Local Fourier Analysis of Domain Decomposition and Multigrid Methods for High-Order Matrix-Free Finite Elements

by

Jeremy L. Thompson

B.S., United States Air Force Academy, 2009

M.Sc., University of Washington, 2011

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Applied Mathematics

2021

Committee Members:

Jed Brown, Chair

Daniel Appelö

Adrianna Gillman

Ian Grooms

Kenneth Jansen

Thompson, Jeremy L. (Ph.D., Applied Mathematics)

Local Fourier Analysis of Domain Decomposition and Multigrid Methods for High-Order Matrix-Free Finite Elements

Thesis directed by Prof. Jed Brown

High-order matrix-free finite element operators offer superior performance on modern high performance computing hardware when compared to assembled sparse matrices, both with respect to floating point operations needed for operator evaluation and the memory transfer needed for a matrix-vector product. However, high-order matrix-free operators require iterative solvers, such as Krylov subspace methods, and these methods converge slowly for ill-conditioned operators. Preconditioning techniques are needed to improve the convergence of these iterative solvers independent of problem size and resolution.

P-multigrid and domain decomposition methods are particularly well suited for problems on unstructured meshes, but these methods can involve parameters that require careful tuning to ensure proper convergence. Local Fourier Analysis (LFA) of these preconditioners describes how these methods affect frequency modes in the error for the operator defined on an infinite or periodic domain and can provide sharp convergence estimates and parameter tuning while only requiring computation on a single representative element or macro-element patch.

We develop LFA of high-order finite element operators, focusing on multigrid and domain decomposition preconditioning techniques. The LFA of p-multigrid is validated with numerical experiments, and we extend this LFA to reproduce previous work with h-multigrid by using macro-elements consisting of multiple low-order finite elements.

We also develop LFA of the lumped and Dirichlet versions of Balancing Domain Decomposition by Constraints (BDDC) preconditioners for high-order finite elements. By using Fast Diagonalization Method approximate subdomain solvers, the increased setup costs for the Dirichlet BDDC preconditioner, relative to the lumped variant, can be substantially reduced, making

Dirichlet BDDC an attractive preconditioner. We validate this work against previous numerical experiments and exactly reproduce previous work on the LFA of BDDC for subdomains with multiple low-order finite elements.

Aggressive coarsening in p-multigrid is not supported by traditional polynomial smoothers, such as Chebyshev. Dirichlet BDDC can be used as a smoother for p-multigrid to target a wider range of frequency modes, which facilitates more aggressive coarsening. We provide LFA of p-multigrid with Dirichlet BDDC smoothing to demonstrate the suitability of this approach for preconditioning high-order matrix-free finite element operators.

Acknowledgements

This work is supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, in support of the nation's exascale computing imperative.

Contents

Chapter

Tables

Table

Figures

Figure

Chapter 1

Introduction

High-order matrix-free finite element operators offer superior performance on modern high performance computing hardware when compared to assembled sparse matrices, both with respect to the number of floating point operations needed for operator evaluation and the memory transfer needed for a matrix-vector product. However, high-order matrix-free operators require iterative solvers, such as Krylov subspace methods, but these iterative solvers converge slowly for the ill-conditioned operators that come from high-order discretizations. Preconditioning techniques can significantly improve the convergence of these iterative solvers for high-order matrix-free finite element operators. Specifically, p-multigrid and domain decomposition methods are particularly well suited for problems on unstructured meshes, but these methods can have parameters that require careful tuning to ensure proper convergence. Local Fourier Analysis (LFA) of these preconditioners analyzes the frequency modes found in the iterate error following the application of these methods and can provide sharp convergence estimates and parameter tuning while only requiring computation on a single representative element or macro-element patch.

In the remainder of Chapter 1 we provide some brief notes on the reproducibility of the work in this dissertation and provide context for this work in relation to previous work on preconditioning high-order finite element operators.

In Chapter 2, we present a representation of arbitrary second-order partial differential equations (PDEs) for high-order finite element discretizations that facilitates matrix-free implementation. This representation is used by the Center for Efficient Exascale Discretizations to provide

performance portable high-order matrix-free implementations of finite element operators for arbitrary second order partial differential equations. LibCEED has multiple backends that can be selected at runtime, and these backends target CPU architectures, NVIDIA GPUs, and AMD GPUs.

In Chapter 3, we develop LFA of arbitrary order finite element operators represented in this fashion and provide examples of analyzing the with weighted Jacobi and the Chebyshev semi-iterative methods, which are commonly used as smoothers in multigrid methods.

In Chapter 4, this LFA framework is expanded to create the LFA of p-multigrid for Continuous Galerkin discretizations. This LFA of p-multigrid is validated with numerical experiments. Furthermore, we extend this LFA to reproduce previous work with h-multigrid by using macro-elements consisting of multiple low-order finite elements.

In Chapter 5, we develop the LFA of the lumped and Dirichlet versions of Balancing Domain Decomposition by Constraints (BDDC) preconditioners. By using Fast Diagonalization Method approximate subdomain solvers, the increased setup costs for the Dirichlet BDDC preconditioner relative to the lumped variant can be substantially reduced, which makes Dirichlet BDDC an attractive preconditioner. We validate this work against previous numerical experiments on high-order finite elements, and we can exactly reproduce previous work [13] on the LFA of BDDC by using macro-elements consisting of multiple low-order finite elements.

The performance of p-multigrid in parallel is partially determined by the number of multigrid levels. A larger number of multigrid levels requires additional neighbor communication on parallel machines, which is a bottleneck on modern HPC hardware. Furthermore, these intermediate representations are less computationally efficient than the fine grid representation. However, aggressive coarsening is not supported by traditional polynomial smoothers; aggressive coarsening leaves the smoother responsible for wide range of error frequency modes, which causes degraded smoothing unless a very high order smoother is used. Dirichlet BDDC can be used as a smoother for p-multigrid to more efficiently target error mode frequencies than polynomial smoothers such as the Chebyshev semi-iterative method, which facilitates more aggressive coarsening. We provide LFA

of p-multigrid with Dirichlet BDDC smoothing to demonstrate the suitability of this combination for preconditioning high-order matrix-free finite element discretizations.

Finally, in Chapter 6 we provide concluding remarks and areas for future research.

1.1 Reproducibility

Transparency and reproducibility are the lifeblood of scientific and software advancement. The software used in this dissertation is all open source and freely available.

The LFA can be reproduced with the Julia package LFAToolkit.jl [79]. LFAToolkit.jl is a toolkit for analyzing the performance of preconditioners for arbitrary, user provided weak forms of second order PDEs.

The implementation of the high-order matrix-free finite element operators can be found in the libCEED GitHub repository, along with the benchmarking, fluid dynamics, and solid mechanics mini-applications. libCEED [12, 1, 83] is a low-level library for the efficient high-order discretization methods developed by the ECP co-design Center for Efficient Exascale Discretizations (CEED). LibCEED has multiple backends that can be selected at runtime, and these backends target CPU architectures, NVIDIA GPUs, and AMD GPUs. While the focus is on high-order finite elements, the approach used in libCEED is mostly algebraic and thus applicable to other discretizations in factored form.

Finally, the linear and nonlinear solver and preconditioning infrastructure can be found in PETSc [6], the Portable, Extensible Toolkit for Scientific Computation). PETSc is a suite of data structures and routines for the scalable, parallel solution of scientific applications modeled by partial differential equations. It supports MPI, and GPUs through CUDA, HIP, or OpenCL, as well as hybrid MPI-GPU parallelism.

1.2 Connections to Previous Work

High-order finite element discretizations offer accuracy advantages over low-order finite elements [18, 66, 70] and have spectral convergence properties for sufficiently smooth problems.

However, high-order finite elements are less common because a linear operator or the Jacobian of a non-linear operator rapidly loses sparsity in a sparse matrix representation. Matrix-free formulations of these methods [11, 19, 48] provide efficient implementations of these methods on modern hardware [1, 28, 50] with better performance than sparse matrix representations, both in terms of storage and floating point operations required to execute a matrix-vector product.

However, high-order discretizations of PDE operators tend to be ill-conditioned, and iterative solvers, such as Krylov subspace methods like the Conjugate Gradient method [39, 77], are sensitive to the spectrum and condition number of the operator. Preconditioning these high-order PDE operators is necessary to ensure fast convergence of iterative solvers.

1.2.1 Multigrid Methods

Multigrid methods [8, 10, 78] are popular for solving linear systems derived from discretizing PDEs. These methods can be used as solvers or as preconditioners for other iterative solvers.

Geometric multigrid has become a common component in many high performance computing applications and is available in many packages, such as PETSc. May et al. [62] highlighted several benefits to geometric multigrid in extreme scale computing. Specifically, geometric multigrid has a scalable setup phase, the ability to use high performance matrix-free smoothers on all levels, and the ability to use identical stencils (or functions that apply the action of the stencil matrix-free) for all coarse level operators. Geometric multigrid is therefore seen as "undoubtedly *the preconditioner of choice*" for elliptic problems due to this scalability and resolution independent convergence.

H-multigrid, which is the typical type of geometric multigrid used for low-order discretizations, coarsens the mesh by aggregating multiple elements. P-multigrid, developed by Rønquist and Patera [74], is instead based on decreasing the order of the bases in high-order or spectral finite element discretizations.

Davydov, et al. [17] used h-multigrid preconditioning with matrix-free high-order finite element methods for hyperelasticity at finite stain, but p-multigrid can be a more natural fit for matrix-free high-order methods, especially on unstructured meshes where aggregating fine grid

elements to form the coarse grid representation can be somewhat complex.

Algebraic Multigrid (AMG) is based on applying multigrid ideas directly to an assembled matrix representation of the discrete finite element operator. Heys et al. [40] investigated AMG for high-order discretizations with some level of success; however this approach relies upon assembling the high-order operator, which discards the performance benefits of using a matrix-free representation. We restrict our use of AMG to the coarse grid solver as part of p-multigrid or domain decomposition. The coarse grid with linear elements has substantially fewer degrees of freedom, by a factor of approximately $(p+1)^3/2^3$ for p-multigrid, which makes assembly and efficient direct solution of the coarse grid far more tractable.

The coarse grid solve on linear elements with AMG can be relatively fast, and the application of the operator on the fine grid is very efficient with a matrix-free representation, but the intermediate grids between the coarsest and finest levels have less efficient matrix-free representations. Furthermore, an increased number of levels requires additional global communication, which can become a bottleneck in parallel applications. We are therefore interested in geometric multigrid with aggressive p-coarsening from the high-order fine grid directly to the linear coarse grid representation.

Polynomial smoothers are common in multigrid methods. They can be applied matrix-free and offer competitive smoothing properties when compared to Gauss-Seidel [2]. Brannick et al. [9] investigated polynomial smoothers for aggressive coarsening in h-multigrid and found that a rather high degree is needed for Chebyshev smoothers to support aggressive h-coarsening. Our analysis indicates that Chebyshev is not an appropriate smoother for aggressive p-coarsening. We need a better smoother to support rapid p-coarsening directly to linear elements, and we turn to domain decomposition methods to find this improved smoother.

1.2.2 Domain Decomposition Methods

Domain decomposition methods are popular techniques for preconditioning high-order finite element and spectral element discretizations. Toselli and Widlund surveyed many of the domain

decomposition techniques [80] and we outline the development of these methods below.

Schwartz methods for domain decomposition were first developed by Hermann Schwarz [76] as an alternating method in which solutions in overlapping subregions of the global problem were solved while using previous solutions in neighboring subregions as boundary conditions for the current subregion problem. In the original formulation, this algorithm is a multiplicative Schwarz method.

Dryja and Widlund [85, 23] developed the additive variant of the Schwarz method. This method is an iterative technique in which the solution is a sum of the projections into the overlapping subspaces, and this modification preserves the symmetric positive-definite property of PDE operators, which allows the additive Schwarz method to be used as a preconditioner for the Conjugate Gradient method in parallel solvers.

