG (see Tab. 7.10).

	L	N	ℓ_∞	T
GMG-1	12	9.5E+6	9.4E-3	9.40
GMG-2	9	1.0E+6	5.3E-3	1.60
GMG-5	6	4.1E+5	2.2E-3	3.78
FMM-7	8	2.7E+6	3.5E-3	0.13
FFT	–	5.6E+7	8.1E-3	0.07

Table 7.9: Time-to-solution for test case 2, using 32 cores on Stampede. We set $\alpha = 40$. We report "Solve" time "T" and the required number of unknowns N and tree refinement level "L" to achieve about two digits of accuracy or higher. For the multigrid-preconditioned runs, CG took 2,3, and 5 iterations for the V-cycle GMG-1, GMG-2, and GMG-5 schemes.

p	q	N	T (AMG)	T (GMG)
32	4	8.9E6	29.3	8.6
32	5	4.4E6	27.7	4.4
32	6	2.1E6	22.8	4.9
32	8	6.0E5	51.4	6.1

Table 7.10: Corresponding results in Tab. 7.8 with $\alpha = 10$ using AMG/GMG with different orders. As the order is increased up to 6th order, the cost per unknown reduces. However, further increase in the order increases this cost and thus the time to solution increases.

But what about solutions with localized features as in test case 2? We report the "Solve" time $u = \exp(-(r/R)^{\alpha})$ for Table 7.8 as a function of α. The larger the α the more localized features u has. The performance of the different solves is now quite different. FFT works well for relatively small values of α but as we make the solution sharper (larger α) it cannot resolve the length-scales of the solution and the uniform grid size shoots up; for the largest problem size FFT requires $100\times$ more unknowns than those required by FMM and GMG. Furthermore, FMM outperforms multigrid but the difference is not as dramatic. Understanding this requires a lengthier analysis and is sensitive to the quality of per core performance. In passing, let us mention that the adaptive multigrid is quite efficiently implemented. Both codes used dense matrix-matrix multiplication kernels (that is they used BLAS DGEMM routine) but their intensity is quite different. GMG uses a high order tensor basis. The main computational kernel for GMG is the element traversal to apply the local stiffness matrix to a vector. This kernel is decomposed to 1D slices to reduce the complexity. The decomposition involves more integer operators and multiple DGEMM invocations. For near interactions FMM truncates the tensor basis and uses a single DGEMM of larger size than GMG for the near interaction. The far interaction is much more involved and briefly summarized in §7.2.2.

Method	Communication		Computation	
	T_{COMM}	σ_{COMM}	T_{COMP}	σ_{COMP}
FFT	$1.7\text{E-}7\ \frac{N}{p}$	7.9E-8	$4.2\text{E-}9\frac{NlogN}{p}$	1.2E-9
FMM-14	$5.3\text{E-}6\left(\frac{N}{p}\right)^{2/3}\log p$	2.0E-6	$4.5\text{E-}6\frac{N}{p}$	5.8E-7
GMG-1	$3.2\text{E-}4\left(\frac{N}{p}\right)^{2/3}$	1.3E-4	$2.8\text{E-}5\frac{N}{p}$	3.9E-6
GMG-4	$3.2\text{E-}4\left(\frac{N}{p}\right)^{2/3}$	1.1E-4	$1.3\text{E-}5\frac{N}{p}$	3.2E-6

Table 7.11: *Here we report the constants of the complexity estimates of each method, for the communication time T_{COMM} and the computation time T_{COMP}. The constants were computed using the data of Table 7.3 and 7.4. We also report the standard deviation (σ_{COMM}, σ_{COMP}) for the constants.*

Due to the significance difference between the GMG and FMM kernels, the result is that their floating point performance per core differs. For our implementations FMM outperforms GMG. We also report AMG/GMG timings for different orders in Table 7.10. As expected AMG is significantly slower than GMG. We can see that the time to solution decreases by increasing the order up to $q = 5$, and then increases afterwards. Finally, we would like to remark that both GMG and FMM over-refine to maintain 2:1 balancing.

It would be tempting to use the expressions for $T_{\text{FMM}}, T_{\text{FFT}}, T_{\text{GMG}}$ of §7.2 with N fixed to try to quantify the effect of the order q on the communication and computation costs and how it compares between the methods. However N *does depend* on q and the relation is non-trivial. That's why our empirical results paint a more accurate picture.

Performance of GMG

Our comparisons against the finite element version of hpGMG Table 7.3 indicate that hpGMG performed $4-5\times$ faster than GMG-1. While the performance of hpGMG is impressive and reflective of its quality as a benchmark, our performance is equally strong given that our code supports adaptivity and arbitrarily high or-

ders; hpGMG only supports linear and quadratic elements. We briefly summarize the main reasons for the $\sim 5\times$ speedup observed with hpGMG.

- Given that our code supports adaptive meshes, we need to store additional mesh-lookup information. This reduces the effective flop-to-mop ratio of our code.

- Adaptivity also forces us to have to deal with nodes that do not represent independent degrees of freedom. This increases the cost of the elemental matrix-vector multiplications. The details of how we treat such nodes is beyond the scope of this chapter. Please refer to (149, 163) for two different approaches to handling these cases.

- hpGMG uses a cartesian grid topology, leading to efficient communication. Our GMG code is capable of handling complex geometries in an adaptive fashion and we use a Morton-ordering for partitioning work across processors. This makes it difficult for us to provide any topological information to enable efficient communication.

We also compared the performance of hpGMG for the oscillatory synthetic problem reported in Table 7.7. For the case of $p, k = 16$, using hpGMG with $Q1$ elements, we could accomodate a mesh of size 512^3 requiring 42.3 seconds to reach a relative ℓ_∞ norm of the error in u of 1.1E-2. Using $Q2$ elements, we could accomodate 256^3 elements, to reduce the error to 2.8E-3 in 19 seconds. Assuming perfect convergence rate (error = error/8 for each refinement), we will need 4096^3 elements to reduce the error to 7E-7. Assuming linear scaling, we will need $77K$ seconds, compared to us needing 10 seconds. This example illustrates that performance uniform

grid discretizations deteriorates significantly for problems that have layers and localized features.

7.5 Conclusions

We outlined three solvers developed in our group for Poisson problems with high-order discretizations and we compared them for the Poisson problem with constant coefficients on the unit cube with periodic boundary conditions. To measure the efficiency of our schemes with the current state of the art we also compared our GMGM to implementations developed by other groups, in particular the finite-element HPGMG and AMG. There is no (parallel) software available that compares to our FMM.

- If we fix the number of unknowns, and we consider just the cost per unknown, FFT is the fastest scheme because it has really small constants in its complexity estimates. FMM performs comparably with FFT for the same number of unknowns (on Stampede). Which method is preferable depends on the type of resolution scheme (uniform vs non-uniform). For this reason, one should note that *it is not possible to compare these methods using exactly the same* N. Often in the literature, in particular the HPC literature, such comparisons appear. Without accuracy for a specific problem such information can be misleading.

- FFT outperforms the other methods by very large margins for uniform grids.[6] As expected the communication cost of FFT does not scale as favorably. How-

[6] Note that the hpGMG benchmark includes a finite-volume stencil code for constant coefficients that it is extremely fast and possibly compares well with FFT. Also it is more general since it supports different types of boundary conditions. So our conclusion regarding FFT pertains strictly to the packages we used and it is not general.

ever, FFT requires far fewer unknowns for functions that are within its resolution, and as we mentioned it also has the smallest constants. So even if the communication scaling is suboptimal, overall the method is much faster. We report results up to 230K cores that confirm this.

- For fields that have sharp local features regular-grid discretizations lose their advantage. Adaptively refined algorithms like GMG and FMM can resolve such fields more effectively since they require orders of magnitude fewer unknowns than FFT.