Dryja and Widlund introduced an additional subspace, the coarse space given by converting each subspace into a linear element. This substructuring improves the Schwarz method's global transmission of information across the domain and thus provides an improved rate of convergence.

Fischer and Lottes [29, 30] used overlapping additive Schwarz with a coarse grid space as a preconditioner for the Poisson pressure solve in the spectral element formulation of the incompressible Navier-Stokes equations. In this work, the overlap between subdomains was successfully reduced to one or two nodes per subdomain; however, $(p+3)^3 - (p+1)^3$ or $(p+5)^3 - (p+1)^3$, where p is the polynomial order of the high-order or spectral element basis, can be a rather substantial number of nodes that require additional communication across subdomains in the additive Schwarz method. Furthermore, the increase in the subdomain sizes makes each sudomain problem more expensive to solve. Because of this additional communication and more expensive subdomain solves, we investigate non-overlapping domain decomposition techniques.

Specifically, we are interested in the Balancing Domain Decomposition By Constraints (BDDC) method. Dohrmann [20] developed BDDC in 2003 as a non-overlapping domain decomposition technique where an energy minimization problem is solved across subdomain boundaries. This technique was developed as a modification to the earlier Balancing Domain Decomposition (BDD)

by Mandel [60] to help overcome the limitations in BDD when addressing higher order problems, such as fourth-order problems on plates.

BDDC is closely related to the Dual Primal version of Finite Element Tearing and Interconnecting (FETI-DP) developed by Farhat, Lesoinne, and Pierson [27]. BDDC is formulated in terms of the energy minimization problem across subdomain interfaces rather than in terms of Lagrange multipliers, as seen in FETI-DP. Connections have been established between FETI techniques and Neumann-Neumann domain decomposition methods [47], and operators preconditioned with BDDC and FETI-DP have been shown to have the same spectrum, under certain assumptions [61]. In this dissertation, the 'lumped' and 'Dirichlet' nomenclature from FETI-DP is used to describe two variants of the BDDC algorithm. The lumped BDDC variant employs a naive injection operator into the subassembled problem space that is cheaper to setup and apply while the Dirichlet BDDC variant creates a harmonic extension for the jump in values across the broken subdomain interface, which ultimately better controls the condition number of the preconditioned operator.

Fischer, Miller, and Tufo [31] paired their overlapping additive Schwarz preconditioner with a fast approximate subdomain solver based upon the Fast Diagonalization Method (FDM) by Lynch [57]. The use of inexact subdomain solvers for BDDC was explored by Li and Widlund [54] and Dohrmann [21]. In Li and Widlund's work, multigrid v-cycles were used to solve the partially assembled subdomain problems, while we investigate the use of a FDM based subdomain solver.

Specifically, we explore the use of FDM based approximate subdomain solvers for BDDC with subdomains consisting of single high-order finite elements. The analysis of single high-order element subdomains was developed by Pavarino, Widlund, and Zampini, and they demonstrated numerical results for nearly incompressible elasticity problems in three dimensions [68].

With finite elements using high-order tensor product discretizations, the construction of these FDM approximate subdomain solvers is cheaper to set up than factoring the dense matrix representation of these problems, as the FDM approximate subdomain solvers are based around simultaneous diagonalization of the one dimensional mass and stiffness problems. Furthermore, the FDM solvers eliminate the increase in setup cost for Dirichlet BDDC when compared to lumped

BDDC, as solver ingredients can be reused between the subassembled problem solver and the subdomain interior solver used in the harmonic extension problem.

BDDC can be used as a preconditioner for an iterative solver; however we are interested in combining BDDC with p-multigrid. As mentioned above and also shown by Klawonn, Reinbach, and Pavarino [46], multigrid methods have been used to provide effective subdomain solvers for BDDC, but using BDDC as a smoother for multigrid methods is novel. BDDC has improved smoothing properties when compared to polynomial smoothers, which facilitates rapid coarsening in p-multigrid. Additionally, BDDC has reduced communication requirements when compared to polynomial smoothers that deliver comparable smoothing properties with conservative p-coarsening.

1.2.3 Local Fourier Analysis

Local Fourier Analysis (LFA) was developed by Brandt [7, 86] as a powerful tool for predicting multigrid performance and tuning multigrid components by examining the spectral radius and/or condition number of the symbol of the underlying discretized operator. LFA provides sharp predictions for large-scale problems. This analysis has subsequently been extended to finite element discretizations and a wide variety of preconditioning methods. LFA is based on the finite difference stencil or the single element operator matrix for finite element discretizations, which allows accurate parameter tuning for preconditioning techniques with a relatively small amount of computation when compared to the size of the true global problem.

We create a new LFA framework for representing LFA of arbitrary second-order PDEs discretized with high-order finite elements, where the PDE has an arbitrary number of components and the problem is in any number of dimensions.

We develop LFA of p-multigrid with arbitrary second-order PDEs using high-order finite element discretizations. Although van der Vegt and Rhebergen [84] discussed LFA of p-multigrid for the discontinuous Galerkin method, this formulation cannot be extended to the continuous Galerkin method. Our LFA of p-multigrid on high-order finite element discretizations for the continuous Galerkin method is novel work.

LFA of h-multigrid with high-order finite elements was developed in [38] for Lagrange bases with uniformly spaced nodes. We extend our framework to reproduce this previous work on the LFA of h-multigrid for high-order finite element discretizations. Furthermore, our LFA framework can reproduce LFA of finite difference discretizations if the finite difference stencil can be represented by a finite element discretization.

We also develop LFA of lumped and Dirichlet BDDC with arbitrary second-order PDEs using subdomains consisting of single high-order elements with exact and inexact subdomain solvers. The LFA of lumped and Dirichlet BDDC with exact subdomain solvers was introduced by Brown, He, and MacLachlan [13] for subdomains consisting of multiple low-order finite elements, and we extend our framework to reproduce this previous work.

This new LFA of p-multigrid and LFA of BDDC on single high-order element subdomains is combined to provide estimates for p-multigrid with BDDC smoothing.

This work is available in an open source Julia package, LFAToolkit.jl [79]. Several software packages have been developed for LFA of h-multigrid methods [71, 44, 86], however, to the best of our knowledge, no packages support LFA of p-multigrid methods, especially with arbitrary PDEs. We are similarly unaware of packages that provide LFA of BDDC for arbitrary PDEs, for high or low-order element subdomains.

Chapter 2

High-Order Matrix-Free Finite Elements

High-order finite elements implemented in a matrix-free fashion are one way to address the hardware constraints for modern high performance computing (HPC) systems. In this chapter, we consider the performance bottlenecks of modern HPC hardware and demonstrate that high-order matrix-free finite element discretizations address these constraints to help application codes best utilize modern hardware.

In Section 2.1 we explore the limitations of modern HPC hardware and in Section 2.2 we discuss the benefits of high-order matrix-free methods on modern HPC hardware. In Section 2.3 we develop notation to describe high-order finite element discretizations for arbitrary second order PDEs implemented in a matrix-free fashion on unstructured meshes. In Section 2.4 we discuss iterative solvers for matrix-free methods as well as the importance of preconditioning high-order finite element operators for the convergence of these solvers. Finally, in Section 2.5 we provide some examples of practical applications with libCEED that demonstrate the flexibility and applicability of this matrix-free representation with these preconditioned iterative solvers for real-world problems.

2.1 High Performance Computing Hardware Limitations

The developments in HPC hardware over the last thirty years make high-order matrix-free finite element operators increasingly attractive. Two key performance metrics for HPC hardware are Floating Point Operatons per Second (FLOPs) and memory and network bandwidth. FLOPs is the more widely popularized of these two metrics, but memory and network bandwidth is a

common bottleneck in HPC application codes.

The Top 500 [65] list tracks the 500 supercomputers with the highest peak FLOPs, as measured by High-Performance Linpack (HPL) [69]. HPL measures the system performance when solving random dense linear systems in double precision via LU factorization and provides the maximum achievable FLOPs for the machine. The machines on the Top 500 list are approaching exascale FLOP rates, with the goal of surpassing 10^{18} FLOPs.

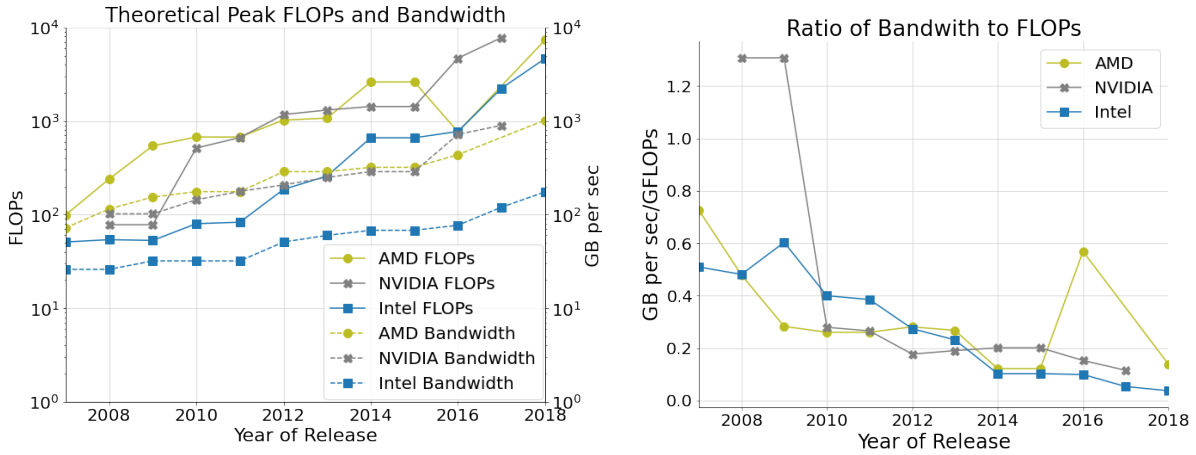
Other benchmarks, such as High-Performance Geometric Multigrid (HPGMG) [3] and High-Performance Conjugate Gradient (HPCG) [22], measure performance based upon solving a more complex benchmark problem. The Top 500 [65] HPCG list achieves far lower peak FLOP rates than the peak FLOP rates seen in the Top 500 HPL list, as shown in Figure 2.1. The disparity between the FLOPs achieved in benchmarks such as HPGMG and HPCG and the peak FLOPs measured by HPL is partially explained by the growing gap between FLOPs and memory and network bandwidth.

Name	HPCG Rank	HPL Rank	HPCG TFLOPs	HPL TFLOPs	Ratio
Fugaku	1	1	16,004.50	442,010.0	3.62%
Summit	2	2	2,925.75	148,600.0	1.97%
Sierra	3	3	1,795.67	94,640.0	1.90%
Selene	4	5	1,622.51	63,460.0	2.56%
JUWELS	5	7	1,275.36	44,120.0	2.89%
Dammam-7	6	10	881.40	22,400.0	3.93%
HPC5	7	8	860.32	35,450.0	2.43%
TOKI-SORA	8	19	614.22	16,592.0	3.70%
Trinity	9	13	546.12	20,158.7	2.71%
Plasma Simulator	10	33	529.16	7,892.7	6.70%

Table 2.1: Top 500 HPCG and Top 500 HPL Comparison

Over the last thirty years, the peak FLOP rates for new HPC hardware have been increasing more rapidly than memory bandwidth and network bandwidth, for both CPUs and GPUs. As discussed in McCalpin's Supercomputing 2016 invited talk [63], peak FLOPs per socket have been increasing at a rate of 50-60% per year while memory bandwidth has only been increasing at a

rate of approximately 23% per year and network bandwidth has only been increasing at a rate of approximately 20% per year. This means that FLOPs have improved approximately twice as much as memory and network bandwidth over the last thirty years. This problem is exacerbated by network latency, which is decreasing at a rate of approximately 20% per year, and memory latency, which is *increasing* at a rate of approximately 20% per year.



(a) Peak FLOPs and Bandwidth

(b) Ratio Bandwidth to FLOPs

Figure 2.1: HPC System Balance

Using data from [75], we can see in Figure 2.1 that AMD, NVIDIA, and Intel top of the line hardware has demonstrated a steady decrease in system balance, the ratio of maximum memory bandwidth to FLOPs, over the last 13 years. HPC applications need to be careful to control the memory bandwidth required so that their codes can better realize the FLOPs capabilities of HPC hardware. High-order matrix-free finite element operators provide one way to reduce memory bandwidth requirements and better utilize FLOPs for HPC application codes on modern hardware.

2.2 Advantages of High-Order Matrix-Free Finite Elements

In this section, we will discuss the benefits of high-order finite element operators implemented in a matrix-free fashion. First we will discuss how matrix-free implementations better agree with system balance on modern HPC hardware. Then we will highlight the spectral convergence prop-

erties of high-order methods.

2.2.1 Storage and FLOPs

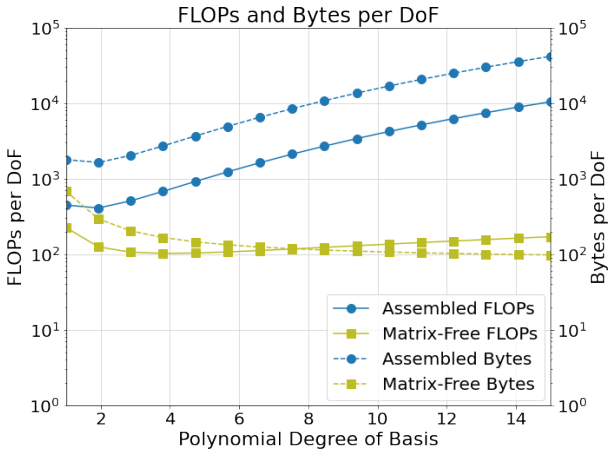
To demonstrate the performance benefits of high-order finite elements implemented in a matrix-free fashion, we consider the specific case of the scalar screened Poisson equation, $\nabla^2 u - \alpha^2 u = f$. We consider the approximate number of bytes required for storage and FLOPs required to apply a matrix-vector product representing a high-order finite element operator corresponding to the Galerkin system for the screened Poisson equation.

High-order finite element discretizations are less common than low-order finite elements, in part, because a linear operator or the Jacobian of a non-linear operator rapidly loses sparsity in a sparse matrix representation. Matrix-free implementations with tensor product finite element bases can overcome this issue by exploiting the tensor product structure to reduce both the data storage requirements and the FLOPs required to apply a matrix-vector product.

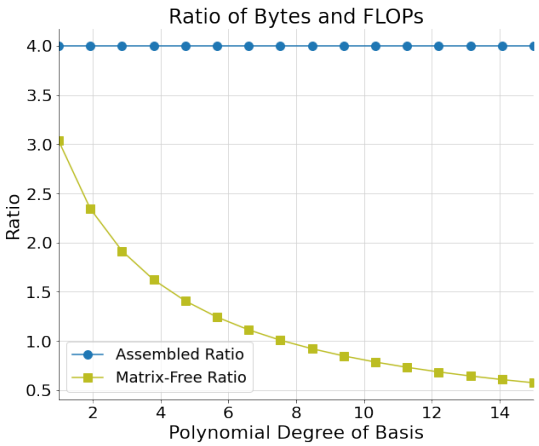
With a sparse matrix representation, application of the finite element operator for a single hexahedral element requires $\mathcal{O}\left((p+1)^6\right)$ matrix entries and $\mathcal{O}\left((p+1)^6\right)$ floating point operations, where p is the polynomial order of the finite element basis. In contrast, application of the matrix-free operator for a single tensor product element requires $\mathcal{O}\left((p+1)^3\right)$ floating point values and $\mathcal{O}\left((p+1)^{d+1}\right)$ FLOPs. When multiple components in the PDE use the same finite element bases, the coefficients in this complexity analysis further favor matrix-free operator implementations.

As seen in Figure 2.2, the balance between bandwidth and FLOPs for matrix-free implementations with tensor product finite elements more closely agrees with the system balance on current HPC hardware shown in Figure 2.1. Matrix-free finite element operator implementations allow for greater arithmetic intensity at high-order, which allows these operators to achieve performance that is closer to the peak performance for this hardware.

It is important to note that generation of high quality hexahedral meshes for tensor product finite elements is a time intensive process when compared to the generation of simplex meshes. However, it is possible to generate meshes comprised predominately, but not necessarily exclusively,



(a) FLOPs and Bytes per DoF



(b) Ratio of Bytes to FLOPs

Figure 2.2: Performance per DoF for Assembled vs Matrix-Free

of high quality hexahedral elements with initial refinement of a simplex mesh without the costly process of fully converting a simplex mesh into only hexahedral elements. Thus, the performance benefits of high-order matrix-free finite elements with tensor product representations can be realized without the substantial additional effort required to generate a mesh exclusively composed of high quality hexahedral elements.

2.2.2 Spectral Accuracy

High-order finite elements offer high accuracy and exponential convergence. The spectral convergence properties of high-order finite elements have been extensively discussed, such as in [36]. In general, the discretization error in the finite element approximation is given by

$$E_{H^1(\Omega)} \leq C\,(p)\,h^{\min(k,p+1)} \tag{2.1}$$

where p is the polynomial order of the mesh, h is the size of the finite elements, and k is the order of the Sobolev space to which the true solution belongs. The coefficient C does depend upon the polynomial order of the finite element basis, but in practice the exponential term dominates this error approximation.

In practical problems, the smoothness of the solution may prevent spectral convergence from being achieved; however, high-order finite elements will still offer convergence that is no worse than the convergence on a comparable low-order mesh with a larger number of elements. In those cases, high-order finite elements implemented in a matrix-free fashion still offer the storage and FLOPs benefits detailed above in Section 2.2.1. For furthe discussion of the convergence of high-order methods, see [5], among others.

Demkowicz, Oden, and Rachowicz, et al. discussed *hp* adaptivity in the context of minimizing the total number of degrees of freedom (DoFs) required to achieve a target accuracy [18, 66, 70]. However, in *hp* adaptivity, the polynomial order of each finite element may differ. While problems utilizing these types of discretizations can be implemented in a matrix-free fashion with the preconditioners discussed in Chapter 4 and Chapter 5, we instead focus on meshes where all elements have the same polynomial order as to simplify the analysis and implementation details.

2.3 Notation for High-Order Matrix-Free Finite Elements

In this section, we develop the notation used for high-order matrix-free finite element operators. The development of this notation will largely follow [11].

As mentioned in Section 2.1, finite element discretizations with tensor product bases offer performance benefits on modern HPC hardware. Therefore, the development of this notation will focus on arbitrary order tensor product bases on unstructured meshes, but this finite element operator representation does not depend upon the type of finite element discretization or the specific mesh used for the PDE and problem under investigation. Simplices or mixed finite element discretizations on arbitrary meshes may be used with this finite element operator representation, although the choice of finite element bases does have performance implications.

2.3.1 High-Order Discretizations

Let $\{X_i\}_{i=1}^{p+1}$ denote the Legendre-Gauss-Lobatto (LGL) nodes of degree p on the reference interval $[-1, 1]$ while $\{\xi_i\}_{i=1}^{q}$ and $\{w_i\}_{i=1}^{q}$ denote the quadrature points and quadrature weights

corresponding to a q point quadrature rule. If we consider Lagrange basis functions $\{\phi_i\}_{i=1}^{p+1}$, we can construct matrices $N_{ij} = \phi_j(\xi_i)$ and $D_{ij} = \partial_x \phi_j(\xi_i)$, representing interpolation of solution values to the quadrature points and computation of derivatives at the quadrature points, respectively. Additionally, the quadrature weight matrix is given by $W_{ij} = w_i \delta_{ij}$.

We can define the corresponding matrices for problems in higher dimensions via tensor products. For the specific case of three dimensions, we have

$$\begin{gathered} \mathbf{N} = N \otimes N \otimes N \\ \mathbf{D}_0 = D \otimes N \otimes N \qquad \mathbf{D}_1 = N \otimes D \otimes N \qquad \mathbf{D}_2 = N \otimes N \otimes D \\ \mathbf{W} = W \otimes W \otimes W \end{gathered} \tag{2.2}$$

The basis operations in Equation 2.2 are defined on a reference element $\hat{K} = [-1, 1]^3$.

In finite element and spectral element methods, we partition the domain Ω into a set of E elements, denoted $\{K^e\}_{e=1}^{E}$ with coordinate mapping to the reference element given by $X : \hat{K} \to K^e$. The Jacobian of this mapping is given by $J_{ij} = \partial x_i / \partial X_j$, where X are the reference coordinates and x the physical coordinates. We can invert the Jacobian and compute the derivatives of the physical coordinates in the reference space at every quadrature point.

$$\mathbf{D}_i^e = \Lambda\left(\frac{\partial X_0}{\partial x_i}\right)\mathbf{D}_0 + \Lambda\left(\frac{\partial X_1}{\partial x_i}\right)\mathbf{D}_1 + \Lambda\left(\frac{\partial X_2}{\partial x_i}\right)\mathbf{D}_2 \tag{2.3}$$

where $\Lambda(X)_{ij} = X_i \delta_{ij}$ expresses pointwise multiplication of J_{ij}^{-1} at quadrature points as a diagonal matrix. With this coordinate mapping, element integration weights become $\mathbf{W}^e = W\Lambda(|J^e(q)|)$.

A global assembly operator is defined as $\mathcal{E}^T = [\mathcal{E}^e]^T$, where $\mathcal{E}^e$ represents local restriction operators extracting degrees of freedom that correspond to element e from the global solution vector. When using an assembled matrix to represent a finite element operator, this global assembly operator is used to construct a global matrix representation of the finite element operator from the individual element operators. With matrix-free implementations, this global assembly operator is instead used to assemble the action of the finite element operator on each element into the action of the finite element operator on the entire mesh. Notice that the local restriction operators do

not assume a structured mesh, a conforming mesh, or consistent polynomial order bases for each element.

With these definitions, we can represent the Galerkin system of equations corresponding to the weak form of arbitrary second-order PDEs. The weak form of PDEs is linear in test functions and can be expressed as pointwise operations where functions of the unknown solution and its derivatives, u and ∇u, are contracted with the test functions and their derivatives, v and ∇v.

Consider the weak form of an arbitrary PDE

$$\begin{gathered} \text{find } u \in V \text{ such that for all } v \in V \\ \langle v, u \rangle = \int_\Omega v \cdot f_0\left(u, \nabla u\right) + \nabla v : f_1\left(u, \nabla u\right) = 0 \end{gathered} \tag{2.4}$$

where $\cdot$ represents contraction over fields and $:$ represents contraction over fields and spatial dimensions. The pointwise representation of the weak form given by f_0 and f_1 does not depend upon discretization choices such as the geometry or polynomial degree of the bases.

The corresponding Galerkin system of equations is

$$\sum_e \mathcal{E}^T \left[\left(\mathbf{N}^e\right)^T \mathbf{W}^e \Lambda\left(f_0\left(u^e, \nabla u^e\right)\right) + \sum_{i=0}^{d-1} \left(\mathbf{D}_i^e\right)^T \mathbf{W}^e \Lambda\left(f_1\left(u^e, \nabla u^e\right)\right) \right] = 0 \tag{2.5}$$

where $u^e = \mathbf{N}^e \mathcal{E}^e u$ and $\nabla u^e = \{\mathbf{D}_i^e \mathcal{E}^e u\}_{i=0}^{d-1}$. In this formulation, the element restriction operators and basis operators can represent different element geometries and different degree polynomial bases, providing a flexible description for arbitrary meshes. Furthermore, this notation can be extended to handle separate fields with different finite element bases, such as with mixed finite element methods.