- GMG, hpGMG, or AMG based on linear elements are significantly slower compared to high-order methods when the accuracy requirements are high. The excessive number of unknowns can result in orders of magnitude in performance penalties. We expect this to hold for problems for which the solution has localized discontinuities.

- For non-uniform grids, both FMM and GMG methods scale quite well since they are matrix free and both use octree-based hierarchical space decomposition methods. The structure of the calculation however is different, with the FMM offering significant opportunities for task parallelism and compute intensity. On the other hand, the setup of FMM is more expensive. Our GMG is much more general than the FMM since it supports more general geometries and boundary conditions. All these incur overheads.

- A state of the art algebraic solver is orders of magnitude slower than the fastest variants of FFT, FMM and GMG. Of course, AMG is much more general as it can deal with arbitrary geometries, variable and anisotropic coefficients, and more general boundary conditions. However, many important

applications require fast Poisson solvers on a cube with boundary conditions that FFT, FMM, or GMG can handle. Therefore, using a method designed for more general situations can incur quite high performance penalties.

- All three methods can be further improved: by integrating tightly with accelerators, further improving single-node performance, and trying to improve the per leaf high-order calculations (for FMM and GMG).

- No method rules the others. All methods have regimes in which they are the fastest. A robust black-box solver should be a hybrid method for cases when the true solution is a superposition of a highly oscillatory field and a highly localized field. This is specially true for problems with more complex geometries and boundary conditions.

Acknowledgment

The authors would like to thank Omar Ghattas and Georg Stadler for useful discussions on this topic. Also, we are grateful to ORNL and TACC for help with the experiments on the Titan and Stampede systems. Support for this work was provided by: the U.S. Air Force Office of Scientific Research (AFOSR) Computational Mathematics program under award number FA9550-09-1-0608; the U.S. Department of Energy Office of Science (DOE-SC), Advanced Scientific Computing Research (ASCR), DE-FG02-09ER25914, and the U.S. National Science Foundation (NSF) Cyber-enabled Discovery and Innovation (CDI) program under awards CMS- 1028889 and OPP-0941678, and the PetaApps program under award OCI-0749334. Computing time on the Cray XK7 Titan was provided by the Oak Ridge Leadership Computing Facility at Oak Ridge National Laboratory, which is sup-

ported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725. Computing time on the Texas Advanced Computing CenterâĂŹs Stampede system was provided by an allocation from TACC and the NSF.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

In this thesis, we presented algorithms and software for parameter estimation for tumor growth problems and diffeomorphic image registration inverse problem. We discussed how our framework can be used for parameter estimation/calibration of tumor models, and fast automatic image registration of healthy images to tumor-bearing medical images.

We first discussed the inverse tumor problem in which we solve an optimization problem to compute/estimate the extent of tumor infiltration into the brain, beyond what is visible in imaging, and possibly invert for other model parameters such as the diffusion coefficient. We discussed how our framework can be used for multiple anisotropic diffusion models, and multi-focal tumors. We presented the optimality conditions and explained how we solve the problem using a second-order, reduced-Hessian approach. We tested the performance of the framework, using synthetically generated data. We discussed the parallel scalability the distributed-memory implementation of the solver on multiple systems. We presented the weak and strong scaling of the whole inversion as well as its components. We then discussed how this reaction-diffusion model should be coupled to linear elasticity equations in order to capture the so called "mass-effect" of gliomas. We discussed the formulation and derived its first-order optimality conditions.

Then we introduced the first parallel 3D framework for large diffeomorphic image registration using a Newton-Krylov method. We discussed the mathematical

formulation for the registration problem and presented its optimality conditions. Solving the registration problem (especially in 3D) is very time consuming. The main computational kernels are the FFT, used for spatial differential operators in our spectral approach, and the interpolations used in the semi-Lagrangian scheme for advection.

We addressed this by developing a novel parallel semi-Lagrangian solver with application to a wide range of transport problems. We discussed the distributed-memory algorithm and single node optimizations for a cubic Lagrange interpolation scheme, that is is nearly 10× faster than the previous state-of-the-art. We discussed the optimization which included Morton reordering of departure points, blocking, and SIMD vectorization for different architectures (Haswell and KNL).

We introduced AccFFT, a new parallel FFT library for CPU and GPUs. We discussed the pencil and slab decomposition algorithms for computing forward and backward FFTs in a distributed-memory setting. We discussed the design of the library, its ghost point communication strategy (used in the semi-Lagrangian scheme), and its specific optimizations for multiple spectral operators that can significantly reduce the communication volume. We presented weak and strong scaling results on multiple systems, up to 4,096 GPUs and 131K cores for both single and double precision transforms.

Using these optimized frameworks, we studied the scalability of our registration and tumor solvers. We showcased results for unprecedented registration problems, 4096^3 resulting in ~ 200 billion unknowns—a problem size that is $64\times$ larger than the state-of-the-art. For the problem sizes of clinical interest, our framework is about 8× faster than the state-of-the-art.

We then addressed the challenge created by lack of data from the patient's brain

prior to the tumor growth (or equivalently the challenge created by having only one time snapshot of the tumor). In a clinical setting, the input consists of multimodal magnetic resonance images, without any classification of the voxels (segmentation) as well. We addressed this lack of data by solving a coupled inverse registration and tumor problem. We discussed the coupled formulation and derived its optimality conditions. We described a Picard iteration scheme for solving the resulting equations, and tested the performance of our method on synthetically generated data. We tested the performance of our framework, even when we do not have the ground truth tumor models used to generate the test case.

We also performed a detailed scaling study and compared FFT, with FEM and FMM for solving a Poisson problem in a unit cube. We examined and reported results for weak/strong scaling, and time to solution for uniform and highly refined grids. We presented results on the Stampede system at the Texas Advanced Computing Center and on the Titan system at the Oak Ridge National Laboratory. In our largest test case, we solved a problem with 600 billion unknowns on 229,379 cores of Titan. We tested all of the methods with different source functions (the right hand side in the Poisson problem). Our results indicated that FFT is the method of choice for smooth source functions that require uniform resolution. However, FFT loses its performance advantage when the source function has highly localized features like internal sharp layers. FMM and GMG considerably outperform FFT for those cases. The distinction between FMM and GMG is less pronounced and is sensitive to the quality (from a performance point of view) of the underlying implementations. In most cases, high-order accurate versions of GMG and FMM significantly outperform their low-order accurate counterparts.

8.2 Future Work

8.2.1 Image Segmentation:

The tumor inverse solver and its coupling with registration, requires segmentation of the tumor-bearing image of the patient as input. Image segmentation is the process in which we assign labels to each voxel from different categories (e.g. white matter, gray matter, cerebrospinal fluid, enhancing and non-enhancing tumor, necrosis, and edema). Typically, medical image segmentation is harder compared to landscapes, shape recognition, or motion since the input data is noisy, contains multi-modal imaging in 3D, and each patient has unique anatomical structures and unique tumor. Although, it is possible to get a manual segmentation of the tumorous brain images from experts, but that is time consuming (expensive), and studies have show significant variability across experts (117). For population studies, reproducible analysis, and robustness, such segmentation must be produced automatically. Existing state-of-the-art algorithms for automatic segmentation of neurooncology MRI, are based on coupling tumor growth models and image-registration to machine-learning based classification algorithms (60).
We can get an automatic segmentation of the brain using supervised learning, which has been studied previously in our group (187). In this approach, the goal is to find a labeling field given a set of manually segmented brain images (as training data), that can "best" describe the observed image intensities. One approach to finding one such field, is to use a Bayesian framework and find the Maximum a Posteriori (MAP) point of the posterior probability, which is the probability of the observed image intensities in each voxel given the labeling field, multiplied by the prior information of the labeling (187). One limitation of supervised machine

learning methods applied to tumor-bearing image segmentation, is that they are not informed by the mathematical models of the tumor growth. Moreover, there is very few training data for gliomas, which makes it very hard to train the classifier without over-fitting. Our coupling tumor and image registration framework can be used to produce automatic, multi-modal segmentation of tumor-bearing images, in conjunction to the above supervised learning algorithm. In this approach, we first compute the above MAP point and assign an initial labeling to the patient's image. Given this labeling field, we then solve the coupling problem, and transport the atlas's segmentation into the patient domain, which gives us the model-informed correction to the starting labeling. This process is then repeated until convergence. This is an on-going research.