Dirichlet boundary conditions are represented in the element restriction operation by enforcing the specified values on the constrained nodes. Neumann or Robin boundary conditions are represented by adding boundary integral terms in the same form as Equation 2.5 with appropriate basis and element restriction operators. Boundary integrals internal to the domain Ω, such as face integrals in Discontinuous Galerkin methods, can also be represented using additional terms with corresponding bases and element restrictions.

2.3.2 Linearization

When the PDE in Equation 2.4 is linear, the pointwise functions f_0 and f_1 are also linear and Krylov subspace methods can be used to solve the Galerkin system of equations in Equation 2.5. When the PDE is non-linear, the Galerkin system in Equation 2.5 provides the residual evaluator for a non-linear solver and Jacobian of this non-linear residual can be represented in a similar fashion as Equation 2.5, based upon the weak form

$$\langle v, J\left(u\right) w\rangle = \int_{\Omega} \begin{bmatrix} v^T & \left(\nabla v\right)^T \end{bmatrix} \begin{bmatrix} f_{0,0} & f_{0,1} \\ f_{1,0} & f_{1,1} \end{bmatrix} \begin{bmatrix} w \\ \nabla w \end{bmatrix} \tag{2.6}$$

where $f_{i,0} = \frac{\partial f_i}{\partial u}\left(u, \nabla u\right)$ and $f_{i,1} = \frac{\partial f_i}{\partial \nabla u}\left(u, \nabla u\right)$. If these pointwise functions $f_{i,j}$ are not available analytically, they can be computed via algorithmic differentiation or finite differencing.

With these pointwise functions, we can describe Jacobian-free Newton-Krylov methods for solving non-linear PDEs. Jacobian-free Newton-Krylov methods were summarized, with preconditioning strategies, by Knoll and Keyes in [48].

2.3.3 Computational Representation

We can represent the systems of equations in Equation 2.5 and Equation 2.6 as a series of efficient computational kernels. This streamlined representation is used by libCEED [1] to provide performance portable implementations for high-order matrix-free finite element operators. In Chapter 3, we use this representation to develop LFA of high-order finite element operators and preconditioning methods.

The action of an arbitrary second-order finite element operator can be represented as

$$\mathbf{A} = \mathbf{P}^T\mathbf{G}^T\mathbf{B}^T\mathbf{D}\mathbf{B}\mathbf{G}\mathbf{P} \tag{2.7}$$

where $\mathbf{P}$ represents the parallel communication portion of the global element restriction operator $\mathcal{E}$, $\mathbf{G}$ represents the device portion of the global element restriction operator, $\mathbf{B}$ represents the basis action kernel that provides solution values and derivatives at the quadrature points $\mathbf{N}$

and $\mathbf{D}_i$, and $\mathbf{D}$ represents the pointwise representation of the weak form, given by f_i or $f_{i,j}$ and the element quadrature weights $\mathbf{W}$ and geometric factors $J_{i,j} = \partial x_i / \partial X_j$.

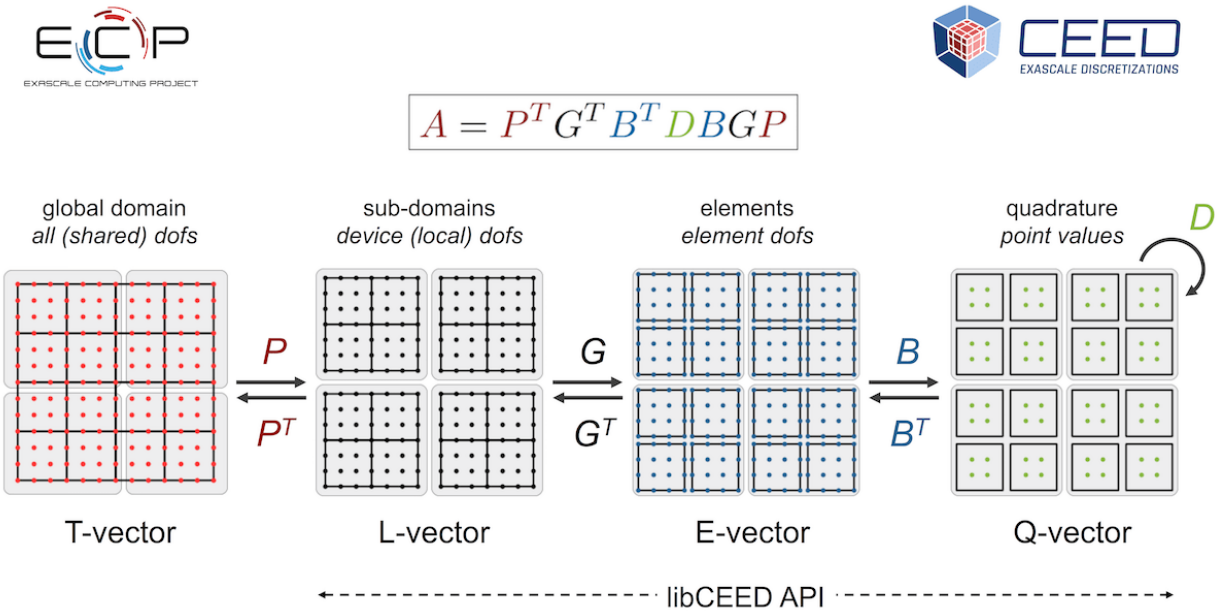


Figure 2.3: libCEED API

This representation is shown graphically in Figure 2.3, where we can see the different spaces that each kernel operates upon. This flexible kernel representations will be used in Chapter 3, where we develop LFA of finite element operators that can be represented in this fashion. **Burgundy** text will be used to represent operators that can be implemented in this matrix-free representation.

2.4 Iterative Solvers for Matrix-Free Finite Elements

Direct solver techniques such as sparse LU factorization or Algebraic Multigrid (AMG) are not available for high-order finite element operators implemented in a matrix-free fashion, as the these techniques require an assembled matrix, which defeats the purpose of using matrix-free implementations. In this section, we will discuss iterative solver techniques for linear and linearized problems and the importance of preconditioning for these techniques.

2.4.1 Krylov Subspace Methods

Krylov subspace methods are a natural fit for matrix-free finite elements, as these methods only require matrix-vector products to populate the Krylov subspace

$$\mathcal{K}\left(\mathbf{r}_0, \mathbf{A}, \mathbf{k}\right) = \{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \ldots, \mathbf{A}^{k-1}\mathbf{r}_0\}. \tag{2.8}$$

The Krylov subspace is used to construct increasingly accurate iterates $\mathbf{u}_k$ that approach the true solution $\mathbf{u}$.

Specifically, we focus on the Conjugate Gradient (CG) method [39, 77] for solving symmetric positive-definite linear systems. CG relies upon the observation that for symmetric positive-definite systems, solving the linear problem $\mathbf{A}\mathbf{u} = \mathbf{b}$ is equivalent to minimizing the quadratic form given by

$$\phi\left(\mathbf{u}\right) = \frac{1}{2}\mathbf{u}^T\mathbf{A}\mathbf{u} - \mathbf{u}^T\mathbf{b}. \tag{2.9}$$

The gradient of this quadratic form is given by $\phi'\left(\mathbf{u}_k\right) = \mathbf{b} - \mathbf{A}\mathbf{u}_k$. This gradient provides the direction of steepest descent, so CG chooses the appropriate step size in the direction of the gradient, which is equivalent to choosing the appropriate weight for each basis vector in the current Krylov subspace.

In the method of Steepest Descent, the iterates $\mathbf{u}^k$ are given by traversing the direction of steepest descent until the gradient of the orthogonal to the search direction.

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \alpha_k\mathbf{u}_k, \quad \alpha_k = \frac{\mathbf{r}_k^T\mathbf{r}_k}{\mathbf{r}_k^T\mathbf{A}\mathbf{r}_k} \tag{2.10}$$

The convergence of the method of Steepest Descent can be slow, especially when the condition number of the operator is large, such as with high order finite element operators.

CG improves upon Steepest Descent by taking the search directions to be A-orthogonal, which helps improve convergence.

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \alpha_k + \mathbf{d}_k, \quad \alpha_k = \frac{\mathbf{r}_k^T\mathbf{r}_k}{\mathbf{d}_k^T\mathbf{A}\mathbf{d}_k}, \quad \mathbf{d}_{k+1} = \mathbf{r}_{k+1} + \beta_{k+1}\mathbf{d}_k, \quad \beta_{k+1} = \frac{\mathbf{r}_k^T\mathbf{r}_k}{\mathbf{r}_k^T\mathbf{r}_k} \tag{2.11}$$

Detailed convergence analysis for CG is complex and we omit it here. In general though, operators with a highly elliptic spectum, which results in a high condition number, have poorer

convergence with CG, while operators, or preconditioned operators, that are closer to the identity operator converge much more quickly [34].

There are more flexible Krylov methods that work for a more general set of operators, such as Generalized Minimum Residual (GMRES). Our preconditioners in Chapter 4 and Chapter 5 maintain the symmetric positive-definite property required for CG, but they are also applicable to a wider range of Krylov methods or iterative solvers, such as GMRES.

2.4.2 Preconditioning Requirements

The iteration count to reach convergence for Krylov subspace methods is based, in part, upon condition number of the operator [56], and high-order finite element operators have notoriously poor condition numbers [42]. Preconditioners help control the condition number of high-order finite elements implemented in a matrix-free fashion and therefore reduce total iteration count and total time to solution for these operators.

Suppose we are solving the linear system given by

$$\mathbf{A}\mathbf{u} = \mathbf{b} \tag{2.12}$$

via a Krylov subspace method. This linear system may come from the Galerkin system of equations of our PDE of interest or the Jacobian of our PDE of interest, as in a Newton-Krylov method. We can improve the convergence of our Krylov iterations for this system by solving the preconditioned system

$$\left(\mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1}\right)\left(\mathbf{M}_R\mathbf{u}\right) = \mathbf{M}_L^{-1}\mathbf{b} \tag{2.13}$$

via our Krylov method instead. We will focus on left preconditioning, where $\mathbf{M}_R = \mathbf{I}$ and $\mathbf{M}_L^{-1} \approx \mathbf{A}^{-1}$. Therefore, we will adopt the notation $\mathbf{M}_L = \mathbf{M}$.

Specifically, we will investigate preconditioning techniques that can implemented predominately or exclusively in a matrix-free fashion, again to retain the performance benefits offered by high-order matrix-free finite element discretizations on modern HPC hardware.

Note that preconditioning CG is more restrictive than other Krylov methods as the CG algorithm requires preconditioners to preserve the symmetric positive-definite property of the operator representing the Galerkin system. As a result, preconditioning techniques used for CG can also be applied to more general Krylov methods, such as GMRES.

We consider multigrid and domain decomposition preconditioners, with a specific focus on p-multigrid and Dirichlet Balancing Domain Decomposition by Constraints. Both of these techniques can be used as solvers on their own or as preconditioners to accelerate the convergence of Krylov methods.

2.5 Applications for Matrix-Free Finite Elements

Several mini-applications are included with libCEED [1] to demonstrate the flexibility and applicability of this matrix-free representation for real-world problems. In this section, we explore three specific libCEED mini-applications, the Center for Efficient Exascale Discretizations (CEED) benchmarks, the Navier-Stokes fluid dynamics example, and the Neo-Hookean hyperelasticity solid mechanics example.