8.2.2 Multi-species tumor models:

Most tumors consist of multiple cancer cells phenotypes. These include edema, enhancing and non-enhancing tumor, necrosis, and angiogenesis. One attempt to modify the model is a Go-or-Grow model, which involves two species: proliferating cancer cells that do not migrate, and infiltrating cells that do not proliferate (136).

An extension to this model is necrosis and nutrition consumption of the cancer cells (150). Although, not discussed in this thesis, but our framework has been extended to support these two models in MATLAB, and we are now working on coupling it with linear elasticity equations and implementing the distributed memory version of it in our parallel C++ implementation.

8.2.3 Uncertainty Quantification:

It is very important to study the uncertainty in the inversion parameters using Bayesian analysis. This is very important since the input to the inversion framework is often times very noisy, and it is important to study how this noise would affect the solution (for instance by computing the variance of the posterior). This information can be very useful specially in treatment planning of the tumors.

Bibliography

[1] 2017. CLFFT. https://github.com/clMathLibraries/clFFT. (2017).

[2] S.J. Aarseth. 2003. *Gravitational N-body simulations*. Cambridge Univ Pr.

[3] Mark Adams, Jed Brown, John Shalf, Brian Van Straalen, Erich Strohmaier, Richard Vuduc, and Sam Williams. 2014. High-performance Geometric Multigrid. (2014). https://hpgmg.org

[4] E D Angelini, O Clatz, E Mandonnet, E Konukoglu, L Capelle, and H Duffau. 2007. Glioma dynamics and computational models: A review of segmentation, registation, in silico growth algorithms and their clinical applications. *Curr Med Imaging Rev* 3, 4 (2007), 262–276.

[5] Axel Arnold, Florian Fahrenberger, Christian Holm, Olaf Lenz, Matthias Bolten, Holger Dachsel, Rene Halver, Ivo Kabadshow, Franz Gähler, Frederik Heber, and others. 2013. Comparison of scalable fast methods for long-range interactions. *Physical Review E* 88, 6 (2013), 063308.

[6] J. Ashburner and K. J. Friston. 2011. Diffeomorphic registration using geodesic shooting and Gauss-Newton optimisation. *NeuroImage* 55, 3 (2011), 954–967.

[7] Orlando Ayala and Lian-Ping Wang. 2013. Parallel implementation and scalability analysis of 3-D Fast Fourier Transform using 2d domain decomposition. *Parallel Comput.* 39, 1 (2013), 58–77.

[8] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes,

K. Rupp, B. F. Smith, S. Zampini, and H. Zhang. 2016. *PETSc users manual*. Technical Report ANL-95/11 - Revision 3.7. Argonne National Laboratory.

[9] Satish Balay, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, and Hong Zhang. 2014. PETSc Web page. (2014). http://www.mcs.anl.gov/petsc

[10] Wolfgang Bangerth, Ralf Hartmann, and Guido Kanschat. 2007. Deal.II – a General Purpose Object Oriented Finite Element Library. *ACM Transactions in Mathematical Software* 33, 4 (2007), 24/1–24/27.

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

[12] M. F. Beg, M. I. Miller, A. Trouvé, and L. Younes. 2005. Computing large deformation metric mappings via geodesic flows of diffeomorphisms. *International Journal of Computer Vision* 61, 2 (2005), 139–157.

[13] Nicola Bellomo, NK Li, and Ph K Maini. 2008. On the foundations of cancer modelling: selected topics, speculations, and perspectives. *Mathematical Models and Methods in Applied Sciences* 18, 04 (2008), 593–646.

[14] Abhinav Bhatele, Pritish Jetley, Hormozd Gahvari, Lukasz Wesolowski, William D Gropp, and Laxmikant Kale. 2011. Architectural constraints to attain 1 Exaflop/s for three scientific application classes. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*. IEEE, 80–91.

[15] L. T. Biegler, O. Ghattas, M. Heinkenschloss, and B. van Bloemen Waanders. 2003. *Large-scale PDE-constrained optimization*. Springer.

[16] Pierre-Yves Bondiau, Olivier Clatz, Maxime Sermesant, Pierre-Yves Marcy, Herve Delingette, Marc Frenay, and Nicholas Ayache. 2008. Biocomputing: numerical simulation of glioblastoma growth using diffusion tensor imaging. *Physics in Medicine and Biology* 53, 4 (2008), 879.

[17] A. Borzì, K. Ito, and K. Kunisch. 2002. Optimal control formulation for determining optical flow. *SIAM Journal on Scientific Computing* 24, 3 (2002), 818–847.

[18] A. Borzì and V. Schulz. 2012. *Computational optimization of systems governed by partial differential equations*. SIAM, Philadelphia, Pennsylvania, US.

[19] Achi Brandt. 1977. Multigrid adaptive solution of boundary value problems. *Mathematics of Computations* 13 (1977), 333–390.

[20] Oscar P. Bruno and Leonid A. Kunyansky. 2001. A Fast, High-Order Algorithm for the Solution of Surface Scattering Problems: Basic Implementation, Tests, and Applications. *J. Comput. Phys.* 169 (2001), 80–110.

[21] Alfonso Bueno-Orovio, Víctor M Pérez-García, and Flavio H Fenton. 2006. Spectral methods for partial differential equations in irregular domains: the spectral smoothed boundary method. *SIAM Journal on Scientific Computing* 28, 3 (2006), 886–900.

[22] Carsten Burstedde, Omar Ghattas, Michael Gurnis, Tobin Isaac, Georg Stadler, Tim Warburton, and Lucas C. Wilcox. 2010. Extreme-Scale AMR. In *SC10: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM/IEEE.

[23] Carsten Burstedde, Lucas C. Wilcox, and Omar Ghattas. 2011. p4est: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees. *SIAM Journal on Scientific Computing* 33, 3 (2011), 1103–1133.

[24] J Carrier, Leslie Greengard, and Vladimir Rokhlin. 1988. A fast adaptive multipole algorithm for particle simulations. *SIAM journal on scientific and statistical computing* 9, 4 (1988), 669–686.

[25] K. Chen and D. A. Lorenz. 2011. Image sequence interpolation using optimal control. *Journal of Mathematical Imaging and Vision* 41 (2011), 222–238.

[26] Hongwei Cheng, Leslie Greengard, and Vladimir Rokhlin. 1999. A Fast Adaptive Multipole Algorithm in Three Dimensions. *J. Comput. Phys.* 155 (1999), 468–498.

[27] 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. 2006. Introduction to the non-rigid image registration evaluation project. In *Proc Biomedical Image Registration*, Vol. LNCS 4057. 128–135.

[28] Olivier Clatz, Maxime Sermesant, P-Y Bondiau, Hervé Delingette, Simon K Warfield, Grégoire Malandain, and Nicholas Ayache. 2005. Realistic simulation of the 3-D growth of brain tumors in MR images coupling diffusion with biomechanical deformation. *Medical Imaging, IEEE Transactions on* 24, 10 (2005), 1334–1346.

[29] Dana Cobzas, Parisa Mosayebi, Albert Murtha, and Martin Jagersand. 2009. Tumor invasion margin on the riemannian space of brain fibers. In *Medical*

Image Computing and Computer-Assisted Intervention–MICCAI 2009. Springer, 531–539.

[30] Chris A Cocosco, Vasken Kollokian, Remi K-S Kwan, G Bruce Pike, and Alan C Evans. 1997. BrainWeb: Online interface to a 3D MRI simulated brain database. In *NeuroImage*. Citeseer.

[31] Kevin D. Costa, Jeffrey W. Holmes, and Andrew D. McCulloch. 2001. Modeling cardiac mechanical properties in three dimensions. *Philosophical Transactions of the Royal Society A* 359 (2001), 1233–1250.

[32] Cotin, S. and Delingette, H. and Ayache N. 1999. Real-time elastic deformations of soft tissues for surgery simulation. *IEEE Transactions On Visualization And Computer Graphics* 5, 1 (1999), 62–73.

[33] Kenneth Czechowski, Casey Battaglino, Chris McClanahan, Kartik Iyer, P-K Yeung, and Richard Vuduc. 2012. On the communication complexity of 3D FFTs and its implications for exascale. In *Proceedings of the 26th ACM international conference on Supercomputing*. ACM, 205–214.

[34] Stéphane Del Pino and Olivier Pironneau. 2003. A fictitious domain based general PDE solver. *Numerical Methods for Scientific Computing Variational Problems and Applications, Barcelona* (2003).

[35] Michel O Deville, Paul F Fischer, and Ernest H Mund. 2002. *High-Order Methods for Incompressible Fluid Flow*. Cambridge Monographs on Applied and Computational Mathematics, Vol. 9. Cambridge University Press.

[36] Therese A Dolecek, Jennifer M Propp, Nancy E Stroup, and Carol Kruchko. 2012. CBTRUS statistical report: primary brain and central nervous system

tumors diagnosed in the United States in 2005–2009. *Neuro-oncology* 14, suppl 5 (2012), v1–v49.

[37] Ron O Dror, JP Grossman, Kenneth M Mackenzie, Brian Towles, Edmond Chow, John K Salmon, Cliff Young, Joseph A Bank, Brannon Batson, Martin M Deneroff, and others. 2010. Exploiting 162-nanosecond end-to-end communication latency on Anton. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 1–12.

[38] Truong Vinh Truong Duy and Taisuke Ozaki. 2014. A decomposition method with minimum communication amount for parallelization of multi-dimensional FFTs. *Computer Physics Communications* 185, 1 (2014), 153 – 164. http://www.sciencedirect.com/science/article/pii/S0010465513003020

[39] S. C. Eisentat and H. F. Walker. 1996. Choosing the forcing terms in an inexact Newton method. *SIAM Journal on Scientific Computing* 17, 1 (1996), 16–32.

[40] Christian Engwer, Thomas Hillen, Markus Knappitsch, and Christina Surulescu. 2014. Glioma follow white matter tracts: a multiscale DTI-based model. *Journal of Mathematical Biology* (2014), 1–32.

[41] Frank Ethridge and Leslie Greengard. 2001. A new fast-multipole accelerated Poisson solver in two dimensions. *SIAM Journal on Scientific Computing* 23, 3 (2001), 741–760.

[42] M. Falcone and R. Ferretti. 1998. Convergence analysis for a class of high-order semi-Lagrangian advection schemes. *SIAM J. Numer. Anal.* 35, 3 (1998), 909–940.

[43] A. Fathi, LF. Kallivokas, and B. Poursartip. 2015. Full waveform inversion in three-dimensional PML-truncated elastic media. *Computer Methods in Applied Mechanics and Engineering (under review)* (2015).

[44] Ferrant, M. and Nabavi, A. and Macq B. and Jolesz F. A. and Kikinis R. and Warfield S. K. 2001. Registration of 3-D intraoperative MR images of the brain using a finite-element biomechanical model. *IEEE Transactions On Medical Imaging* 20, 12 (2001), 1384–1397.

[45] Salvatore Filippone. 1996. The IBM parallel engineering and scientific subroutine library. In *Applied Parallel Computing Computations in Physics, Chemistry and Engineering Science*. Springer, 199–206.

[46] B. Fischer and J. Modersitzki. 2008. Ill-posed medicine – an introduction to image registration. *Inverse Problems* 24, 3 (2008), 1–16.

[47] Ian T Foster and Patrick H Worley. 1997. Parallel algorithms for the spectral transform method. *SIAM Journal on Scientific Computing* 18, 3 (1997), 806–837.

[48] Franz Franchetti, Markus Püschel, Yevgen Voronenko, Srinivas Chellappa, and José MF Moura. 2009. Discrete Fourier transform on multicore. *Signal Processing Magazine, IEEE* 26, 6 (2009), 90–102.

[49] Franz Franchetti, Yevgen Voronenko, and Gheorghe Almasi. 2013. Automatic Generation of the HPC Challenge Global FFT Benchmark for Blue Gene/P. In *High Performance Computing for Computational Science-VECPAR 2012*. Springer, 187–200.

[50] Matteo Frigo and Steven G Johnson. 2005. The design and implementation of FFTW3. *Proc. IEEE* 93, 2 (2005), 216–231.

[51] Matteo Frigo and Steven G. Johnson. 2017. FFTW home page. (2017). http://www.fftw.org

[52] Hormozd Gahvari, Allison H Baker, Martin Schulz, Ulrike Meier Yang, Kirk E Jordan, and William Gropp. 2011. Modeling the performance of an algebraic multigrid cycle on HPC platforms. In *Proceedings of the international conference on Supercomputing*. ACM, 172–181.

[53] Hormozd Gahvari and William Gropp. 2010. An introductory exascale feasibility study for FFTs and multigrid. In *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 1–9.

[54] Michael W. Gee, Christopher M. Siefert, Jonathan J. Hu, Ray S. Tuminaro, and Marzion G. Sala. 2006. *ML 5.0 Smoothed Aggregation User's Guide*. Technical Report SAND2006-2649. Sandia National Laboratories.

[55] D.T. Gering, A. Nabavi, R. Kikinis, N. Hata, L.J. Donnell, and W.E. Grimson. 2001. An integrated visualization system for surgical planning and guidance using image fusion and an open MR. *Journal of Magnetic Resonance Imaging* 13, 6 (2001), 967–975.

[56] A. Gholami and G. Biros. 2017. AccFFT. (2017). https://github.com/amirgholami/accfft

[57] A. Gholami, J. Hill, D. Malhotra, and G. Biros. 2016. AccFFT: A library for distributed-memory FFT on CPU and GPU architectures. *arXiv e-prints* (2016). in review (arXiv preprint: http://arxiv.org/abs/1506.07933);.

[58] A. Gholami, A. Mang, and G. Biros. 2016. An inverse problem formulation

for parameter estimation of a reaction-diffusion model of low grade gliomas. *Journal of Mathematical Biology* 72, 1 (2016), 409–433.

[59] Alf Giese, Lan Kluwe, Britta Laube, Hildegard Meissner, Michael E Berens, and Manfred Westphal. 1996. Migration of human glioma cells on myelin. *Neurosurgery* 38, 4 (1996), 755–764.

[60] Ali Gooya, George Biros, and Christos Davatzikos. 2011. Deformable registration of glioma images using EM algorithm and diffusion reaction modeling. *IEEE transactions on medical imaging* 30, 2 (2011), 375–390.

[61] Ali Gooya, Kilian M. Pohl, Michel Bilello, George Biros, and Christos Davatzikos. 2011. Joint Segmentation and Deformable Registration of Brain Scans Guided by a Tumor Growth Model. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2011*. Lecture Notes in Computer Science, Vol. 6892. Springer Berlin Heidelberg, 532–540.

[62] A. Gooya, K. M. Pohl, M. Bilello, L. Cirillo, G. Biros, E. R. Melhem, and C. Davatzikos. 2013. GLISTR: Glioma image segmentation and registration. *Medical Imaging, IEEE Transactions on* 31, 10 (2013), 1941–1954.