2.5.1 Performance Benchmarking

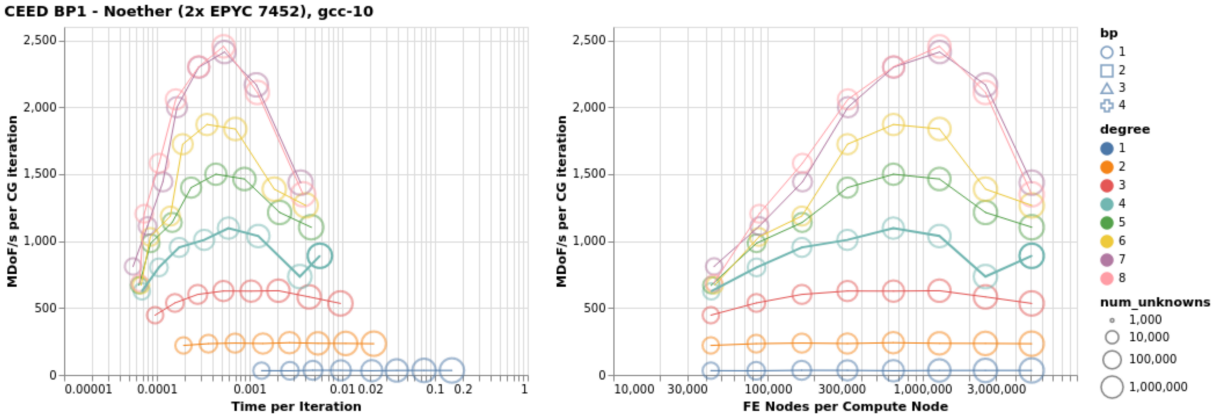
The CEED uses Bakeoff Problems (BPs) to test and compare the performance of high-order finite element implementations. The definitions of these bakeoff problems are given on the CEED website [15] and are summarized in Table 2.2.

In these BPs, the stiffness matrix is given by the Laplacian. These problems use nodal bases on the Gauss-Legendre-Lobatto points with order p nodal basis functions with q quadrature points in each dimension for a total of $(p+1)^d$ nodes and q^d quadrature points, where d is the dimension of the problem. For BP5 and BP6 the quadrature points are collocated with the nodal points. Since the collocated quadrature is appropriate for a smaller range of practical applications, we will focus on results for BP1, BP2, BP3, and BP4, on both CPU and GPU hardware.

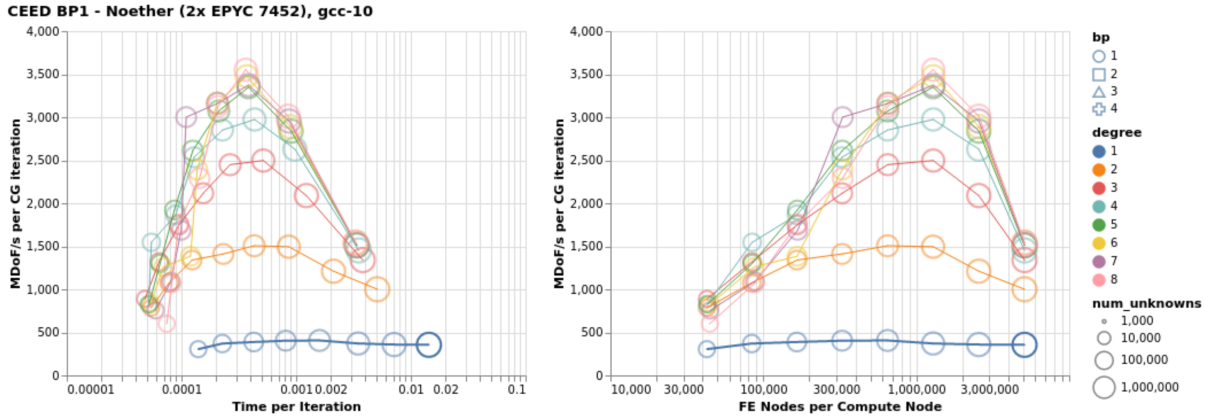
The CPU performance study was conducted on a two-socket AMD EPYC 7452 machine.

BP	Problem	components	quadrature points
BP1	mass matix	1	$q = p + 2$
BP2	mass matix	3	$q = p + 2$
BP3	stiffness matix	1	$q = p + 2$
BP4	stiffness matix	3	$q = p + 2$
BP5	stiffness matix	1	$q = p + 1$
BP6	stiffness matix	3	$q = p + 1$

Table 2.2: CEED Bakeoff Problems



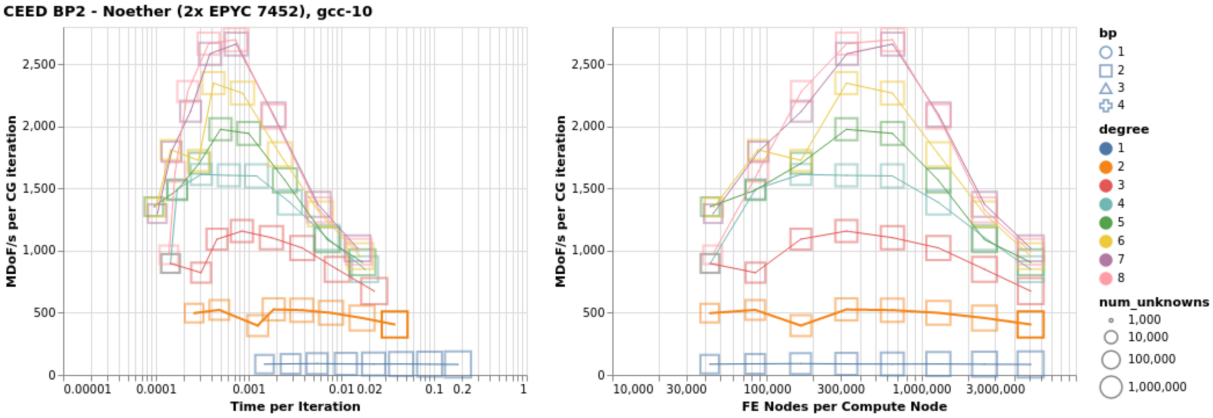
(a) Serial XSMM Backend



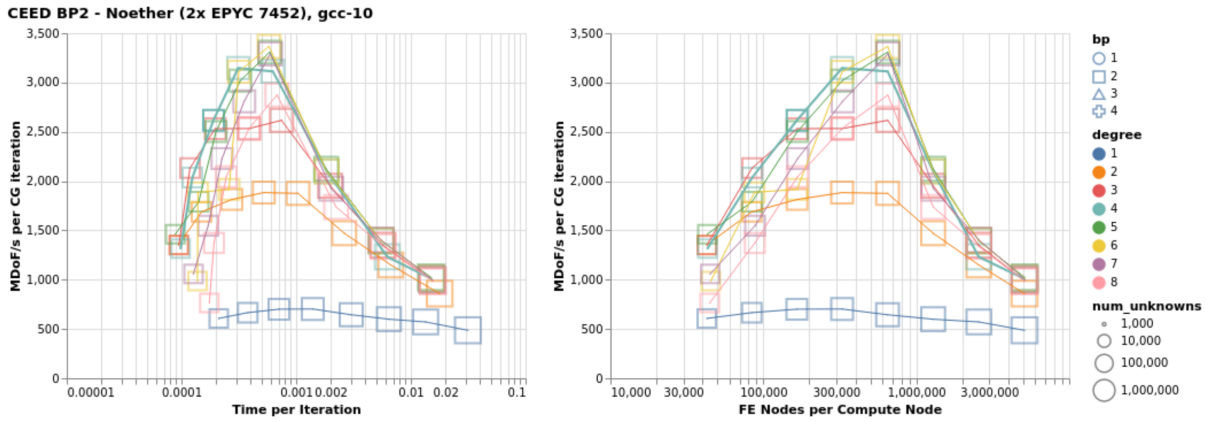
(b) Blocked XSMM Backend

Figure 2.4: CEED Benchmark BP1 - AMD EPYC

Figures 2.4, 2.5, 2.6, and 2.7 show the results of this performance study. Each socket contains 32 CPU cores with a base clock speed of 2.35 GHz and 128 MB of L3 cache. The NPS4 BIOS



(a) Serial XSMM Backend

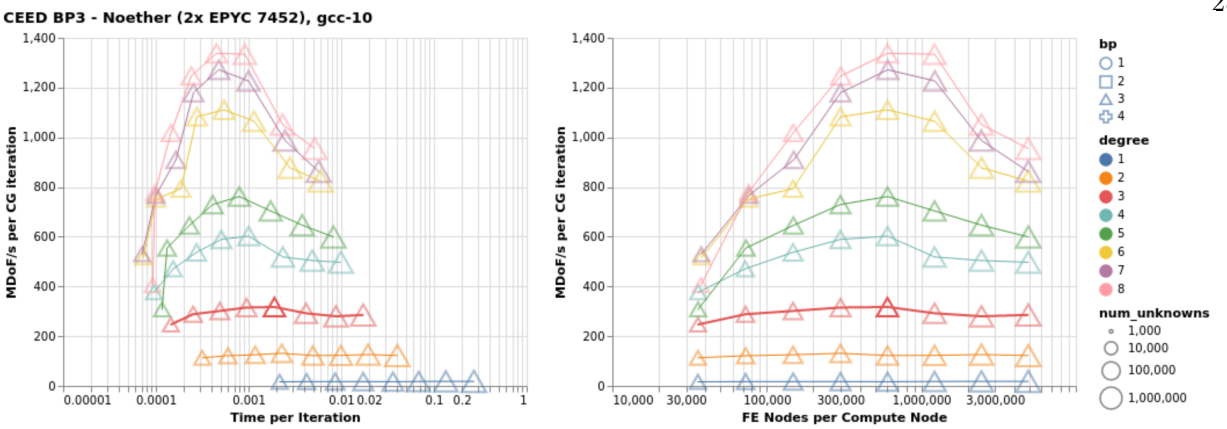


(b) Blocked XSMM Backend

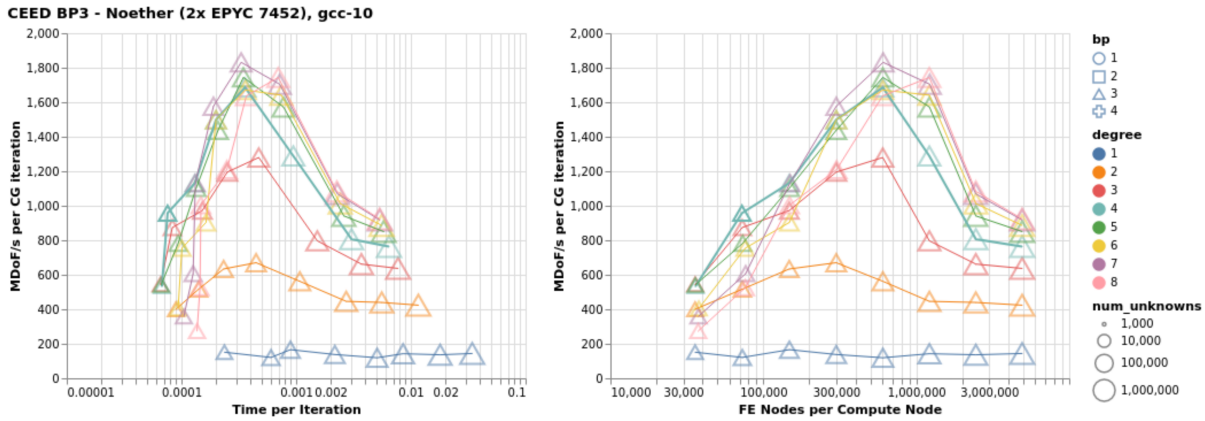
Figure 2.5: CEED Benchmark BP2 - AMD EPYC

configuration was used and processes were bound to cores. MPICH-3.3.2 was used with PETSc [6] version 3.15 and libCEED [12] version 0.9.

libCEED used the serial and blocked LIBXSMM [55] backends, \cpu\self\xsmm\serial and \cpu\self\xsmm\blocked. In the serial backends, elements are processed in sequence and operations are vectorized across the data for a single element while in the blocked backends elements are processed in interlaced 8 element batches to facilitate vectorization. Both backends use LIBXSMM Just in Time (JiT) compilation of small matrix-matrix product kernels using Advanced Vector Extensions (AVX) instructions to provide high-performance tensor product evaluations on CPU architecture.