[63] A. Grama, A. Gupta, G. Karypis, and V. Kumar. 2003. *An Introduction to Parallel Computing: Design and Analysis of Algorithms* (second ed.). Addison Wesley.

[64] Liang Gu, Jakob Siegel, and Xiaoming Li. 2011. Using GPUs to compute large out-of-card FFTs. In *Proceedings of the international conference on Supercomputing*. ACM, 255–264.

[65] Morton E Gurtin. 1982. *An introduction to continuum mechanics*. Vol. 158. Academic press.

[66] Wolfgang Hackbusch. 1985. *Multigrid methods and applications*. Springer Series in Computational Mathematics, Vol. 4. Springer-Verlag, Berlin. xiv+377 pages.

[67] Per Christian Hansen. 1992. Analysis of discrete ill-posed problems by means of the L-curve. *SIAM review* 34, 4 (1992), 561–580.

[68] Per Christian Hansen. 1999. *The L-curve and its use in the numerical treatment of inverse problems*. IMM, Department of Mathematical Modelling, Technical Universityof Denmark.

[69] Per Christian Hansen. 2010. *Discrete inverse problems: Insight and algorithms*. SIAM.

[70] G. L. Hart, C. Zach, and M. Niethammer. 2009. An optimal control approach for deformable registration. In *Proc IEEE Conference on Computer Vision and Pattern Recognition*. 9–16.

[71] A. Hawkins-Daarud, R. C. Rockne, A. R. A. Anderson, and K. R. Swanson. 2013. Modeling tumor-associated edema in gliomas during anti-angiogenic therapy and its imapct on imageable tumor. *Front Oncol* 3, 66 (2013).

[72] Andrea Hawkins-Daarud, Kristoffer G van der Zee, and J Tinsley Oden. 2012. Numerical simulation of a thermodynamically consistent four-species tumor growth model. *International journal for numerical methods in biomedical engineering* 28, 1 (2012), 3–24.

[73] Michael A Heroux, Roscoe A Bartlett, Vicki E Howle, Robert J Hoekstra, Jonathan J Hu, Tamara G Kolda, Richard B Lehoucq, Kevin R Long, Roger P

Pawlowski, Eric T Phipps, and others. 2005. An overview of the Trilinos project. *ACM Transactions on Mathematical Software (TOMS)* 31, 3 (2005), 397–423.

[74] Thomas Hillen. 2006. M5 mesoscopic and macroscopic models for mesenchymal motion. *Journal of Mathematical Biology* 53, 4 (2006), 585–616.

[75] J. Hinkle, P. T. Fletcher, B. Wang, B. Salter, and S. Joshi. 2009. 4D MAP image reconstruction incorporating organ motion. In *Proc Information Processing in Medical Imaging*. 676–687.

[76] M. Hinze, R. Pinnau, M. Ulbrich, and S. Ulbrich. 2009. *Optimization with PDE constraints*. Springer, Berlin, DE.

[77] Fred H Hochberg and Amy Pruitt. 1980. Assumptions in the radiotherapy of glioblastoma. *Neurology* 30, 9 (1980), 907–907.

[78] C.S. Hogea, F. Abraham, G. Biros, and C. Davatzikos. 2006. A Framework for Soft Tissue Simulations with Applications to Modeling Brain Tumor Mass-Effect in 3D Images. In *Medical Image Computing and Computer-Assisted Intervention Workshop on Biomechanics*. Copenhagen.

[79] Cosmina Hogea, Christos Davatzikos, and George Biros. 2007. Modeling glioma growth and mass effect in 3D MR images of the brain. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2007*. Springer, 642–650.

[80] Cosmina Hogea, Christos Davatzikos, and George Biros. 2008. Brain-tumor interaction biophysical models for medical image registration. *SIAM Journal on Scientific Computing* 30, 6 (2008), 3050–3072.

[81] C. Hogea, C. Davatzikos, and G. Biros. 2008. An image-driven parameter estimation problem for a reaction-diffusion glioma growth model with mass effect. *J Math Biol* 56 (2008), 793–825.

[82] Betsy A Holland, Michael Brant-Zawadzki, David Norman, and T Hans Newton. 1985. Magnetic resonance imaging of primary intracranial tumors: a review. *International Journal of Radiation Oncology* Biology* Physics* 11, 2 (1985), 315–321.

[83] Mark A Horsfield and Derek K Jones. 2002. Applications of diffusion-weighted and diffusion tensor MRI to white matter diseases–a review. *NMR in Biomedicine* 15, 7-8 (2002), 570–577.

[84] Sanja Ivkovic, Christopher Beadle, Sonal Noticewala, Susan C Massey, Kristin R Swanson, Laura N Toro, Anne R Bresnick, Peter Canoll, and Steven S Rosenfeld. 2012. Direct inhibition of myosin II effectively blocks glioma invasion in the presence of multiple motogens. *Molecular biology of the cell* 23, 4 (2012), 533–542.

[85] Saâd Jbabdi, Emmanuel Mandonnet, Hugues Duffau, Laurent Capelle, Kristin Rae Swanson, Mélanie Pélégrini-Issac, Rémy Guillevin, and Habib Benali. 2005. Simulation of anisotropic growth of low-grade gliomas using diffusion tensor imaging. *Magnetic Resonance in Medicine* 54, 3 (2005), 616–624.

[86] LF Kallivokas, Arash Fathi, S Kucukcoban, KH Stokoe II, J Bielak, and O Ghattas. 2013. Site characterization using full waveform inversion. *Soil Dynamics and Earthquake Engineering* 47 (2013), 62–82.

[87] Krishna Kandalla, Hari Subramoni, Karen Tomko, Dmitry Pekurovsky, Sayantan Sur, and Dhabaleswar K Panda. 2011. High-performance and scalable non-blocking all-to-all with collective offload on InfiniBand clusters: a study with parallel 3-D FFT. *Computer Science-Research and Development* 26, 3-4 (2011), 237–246.

[88] A. Kirsch. 2011. *An introduction to the mathematical theory of inverse problems*. Springer.

[89] Ender Konukoglu, Olivier Clatz, Pierre-Yves Bondiau, Hervé Delingette, and Nicholas Ayache. 2010. Extrapolating glioma invasion margin in brain magnetic resonance images: suggesting new irradiation margins. *Medical Image Analysis* 14, 2 (2010), 111–125.

[90] Ender Konukoglu, Olivier Clatz, Bjoern H Menze, Bram Stieltjes, M-A Weber, Emmanuel Mandonnet, Hervé Delingette, and Nicholas Ayache. 2010. Image guided personalization of reaction-diffusion type tumor growth models using modified anisotropic eikonal equations. *Medical Imaging, IEEE Transactions on* 29, 1 (2010), 77–95.

[91] Dima Kozakov, Ryan Brenke, Stephen R Comeau, and Sandor Vajda. 2006. PIPER: An FFT-based protein docking program with pairwise potentials. *Proteins: Structure, Function, and Bioinformatics* 65, 2 (2006), 392–406.

[92] Harper Langston, Leslie Greengard, and Denis Zorin. 2011. A free-space adaptive FMM-based PDE solver in three dimensions. *Communications in Applied Mathematics and Computational Science* 6, 1 (2011), 79–122.

[93] Ilya Lashuk, Aparna Chandramowlishwaran, Harper Langston, Tuan-Anh

Nguyen, Rahul Sampath, Aashay Shringarpure, Richard Vuduc, Lexing Ying, Denis Zorin, and George Biros. 2012. A massively parallel adaptive fast multipole method on heterogeneous architectures. *Commun. ACM* 55, 5 (May 2012), 101–109.

[94] Yaacov Richard Lawrence, X Allen Li, Issam El Naqa, Carol A Hahn, Lawrence B Marks, Thomas E Merchant, and Adam P Dicker. 2010. Radiation dose–volume effects in the brain. *International Journal of Radiation Oncology* Biology* Physics* 76, 3 (2010), S20–S27.

[95] Denis Le Bihan, Jean-François Mangin, Cyril Poupon, Chris A Clark, Sabina Pappata, Nicolas Molko, and Hughes Chabriat. 2001. Diffusion tensor imaging: concepts and applications. *Journal of Magnetic Resonance Imaging* 13, 4 (2001), 534–546.

[96] E. Lee and M. Gunzburger. 2010. An optimal control formulation of an image registration problem. *Journal of Mathematical Imaging and Vision* 36, 1 (2010), 69–80.

[97] Myoungkyu Lee, Nicholas Malaya, and Robert D Moser. 2013. Petascale direct numerical simulation of turbulent channel flow on up to 786k cores. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 61.

[98] J Li, W Liao, A Choudhary, R Ross, R Thakur, W Gropp, R Latham, A Siegel, B Gallagher, and M Zingale. 2003. Parallel netCDF: A scientific highperformance I. In *O Interface the Supercomputing Conference*.

[99] Ning Li and Sylvain Laizet. 2010. 2DECOMP & FFT-A Highly Scalable 2D

Decomposition Library and FFT Interface. In *Cray User Group 2010 conference*. 1–13.

[100] Xiaoye S. Li and James W. Demmel. 2003. SuperLU_DIST: A Scalable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems. *ACM Transactions of Mathematical Software* 29, 2 (June 2003), 110–140.

[101] EABF Lima, JT Oden, DA Hormuth, TE Yankeelov, and RC Almeida. 2016. Selection, calibration, and validation of models of tumor growth. *Mathematical Models and Methods in Applied Sciences* 26, 12 (2016), 2341–2368.

[102] J. L. Lions. 1971. *Optimal control of systems governed by partial differential equations*. Springer.

[103] James W Lottes and Paul F Fischer. 2005. Hybrid multigrid/Schwarz algorithms for the spectral element method. *Journal of Scientific Computing* 24, 1 (2005), 45–78.

[104] Xingshuang Ma, Hao Gao, Boyce E Griffith, Colin Berry, and Xiaoyu Luo. 2013. Image-based fluid–structure interaction model of the human mitral valve. *Computers & Fluids* 71 (2013), 417–425.

[105] Luke Macyszyn, Hamed Akbari, Jared M Pisapia, Xiao Da, Mark Attiah, Vadim Pigrish, Yingtao Bi, Sharmistha Pal, Ramana V Davuluri, Laura Roccograndi, and others. 2016. Imaging patterns predict patient survival and molecular subtype in glioblastoma via machine learning techniques. *Neuro-oncology* 18, 3 (2016), 417–425.

[106] Dhairya Malhotra and George Biros. 2016. Algorithm 967: A Distributed-

Memory Fast Multipole Method for Volume Potentials. *ACM Trans. Math. Softw.* 43, 2, Article 17 (Aug. 2016), 27 pages.

[107] Dhairya Malhotra, Amir Gholami, and George Biros. 2014. A volume integral equation stokes solver for problems with variable coefficients. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 92–102.

[108] A. Mang and G. Biros. 2015. An inexact Newton–Krylov algorithm for constrained diffeomorphic image registration. *SIAM Journal on Imaging Sciences* 8, 2 (2015), 1030–1069.

[109] A. Mang and G. Biros. 2016. Constrained H^1-regularization schemes for diffeomorphic image registration. *SIAM Journal on Imaging Sciences* 9, 3 (2016), 1154–1194.

[110] A. Mang and G. Biros. 2016. A Semi-Lagrangian two-level preconditioned Newton–Krylov solver for constrained diffeomorphic image registration. *arXiv e-prints* (2016). in review (arXiv preprint: http://arxiv.org/abs/1604.02153).

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

[112] A. Mang, T. A. Schuetz, S. Becker, A. Toma, and T. M. Buzug. 2012. Cyclic numerical time integration in variational non-rigid image registration based on quadratic regularisation. In *Proc Vision, Modeling and Visualization Workshop*. 143–150.

[113] Andreas Mang, Alina Toma, Tina A Schuetz, Stefan Becker, Thomas Eckey, Christian Mohr, Dirk Petersen, and Thorsten M Buzug. 2012. Biophysical modeling of brain tumor progression: from unconditionally stable explicit time integration to an inverse problem with parabolic PDE constraints for model calibration. *Medical Physics* 39, 7 (2012), 4444–4459.

[114] Peter McCorquodale, Phillip Colella, Gregory T Balls, and Scott B Baden. 2006. A Local Corrections Algorithm for Solving Poisson's Equation in Three Dimensions. *Communications in Applied Mathematics and Computational Science* 2, LBNL–62377 (2006).

[115] D. M. McQueen and C. S. Peskin. 1997. Shared-memory parallel vector implementation of the immersed boundary method for the computation of blood flow in the beating mammalian heart. *Journal Of Supercomputing* 11, 3 (1997), 213–236.

[116] B. H. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, and others. 2015. The multimodal brain tumor image segmentation benchmark (BRATS). *IEEE Transactions on Medical Imaging* 34, 10 (2015), 1993–2024.

[117] Bjoern H Menze, Andras Jakab, Stefan Bauer, Jayashree Kalpathy-Cramer, Keyvan Farahani, Justin Kirby, Yuliya Burren, Nicole Porz, Johannes Slotboom, Roland Wiest, and others. 2015. The multimodal brain tumor image segmentation benchmark (BRATS). *IEEE transactions on medical imaging* 34, 10 (2015), 1993–2024.

[118] B. H. Menze, K. V. Leemput, E. Konukoglu, A. Honkela, M.-A. Weber, N. Ayache, and P. Golland. 2011. A generative approach for image-based model-

ing of tumor growth. In *Proc Information Processing in Medical Imaging*, Vol. 22. 735–747.

[119] J. C. Michel, H. Moulinec, and P. Suquet. 1999. Effective properties of composite materials with periodic microstructure: a computational approach. *Comput. Methods Appl. Mech. Engrg* 172, 1–4 (1999).

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

[121] Ashraf Mohamed and Christos Davatzikos. 2005. Finite element modeling of brain tumor mass-effect from 3D medical images. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2005*. Springer, 400–408.

[122] Susumu Mori, Kenichi Oishi, Hangyi Jiang, Li Jiang, Xin Li, Kazi Akhter, Kegang Hua, Andreia V Faria, Asif Mahmood, Roger Woods, and others. 2008. Stereotaxic white matter atlas based on diffusion tensor imaging in an ICBM template. *Neuroimage* 40, 2 (2008), 570–582.

[123] Parisa Mosayebi, Dana Cobzas, Albert Murtha, and Martin Jagersand. 2012. Tumor invasion margin on the Riemannian space of brain fibers. *Medical Image Analysis* 16, 2 (2012), 361–373.

[124] J. D. Murray. 1989. *Mathematical biology*. Springer-Verlag, New York.

[125] Nimalan Nandapalan, Jiri Jaros, Alistair P Rendell, and Bradley Treeby. 2012. Implementation of 3-D FFTs Across Multiple GPUs in Shared Memory Environments. In *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2012 13th International Conference on*. IEEE, 167–172.

[126] HB Newton. 1994. Primary brain tumors: review of etiology, diagnosis and treatment. *American Family Physician* 49, 4 (1994), 787–797.

[127] J. Nocedal and S. J. Wright. 2006. *Numerical Optimization*. Springer, New York, New York, US.

[128] Akira Nukada, Kento Sato, and Satoshi Matsuoka. 2012. Scalable multi-GPU 3-D FFT for TSUBAME 2.0 Supercomputer. In *2012 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE, 1–10.

[129] CUDA Nvidia. 2010. CUFFT library. (2010).