(a) Serial XSMM Backend

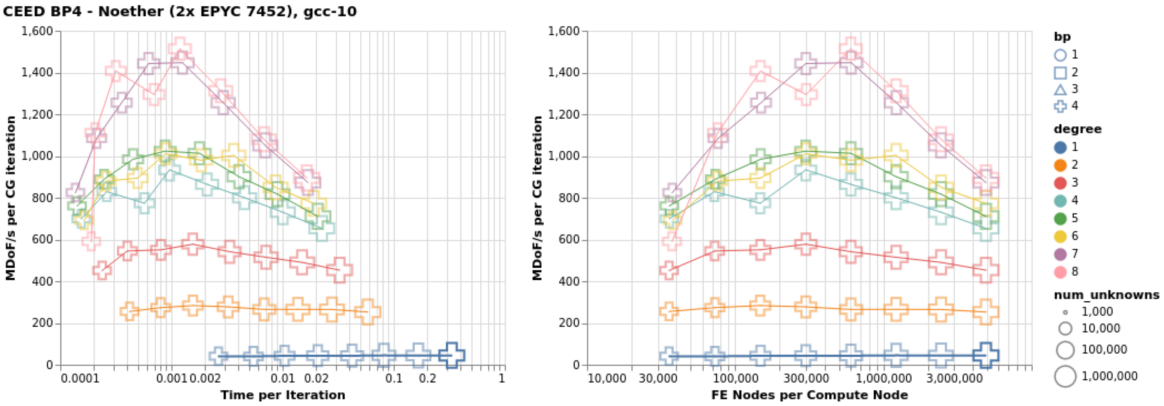


(b) Blocked XSMM Backend

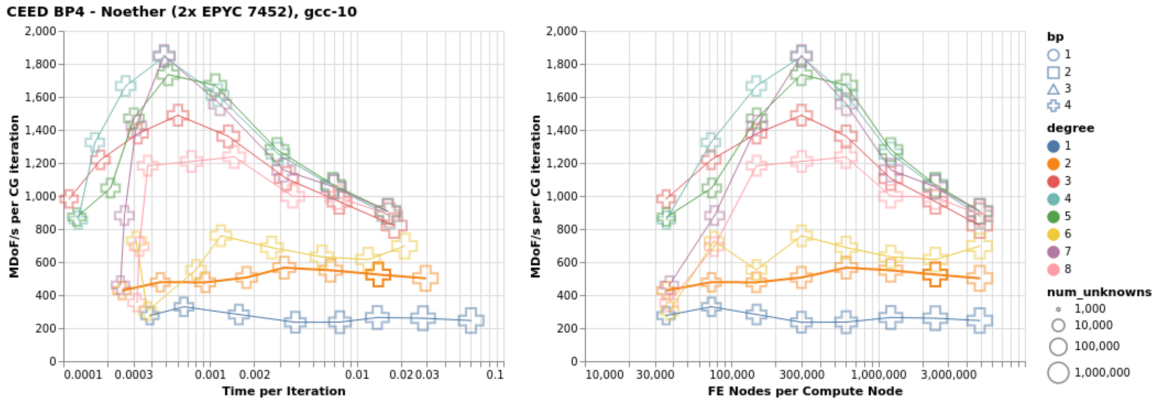
Figure 2.6: CEED Benchmark BP3 - AMD EPYC

Figure 2.4 shows the results for CEED BP1. As anticipated, we see higher throughput as the polynomial order of the basis increases. The high throughput with low latency indicates that this architecture demonstrates good strong scaling, while high throughput with high latency would indicate that the architecture weak scales effectively.

In Figure 2.5 we see the results for CEED BP2. For this problem, throughput starts to drop off for the blocked backend as the polynomial order of the bases increases above $p = 6$. This is because the 8 element batch on the 3 component vector problem is large enough to spill out of the cache on this architecture. This is an indication that for above this polynomial order the single element batching strategy used by the serial element processing LIBXSMM backend,



(a) Serial XSMM Backend



(b) Blocked XSMM Backend

Figure 2.7: CEED Benchmark BP4 - AMD EPYC

\cpu\self\xsmm\serial, is more appropriate.

In Figure 2.6 and Figure 2.7, we see a similar behavior, but for the more computationally intensive Laplacian rather than the mass matrix. In this case, the throughput in the 3 component vector problem drops off after the polynomial order of the bases increases above $p = 5$.

The GPU performance study was conducted on a single node with two-socket IBM POWER9 and four NVIDIA Volta V100s on the Lassen system at Lawrence Livermore National Laboratory. Figures 2.8, 2.9, 2.10, and 2.11 show the results of this performance study. Each socket contains 22 CPU cores with a base clock speed of 3.45 GHz and each GPU has 5120 CUDA cores with a total

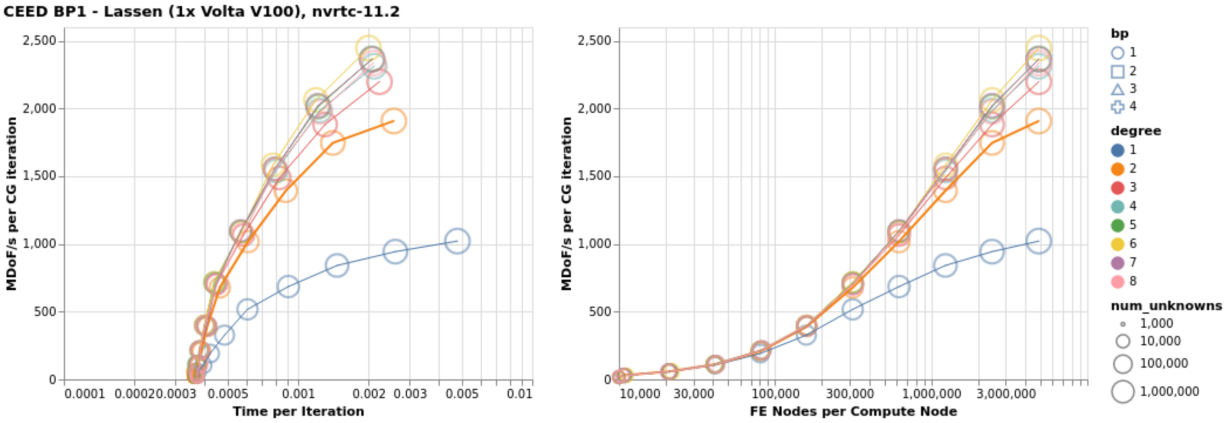


Figure 2.8: CEED Benchmark BP1 - NVIDIA V100

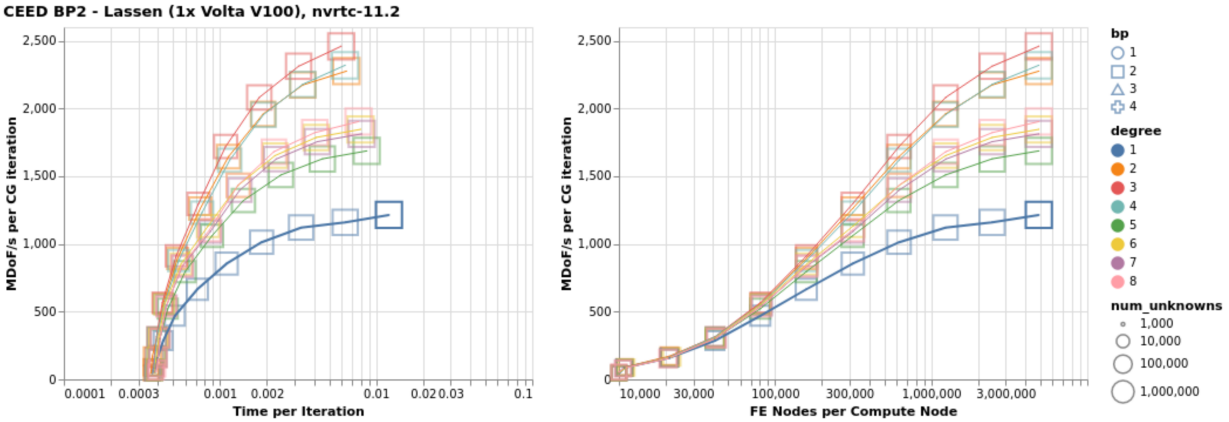


Figure 2.9: CEED Benchmark BP2 - NVIDIA V100

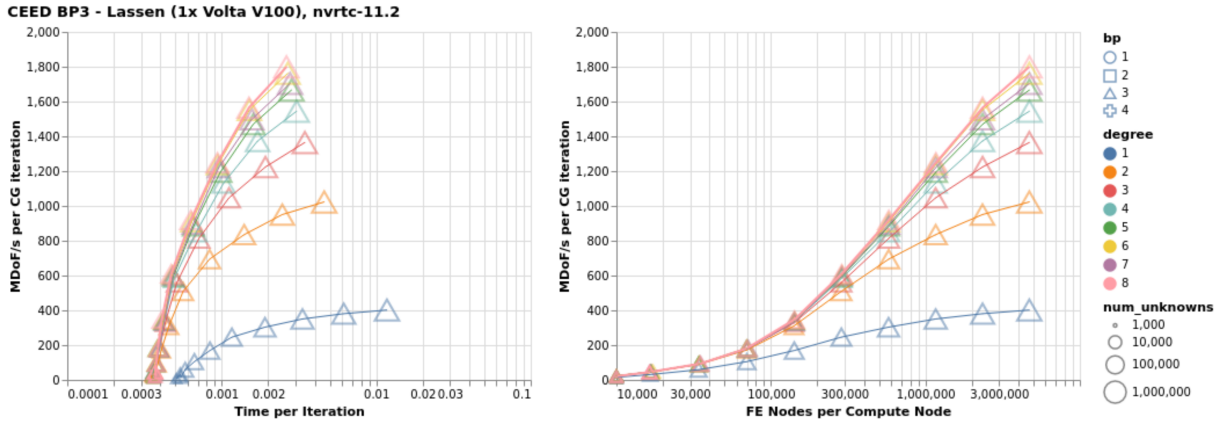


Figure 2.10: CEED Benchmark BP3 - NVIDIA V100

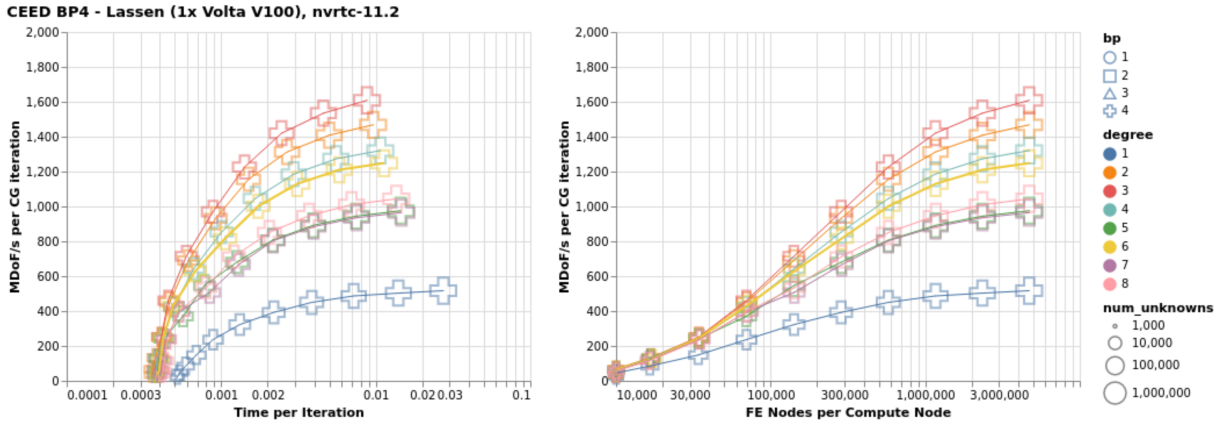


Figure 2.11: CEED Benchmark BP4 - NVIDIA V100

of 256 GB memory for each node. CUDA-aware Spectrum MPI was used with PETSc [6] version 3.15 and libCEED [12] version 0.9.

libCEED used the CUDA code generation backend, \gpu\cuda\gen. In the code generation backends, the user source code for the application of the weak form at the quadrature points **D** is JiT compiled into a single kernel that represents the action of the full operator **A**. This approach reduces latency from additional kernel launches and helps provide the best performance on NVIDIA and AMD GPU hardware.