[130] J Tinsley Oden. 2011. *An Introduction to Mathematical Modeling: A Course in Mechanics*. Vol. 1. John Wiley & Sons.

[131] J Tinsley Oden, Ernesto ABF Lima, Regina C Almeida, Yusheng Feng, Marissa Nichole Rylander, David Fuentes, Danial Faghihi, Mohammad M Rahman, Matthew DeWitt, Manasa Gadde, and others. 2016. Toward predictive multiscale modeling of vascular tumor growth. *Archives of Computational Methods in Engineering* 23, 4 (2016), 735–779.

[132] Daniel Orozco, Elkin Garcia, Robert Pavel, Orlando Ayala, Lian-Ping Wang, and Guang Gao. 2012. Demystifying Performance Predictions of Distributed FFT 3-D Implementations. In *Network and Parallel Computing*. Springer, 196–207.

[133] KJ Painter and T Hillen. 2013. Mathematical modelling of glioma growth: the use of Diffusion Tensor Imaging (DTI) data to predict the anisotropic pathways of cancer invasion. *Journal of Theoretical Biology* 323 (2013), 25–39.

[134] Jongsoo Park, Ganesh Bikshandi, Karthikeyan Vaidyanathan, Ping Tak Peter Tang, Pradeep Dubey, and Daehyun Kim. 2013. Tera-scale 1D FFT with low-communication algorithm and Intel® Xeon Phi coprocessors. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 34.

[135] Dmitry Pekurovsky. 2012. P3DFFT: A framework for parallel computations of Fourier transforms in three dimensions. *SIAM Journal on Scientific Computing* 34, 4 (2012), C192–C209.

[136] Kara Pham, Arnaud Chauviere, Haralambos Hatzikirou, Xiangrong Li, Helen M Byrne, Vittorio Cristini, and John Lowengrub. 2012. Density-dependent quiescence in glioma invasion: instability in a simple reaction–diffusion model for the migration/proliferation dichotomy. *Journal of biological dynamics* 6, sup1 (2012), 54–71.

[137] J.R. Phillips and J.K. White. 1997. A precorrected-FFT method for electrostatic analysis of complicated 3-D structures. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 16, 10 (1997), 1059–1072.

[138] James C. Phillips, Gengbin Zheng, Sameer Kumar, and Laxmikant V. Kalé. 2002. NAMD: Biomolecular Simulation on Thousands of Processors. In *Proceedings of Supercomputing (The SCxy Conference series)*. ACM/IEEE, Baltimore, Maryland.

[139] Joel R. Phillips and Jacob K. White. 1997. A Precorrected-FFT Method for Electorstatic Analysis of Complicated 3-D Structures. *IEEE Transactions on*

Computer-Aided Design of Integrated Circuits and Systems 16, 10 (1997), 1059–1072.

[140] Michael Pippig. 2013. PFFT: An extension of FFTW to massively parallel architectures. *SIAM Journal on Scientific Computing* 35, 3 (2013), C213–C236.

[141] Michael Pippig and Daniel Potts. 2013. Parallel three-dimensional nonequispaced Fast Fourier Transforms and their application to particle simulation. *SIAM Journal on Scientific Computing* 35, 4 (2013), C411–C437.

[142] G Powathil, M Kohandel, S Sivaloganathan, A Oza, and M Milosevic. 2007. Mathematical modeling of brain tumors: effects of radiotherapy and chemotherapy. *Physics in Medicine and Biology* 52, 11 (2007), 3291.

[143] Markus Püschel, José MF Moura, Jeremy R Johnson, David Padua, Manuela M Veloso, Bryan W Singer, Jianxin Xiong, Franz Franchetti, Aca Gačic, Yevgen Voronenko, and others. 2005. SPIRAL: Code generation for DSP transforms. *Proc. IEEE* 93, 2 (2005), 232–275.

[144] Islem Rekik, Stéphanie Allassonnière, Olivier Clatz, Ezequiel Geremia, Erin Stretton, Hervé Delingette, and Nicholas Ayache. 2013. Tumor growth parameters estimation and source localization from a unique time point: Application to low-grade gliomas. *Computer Vision and Image Understanding* 117, 3 (2013), 238–249.

[145] R Rockne, EC Alvord Jr, JK Rockhill, and KR Swanson. 2009. A mathematical model for brain tumor response to radiation therapy. *Journal of Mathematical Biology* 58, 4-5 (2009), 561–578.

[146] R Rockne, JK Rockhill, M Mrugala, AM Spence, I Kalet, K Hendrickson, A Lai, T Cloughesy, EC Alvord Jr, and KR Swanson. 2010. Predicting the efficacy of radiotherapy in individual glioblastoma patients in vivo: a mathematical modeling approach. *Physics in Medicine and Biology* 55, 12 (2010), 3271.

[147] David L Ropp and John N Shadid. 2009. Stability of operator splitting methods for systems with indefinite operators: advection–diffusion–reaction systems. *J. Comput. Phys.* 228, 9 (2009), 3508–3516.

[148] Michael Salcman. 1980. Survival in glioblastoma: historical perspective. *Neurosurgery* 7, 5 (1980), 435–439.

[149] Rahul S. Sampath and George Biros. 2010. A Parallel Geometric Multigrid Method for Finite Elements on Octree Meshes. *SIAM Journal on Scientific Computing* 32, 3 (2010), 1361–1392.

[150] Olivier Saut, Jean-Baptiste Lagaert, Thierry Colin, and Hassan M Fathallah-Shaykh. 2014. A multilayer grow-or-go model for GBM: effects of invasive cells and anti-angiogenesis on growth. *Bulletin of mathematical biology* 76, 9 (2014), 2306–2333.

[151] IF Sbalzarini, JH Walther, M. Bergdorf, SE Hieber, EM Kotsalis, and P. Koumoutsakos. 2006. PPM–A highly efficient parallel particle–mesh library for the simulation of continuum systems. *J. Comput. Phys.* 215, 2 (2006), 566–588.

[152] M. Schatz, J. Poulson, and R. A. van de Geijn. 2014. Parallel Matrix Multiplication: A Systematic Journey. *(submitted to) SIAM Journal on Scientific Computing* (2014).

[153] RB Seither, B Jose, KJ Paris, RD Lindberg, and WJ Spanos. 1995. Results of irradiation in patients with high-grade gliomas evaluated by magnetic resonance imaging. *American Journal of Clinical Oncology* 18, 4 (1995), 297–299.

[154] Maxime Sermesant, Hervé Delingette, and Nicholas Ayache. 2006. An Electromechanical Model of the Heart for Image Analysis and Simulation. *IEEE Transactions in Medical Imaging* 25, 5 (2006), 612–625.

[155] Maxime Sermesant, Philippe Moireau, Oscar Camara, Jacques Sainte-Marie, Rado Andriantsimiavona, Robert Cimrman, Derek L. Hill, Dominique Chapelle, and Reza Razavi. 2006. Cardiac function estimation from MRI using a heart model and data assimilation: Advances and difficulties. *Medical Image Analysis* 10, 4 (2006), 642–656.

[156] Daniel L Silbergeld and Michael R Chicoine. 1997. Isolation and characterization of human malignant glioma cells from histologically normal brain. *Journal of Neurosurgery* 86, 3 (1997), 525–531.

[157] R Sodt, R Rockne, ML Neal, I Kalet, and K R Swanson. 2014. Quantifying the role of anisotropic invasion in human glioblastoma. In *Computational Surgery and Dual Training*. Springer, 315–329.

[158] Sukhyun Song and Jeffrey K. Hollingsworth. 2014. Scaling Parallel 3-D FFT with Non-blocking MPI Collectives. In *Proceedings of the 5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA '14)*. IEEE Press, Piscataway, NJ, USA, 1–8.

[159] A. Sotiras, C. Davatzikos, and N. Paragios. 2013. Deformable medical image

registration: A survey. *Medical Imaging, IEEE Transactions on* 32, 7 (2013), 1153–1190.

[160] Andreas Stadlbauer, Esther Pölking, Olaf Prante, Christopher Nimsky, Michael Buchfelder, Torsten Kuwert, Rainer Linke, Marc Doelken, and Oliver Ganslandt. 2009. Detection of tumour invasion into the pyramidal tract in glioma patients with sensorimotor deficits by correlation of 18F-fluoroethyl-L-tyrosine PET and magnetic resonance diffusion tensor imaging. *Acta Neurochirurgica* 151, 9 (2009), 1061–1069.

[161] Andrew M Stein, Tim Demuth, David Mobley, Michael Berens, and Leonard M Sander. 2007. A mathematical model of glioblastoma tumor spheroid invasion in a three-dimensional in vitro experiment. *Biophysical Journal* 92, 1 (2007), 356–365.

[162] Gilbert Strang. 1968. On the construction and comparison of difference schemes. *SIAM J. Numer. Anal.* 5, 3 (1968), 506–517.

[163] Hari Sundar, George Biros, Carsten Burstedde, Johann Rudi, Omar Ghattas, and Georg Stadler. 2012. Parallel geometric-algebraic multigrid on unstructured forests of octrees. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 43.

[164] Hari Sundar, Rahul S. Sampath, and George Biros. 2008. Bottom-Up Construction and 2:1 Balance Refinement of Linear Octrees in Parallel. *SIAM Journal on Scientific Computing* 30, 5 (2008), 2675–2708.

[165] Hari Sundar, Georg Stadler, and George Biros. 2015. Comparison of multi-

grid algorithms for high-order continuous finite element discretizations. *Numerical Linear Algebra with Applications* 22, 4 (2015), 664–680.

[166] KR Swanson, EC Alvord, and JD Murray. 2000. A quantitative model for differential motility of gliomas in grey and white matter. *Cell Proliferation* 33, 5 (2000), 317–330.

[167] KR Swanson, RC Rostomily, and EC Alvord. 2008. A mathematical modelling tool for predicting survival of individual patients following resection of glioblastoma: a proof of principle. *British Journal of Cancer* 98, 1 (2008), 113–119.

[168] Kristin R Swanson, EC Alvord, and JD Murray. 2002. Virtual brain tumours (gliomas) enhance the reality of medical imaging and highlight inadequacies of current therapy. *British Journal of Cancer* 86, 1 (2002), 14–18.

[169] K. R. Swanson, E. C. Alvord, and J. D. Murray. 2000. A quantitative model for differential motility of gliomas in grey and white matter. *Cell Proliferation* 33, 5 (2000), 317–330.

[170] Paul N Swarztrauber. 1987. Multiprocessor FFTs. *Parallel computing* 5, 1 (1987), 197–210.

[171] Roland A Sweet, William L Briggs, Suely Oliveira, Jules L Porsche, and Tom Turnbull. 1991. FFTs and three-dimensional Poisson solvers for hypercubes. *Parallel computing* 17, 2 (1991), 121–131.

[172] Daisuke Takahashi. 2010. An implementation of parallel 3-D FFT with 2-D decomposition on a massively parallel cluster of multi-core processors. In *Parallel Processing and Applied Mathematics*. Springer, 606–614.

[173] D Takahashi. 2013. Implementation of Parallel 1-D FFT on GPU Clusters. In *Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on*. 174–180.

[174] Ping Tak Peter Tang, Jongsoo Park, Daehyun Kim, and Vladimir Petrov. 2013. A framework for low-communication 1-D FFT. *Scientific Programming* 21, 3 (2013), 181–195.

[175] Osamu Tatebe and Yoshio Oyanagi. 1994. Efficient implementation of the multigrid preconditioned conjugate gradient method on distributed memory machines. In *Supercomputing'94. Proceedings*. IEEE, 194–203.

[176] Andrey Tikhonov. 1963. Solution of incorrectly formulated problems and the regularization method. In *Soviet Math. Dokl.*, Vol. 5. 1035–1038.

[177] Raju Tomer, Li Ye, Brian Hsueh, and Karl Deisseroth. 2014. Advanced CLARITY for rapid and high-resolution imaging of intact tissues. *Nature protocols* 9, 7 (2014), 1682–1697.

[178] P Tracqui, GC Cruywagen, DE Woodward, GT Bartoo, JD Murray, and EC Alvord. 1995. A mathematical model of glioma growth: the effect of chemotherapy on spatio-temporal growth. *Cell Proliferation* 28, 1 (1995), 17–31.

[179] Ulrich Trottenberg, Cornelius W Oosterlee, and Anton Schuller. 2000. *Multigrid*. Academic press.

[180] Raymond S Tuminaro, Erik G Boman, Jonathan J Hu, Andrey Prokopenko, Christopher Siefert, Paul H Tsuji, Jeremie Gaidamour, Luke Olson, Jacob

Schroder, Badri Hiriyur, and others. 2013. *Toward Flexible Scalable Algebraic Multigrid Solvers.* Technical Report. Sandia National Laboratories Albuquerque, NM; Sandia National Laboratories (SNL-CA), Livermore, CA (United States).

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

[182] Endong Wang, Qing Zhang, Bo Shen, Guangyong Zhang, Xiaowei Lu, Qing Wu, and Yajuan Wang. 2014. Intel Math Kernel Library. In *High-Performance Computing on the Intel® Xeon Phi*. Springer, 167–188.

[183] Warfield, S. K. and Talos, F. and Kemper C. and O'Donnell L. and Westin C. F. and Wells W. M. and Black P. Mc L. and Jolesz F. A. and Kikinis R. 2003. Capturing brain deformation *(Surgery Simulation and Soft Tissue Modeling. International Symposium, IS4TM 2003. Proceedings (Lecture Notes in Comput. Sci. Vol.2673))*. Springer-Verlag, Juan-Les-Pins, France, 203.

[184] Ken C.L. Wong, Linwei Wang, Heye Zhang, Huafeng Liu, and Pengcheng Shi. 2007. Integrating Functional and Structural Images for Simultaneous Cardiac Segmentation and Deformation Recovery. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2007 (LNCS)*, Nicholas Ayache, Sébastien Ourselin, and Anthony Maeder (Eds.), Vol. 4791. Springer, 270–277.

[185] Margaret Wrensch, Yuriko Minn, Terri Chew, Melissa Bondy, and Mitchel S Berger. 2002. Epidemiology of primary brain tumors: current concepts and review of the literature. *Neuro-oncology* 4, 4 (2002), 278–299.

[186] Jing Wu and Joseph JaJa. 2012. Optimized strategies for mapping three-dimensional FFTs onto CUDA GPUs. In *Innovative Parallel Computing (InPar), 2012*. IEEE, 1–12.

[187] Bo Xiao. 2014. *Parallel algorithms for generalized N-body problem in high dimensions and their applications for bayesian inference and image analysis*. Ph.D. Dissertation. Georgia Institute of Technology.

[188] Thomas E Yankeelov, Nkiruka Atuegwu, David Hormuth, Jared A Weis, Stephanie L Barnes, Michael I Miga, Erin C Rericha, and Vito Quaranta. 2013. Clinically relevant modeling of tumor growth and treatment response. *Science translational medicine* 5, 187 (2013), 187ps9–187ps9.

[189] Lexing Ying, George Biros, and Denis Zorin. 2004. A kernel-independent adaptive fast multipole method in two and three dimensions. *J. Comput. Phys.* 196, 2 (2004), 591–626.

[190] Rio Yokota and Lorena A Barba. 2012. A tuned and scalable fast multipole method as a preeminent algorithm for exascale systems. *International Journal of High Performance Computing Applications* 26, 4 (2012), 337–346.