As before, throughput increases as polynomial order increases. The high throughput with high latency indicates that the architecture weak scales effectively, while high throughput with low latency would indicate that this architecture demonstrates good strong scaling.

The specific performance tradeoffs depend upon the hardware, and these benchmark problems help inform users about the performance capabilities of their hardware for high-order matrix-free representations of PDE operators.

2.5.2 Fluid Dynamics

The libCEED [1] fluid dynamics mini-application solves the time-dependent Navier-Stokes equations of compressible gas dynamics in a static Eulerian three-dimensional frame using unstructured high-order finite element spatial discretizations and explicit or implicit high-order time-

stepping. Moreover, since the Navier-Stokes example has been developed using PETSc, the pointwise physics represented in the quadrature point function **D** are separated from the parallelization, meshing, and linear and non-linear solvers.

The mathematical formulation used in the fluid dynamics mini-application is similar to the discussion in [33]. The compressible Navier-Stokes equations in conservative form are given by

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \nabla \cdot \mathbf{U} &= 0 \\ \frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \left(\frac{\mathbf{U} \otimes \mathbf{U}}{\rho} + P\mathbf{I}_3 - \sigma \right) + \rho g \hat{\mathbf{k}} &= 0 \\ \frac{\partial E}{\partial t} + \nabla \cdot \left(\frac{(E+P)\mathbf{U}}{\rho} - \mathbf{u} \cdot \sigma - k \nabla T \right) &= 0 \,, \end{aligned} \tag{2.14}$$

where $\boldsymbol{\sigma} = \mu \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T + \lambda (\nabla \cdot \mathbf{u}) \mathbf{I}_3 \right)$ is the Cauchy (symmetric) stress tensor, μ is the dynamic viscosity coefficient, and $\lambda = -2/3$ is the Stokes hypothesis constant. In Equation 2.14, ρ represents the volume mass density, $\mathbf{U}$ the momentum density (defined as $\mathbf{U} = \rho\mathbf{u}$, where $\mathbf{u}$ is the vector velocity field), E the total energy density (defined as $E = \rho e$, where e is the total energy), $\mathbf{I}_3$ represents the 3×3 identity matrix, g the gravitational acceleration constant, $\hat{\mathbf{k}}$ the unit vector in the z direction, k the thermal conductivity constant, T represents the temperature, and P the pressure. Pressure is given by the equation of state

$$P = (c_p/c_v - 1)\left(E - \mathbf{U} \cdot \mathbf{U}/(2\rho) - \rho g z\right) \,, \tag{2.15}$$

where c_p is the specific heat at constant pressure and c_v is the specific heat at constant volume, which defines the specific heat ratio, $\gamma = c_p/c_v$.

The system of equations in Equation 2.14 can be rewritten in vector form

$$\frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}) - S(\mathbf{q}) = 0 \,, \tag{2.16}$$

for the state variables 5-dimensional vector

$$\mathbf{q} = \begin{pmatrix} \rho & \leftarrow \text{ volume mass density} \\ \mathbf{U} \equiv \rho\mathbf{u} & \leftarrow \text{ momentum density} \\ E \equiv \rho e & \leftarrow \text{ energy density} \end{pmatrix} \,, \tag{2.17}$$

where the flux and the source terms, respectively, are given by

$$\mathbf{F}(\mathbf{q}) = \begin{pmatrix} \mathbf{U} \\ (\mathbf{U} \otimes \mathbf{U})/\rho + P\mathbf{I}_3 - \boldsymbol{\sigma} \\ (E+P)\mathbf{U}/\rho - \mathbf{u}\cdot\boldsymbol{\sigma} - k\nabla T \end{pmatrix}, \quad S(\mathbf{q}) = -\begin{pmatrix} 0 \\ \rho g\hat{\mathbf{k}} \\ 0 \end{pmatrix}. \tag{2.18}$$

The discrete solution is therefore given by

$$\mathbf{q}_N(\mathbf{x},t)^{(e)} = \sum_{k=1}^{p+1} \phi_k(\mathbf{x})\mathbf{q}_k^{(e)}, \tag{2.19}$$

where $p+1$ is the number of nodes on the element e and ϕ represents the finite elment basis functions.

For the time discretization, we use two types of time-stepping schemes, explicit and implicit.

2.5.2.1 Explicit Time-Stepping Formulation

The explicit formulation for the adaptive Runge-Kutta-Fehlberg (RKF) method is given by

$$\mathbf{q}_N^{n+1} = \mathbf{q}_N^n + \Delta t \sum_{i=1}^{s} b_i k_i \,, \tag{2.20}$$

where

$$\begin{aligned} k_1 &= f(t^n, \mathbf{q}_N^n) \\ k_2 &= f(t^n + c_2\Delta t, \mathbf{q}_N^n + \Delta t(a_{21}k_1)) \\ k_3 &= f(t^n + c_3\Delta t, \mathbf{q}_N^n + \Delta t(a_{31}k_1 + a_{32}k_2)) \\ &\vdots \\ k_i &= f\left(t^n + c_i\Delta t, \mathbf{q}_N^n + \Delta t\sum_{j=1}^{s} a_{ij}k_j\right) \end{aligned} \tag{2.21}$$

and with

$$f(t^n, \mathbf{q}_N^n) = -[\nabla\cdot\mathbf{F}(\mathbf{q}_N)]^n + [S(\mathbf{q}_N)]^n \,. \tag{2.22}$$

Note that any explicit time-stepping scheme available in PETSc can be chosen at runtime.

2.5.2.2 Implicit Time-Stepping Formulation

The implicit formulation solves nonlinear systems for $\mathbf{q}_N$, given by

$$\mathbf{f}(\mathbf{q}_N) \equiv \mathbf{g}(t^{n+1}, \mathbf{q}_N, \dot{\mathbf{q}}_N) = 0\,, \tag{2.23}$$

where the time derivative $\dot{\mathbf{q}}_N$ is defined by

$$\dot{\mathbf{q}}_N(\mathbf{q}_N) = \alpha \mathbf{q}_N + \mathbf{z}_N \tag{2.24}$$

in terms of $\mathbf{z}_N$ from prior state and $\alpha > 0$, both of which depend on the specific time integration scheme.

To determine how difficult a given non-linear time integration problem is to solve, consider the Jacobian of Equation 2.23,

$$\frac{\partial \mathbf{f}}{\partial \mathbf{q}_N} = \frac{\partial \mathbf{g}}{\partial \mathbf{q}_N} + \alpha \frac{\partial \mathbf{g}}{\partial \dot{\mathbf{q}}_N}. \tag{2.25}$$

The scalar shift α scales inversely with the time step Δt, so small time steps result in the Jacobian being dominated by the second term, which is a sort of "mass matrix", and typically well-conditioned independent of grid resolution with a simple preconditioner. In contrast, the first term dominates for large time steps, with a condition number that grows with the diameter of the domain and polynomial degree of the approximation space. Both terms are significant for time-accurate simulation and the setup costs of strong preconditioners must be balanced with the convergence rate of Krylov methods using weak preconditioners.

To obtain a finite element discretization, we first multiply the strong form in Equation 2.16 by a test function $\mathbf{v} \in H^1(\Omega)$ and integrate,

$$\int_\Omega \mathbf{v} \cdot \left(\frac{\partial \mathbf{q}_N}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}_N) - \mathbf{S}(\mathbf{q}_N) \right) dV = 0\,,\ \forall \mathbf{v} \in \mathcal{V}_p\,, \tag{2.26}$$

with $\mathcal{V}_q = \{\mathbf{v}(\mathbf{x}) \in H^1(\Omega_e) \,|\, \mathbf{v}(\mathbf{x}_e(\mathbf{X})) \in P_q(\mathbf{I}), e = 1, \ldots, N_e\}$ a mapped space of polynomials containing at least polynomials of degree q, with or without the higher mixed terms that appear in tensor product spaces.

Integrating by parts on the divergence term, we arrive at the weak form,

$$\int_{\Omega} \mathbf{v} \cdot \left(\frac{\partial \mathbf{q}_N}{\partial t} - \mathbf{S}(\mathbf{q}_N) \right) dV - \int_{\Omega} \nabla \mathbf{v} : \mathbf{F}(\mathbf{q}_N) \, dV + \int_{\partial \Omega} \mathbf{v} \cdot \mathbf{F}(\mathbf{q}_N) \cdot \hat{\mathbf{n}} \, dS = 0 \,, \; \forall \mathbf{v} \in \mathcal{V}_p \,, \tag{2.27}$$

where $\mathbf{F}(\mathbf{q}_N) \cdot \hat{\mathbf{n}}$ is typically replaced with a boundary condition. We can solve Equation 2.27 with a Galerkin discretization or a stabilized formulation.

2.5.2.3 Ongoing Research

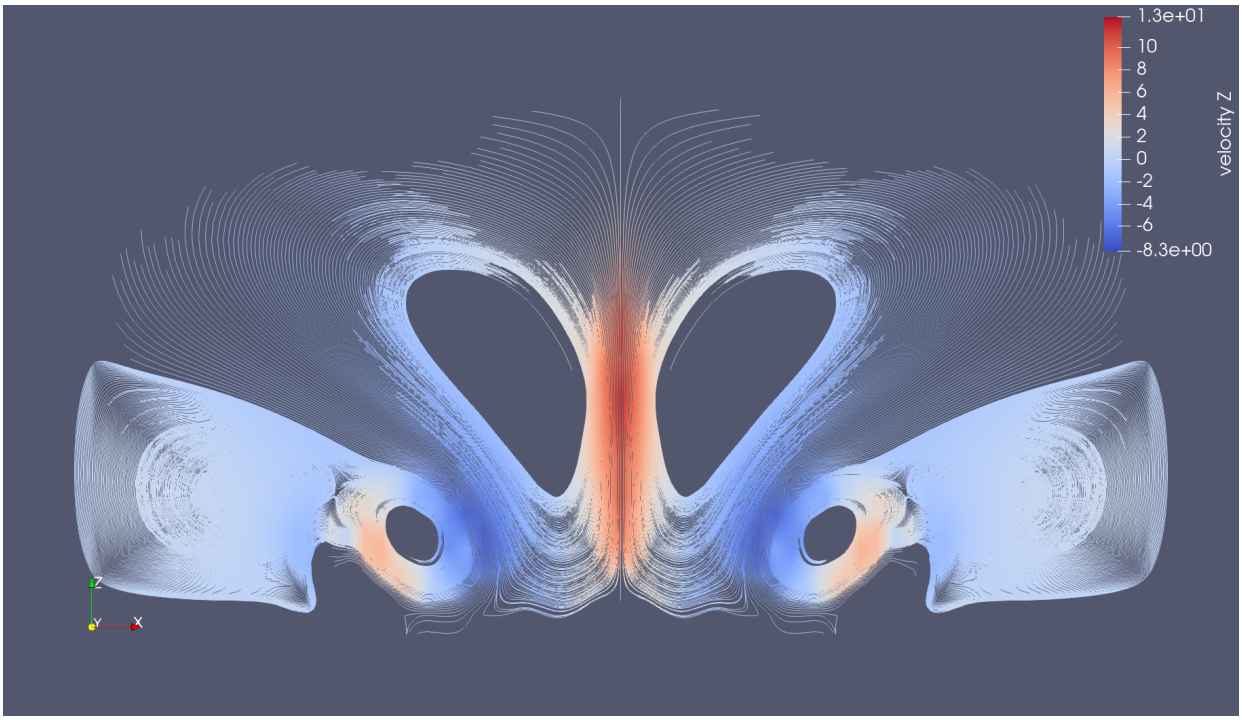


Figure 2.12: Fluid Dynamics - Vortices Developing as a Cold Air Bubble Drops to the Ground

Figure 2.12 shows an atmospheric simulation where vortices develop as a cold air bubble drops to the ground. This mini-application can run on host or device processors and supports stabilization techniques such as streamline-upwind (SU) and streamline-upwind Petrov-Galerkin (SUPG). Ongoing research includes the stabalization techniques and preconditioning.

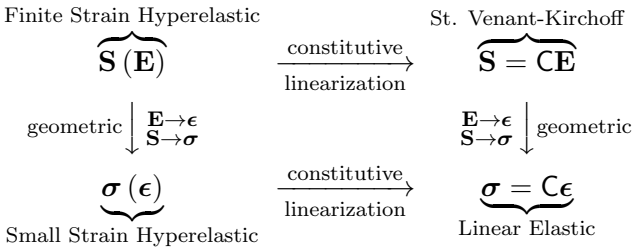


Figure 2.13: Linearization of Hyperelasticitic Model

2.5.3 Solid Mechanics

The solid mechanics mini-application solves the steady-state static momentum balance equations using unstructured high-order finite element spatial discretizations. As with the fluid dynamics mini-application, the solid mechanics elasticity mini-application has been developed using PETSc so that the pointwise physics D are separated from the parallelization, meshing, and solver concerns. This mini-application uses p-multigrid preconditioning as described in Chapter 4.

In this mini-application, we consider three formulations used in solid mechanics applications: linear elasticity, Neo-Hookean hyperelasticity at small strain, and Neo-Hookean hyperelasticity at finite strain. We provide the strong and weak forms of the static balance of linear momentum in the small strain and finite strain regimes below. The stress-strain relationship for each of the material models is provided. Due to the nonlinearity of material models in Neo-Hookean hyperelasticity, the Newton linearization of the material models is also provided.

Linear elasticity and small-strain hyperelasticity can both by obtained from the finite-strain hyperelastic formulation by linearization of geometric and constitutive nonlinearities. The effect of these linearizations is sketched in Figure 2.13, where $\boldsymbol{\sigma}$ and $\boldsymbol{\epsilon}$ are stress and strain, respectively, in the small strain regime, while $\mathbf{S}$ and $\mathbf{E}$ are their finite-strain generalizations, the second Piola-Kirchoff tensor and Green-Lagrange strain tensor, respectively, defined in the initial configuration, and C is a linearized constitutive model.

2.5.3.1 Hyperelasticity at Finite Strain

In the total Lagrangian approach for the Neo-Hookean hyperelasticity problem, the discrete equations are formulated with respect to the initial configuration. In this formulation, we solve for displacement $\mathbf{u}(\mathbf{X})$ in the reference frame $\mathbf{X}$. The notation for elasticity at finite strain is inspired by [41] to distinguish between the current and initial configurations. We denote the reference frame by capital letters and the current frame by lower case letters.

The strong form of the static balance of linear-momentum at finite strain is given by

$$-\nabla_X \cdot \mathbf{P} - \rho_0 \mathbf{g} = \mathbf{0} \tag{2.28}$$

where the ∇_X indicates that the gradient is calculated with respect to the initial configuration in the finite strain regime. $\mathbf{P}$ is the first Piola-Kirchhoff stress tensor and $\mathbf{g}$ is the prescribed forcing function, while ρ_0 gives the initial mass density. The tensor $\mathbf{P}$ is not symmetric, living in the current configuration on the left and the initial configuration on the right.

The first Piola-Kirchoff stress tensor can be decomposed as

$$\mathbf{P} = \mathbf{F}\,\mathbf{S}, \tag{2.29}$$

where $\mathbf{S}$ is the second Piola-Kirchhoff stress tensor, a symmetric tensor defined entirely in the initial configuration, and $\mathbf{F} = \mathbf{I}_3 + \nabla_X \mathbf{u}$ is the deformation gradient. Different constitutive models can define $\mathbf{S}$.

For the constitutive modeling of hyperelasticity at finite strain, we begin by defining two symmetric tensors in the initial configuration, the right Cauchy-Green tensor

$$\mathbf{C} = \mathbf{F}^T \mathbf{F} \tag{2.30}$$

and the Green-Lagrange strain tensor

$$\mathbf{E} = \frac{1}{2}\left(\mathbf{C} - \mathbf{I}_3\right) = \frac{1}{2}\left(\nabla_X \mathbf{u} + \left(\nabla_X \mathbf{u}\right)^T + \left(\nabla_X \mathbf{u}\right)^T \nabla_X \mathbf{u}\right). \tag{2.31}$$

The Green-Lagrange strain tensor converges to the linear strain tensor $\boldsymbol{\epsilon}$ in the small-deformation limit. The constitutive models we consider express $\mathbf{S}$ as a function of $\mathbf{E}$ and are appropriate for large deformations.

In their most general form, constitutive models define $\mathbf{S}$ in terms of state variables. For the model used in the present mini-application, the state variables are constituted by the vector displacement field $\mathbf{u}$ and its gradient $\nabla\mathbf{u}$.

The constitutive model $\mathbf{S}(\mathbf{E})$ is a tensor-valued function of a tensor-valued input. An arbitrary choice of such a function will generally not be invariant under orthogonal transformations and thus will not admissible as a physical model must not depend on the coordinate system chosen to express it. In particular, given an orthogonal transformation Q, we require

$$Q\mathbf{S}(\mathbf{E})\,Q^T = \mathbf{S}\left(Q\mathbf{E}Q^T\right), \tag{2.32}$$

which means that we can change our reference frame before or after computing $\mathbf{S}$ and get the same result. Materials with constitutive relations in which $\mathbf{S}$ is uniquely determined by $\mathbf{E}$ while satisfying the invariance property given by Equation 2.32 are known as Cauchy elastic materials.

We define a strain energy density functional $\Phi(\mathbf{E}) \in \mathbb{R}$ and obtain the strain energy from its gradient,

$$\mathbf{S}(\mathbf{E}) = \frac{\partial\Phi}{\partial\mathbf{E}}. \tag{2.33}$$

The strain energy density functional can only depend upon invariants, which are scalar-valued functions γ satisfying

$$\gamma(\mathbf{E}) = \gamma\left(Q\mathbf{E}Q^T\right) \tag{2.34}$$

for all orthogonal matrices Q.

We will assume without loss of generality that $\mathbf{E}$ is diagonal and take its set of eigenvalues as the invariants. It is immediately clear that there can be only three invariants and that there are many alternate choices, such as $\mathrm{trace}(\mathbf{E}), \mathrm{trace}(\mathbf{E}^2), |\mathbf{E}|$, and combinations thereof. It is common in the literature for invariants to be taken from $\mathbf{C} = \mathbf{I}_3 + 2\mathbf{E}$ instead of $\mathbf{E}$.

We use the compressible Neo-Hookean model,

$$\begin{aligned}\Phi(\mathbf{E}) &= \frac{\lambda}{2}(\log J)^2 + \frac{\mu}{2}(\mathrm{trace}\,\mathbf{C} - 3) - \mu\log J \\ &= \frac{\lambda}{2}(\log J)^2 + \mu\,\mathrm{trace}\,\mathbf{E} - \mu\log J,\end{aligned} \tag{2.35}$$

where $J = |\mathbf{F}| = \sqrt{|\mathbf{C}|}$ is the determinant of deformation, volumetric change. λ and μ are the Lamé parameters in the infinitesimal strain limit, given by

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \quad \mu = \frac{E}{2(1+\nu)}, \tag{2.36}$$

where E is the Young's modulus and ν is the Poisson's ratio for the materiel.

To evaluate Equation 2.33, we make use of

$$\frac{\partial J}{\partial \mathbf{E}} = \frac{\partial \sqrt{|\mathbf{C}|}}{\partial \mathbf{E}} = |\mathbf{C}|^{-1/2}|\mathbf{C}|\mathbf{C}^{-1} = J\mathbf{C}^{-1}, \tag{2.37}$$

where the factor of $\frac{1}{2}$ has been absorbed due to the fact that$\mathbf{C} = \mathbf{I}_3 + 2\mathbf{E}$. Carrying through the differentiation in Equation 2.33 for the model in Equation 2.35, we arrive at

$$\mathbf{S} = \lambda \log J\mathbf{C}^{-1} + \mu(\mathbf{I}_3 - \mathbf{C}^{-1}) \tag{2.38}$$

which is the strain energy for Neo-Hookean hyperelasticity at finite strain.

2.5.3.2 Hyperelasticity Weak Form

We multiply Equation 2.28 by a test function $\mathbf{v}$ and integrate by parts to obtain the weak form for finite-strain hyperelasticity: find $\mathbf{u} \in \mathcal{V} \subset H^1(\Omega_0)$ such that

$$\int_{\Omega_0} \nabla_X \mathbf{v} : \mathbf{P}\, dV - \int_{\Omega_0} \mathbf{v} \cdot \rho_0 \mathbf{g}\, dV - \int_{\partial\Omega_0} \mathbf{v} \cdot (\mathbf{P} \cdot \hat{\mathbf{N}})\, dS = 0, \quad \forall \mathbf{v} \in \mathcal{V}, \tag{2.39}$$

where $\mathbf{P} \cdot \hat{\mathbf{N}}|_{\partial\Omega}$ is replaced by any prescribed force/traction boundary condition written in terms of the initial configuration.

This equation contains material and constitutive nonlinearities in defining $\mathbf{S}(\mathbf{E})$, as well as geometric nonlinearities through $\mathbf{P} = \mathbf{F}\,\mathbf{S}$, $\mathbf{E}(\mathbf{F})$, and the body force $\mathbf{g}$, which must be pulled back from the current configuration to the initial configuration. Discretization of Equation 2.39 produces a finite-dimensional system of nonlinear algebraic equations, which we solve using Newton-Raphson methods. One attractive feature of Galerkin discretization is that we can arrive at the same linear system by discretizing the Newton linearization of the continuous form; that is, discretization and differentiation (Newton linearization) commute.

2.5.3.3 Newton Linearization

To derive a Newton linearization of Equation 2.39, we begin by expressing the derivative of Equation 2.29 in incremental form,

$$\mathrm{d}\mathbf{P} = \frac{\partial \mathbf{P}}{\partial \mathbf{F}} : \mathrm{d}\mathbf{F} = \mathrm{d}\mathbf{F}\,\mathbf{S} + \mathbf{F}\,\underbrace{\frac{\partial \mathbf{S}}{\partial \mathbf{E}} : \mathrm{d}\mathbf{E}}_{\mathrm{d}\mathbf{S}} \tag{2.40}$$

where

$$\mathrm{d}\mathbf{E} = \frac{\partial \mathbf{E}}{\partial \mathbf{F}} : \mathrm{d}\mathbf{F} = \frac{1}{2}\left(\mathrm{d}\mathbf{F}^T\mathbf{F} + \mathbf{F}^T\mathrm{d}\mathbf{F}\right) \tag{2.41}$$

and $\mathrm{d}\mathbf{F} = \nabla_X \mathrm{d}\mathbf{u}$. The quantity $\partial\mathbf{S}/\partial\mathbf{E}$ is known as the incremental elasticity tensor. We now evaluate $\mathrm{d}\mathbf{S}$ for the Neo-Hookean model given in Equation 2.38,

$$\mathrm{d}\mathbf{S} = \frac{\partial \mathbf{S}}{\partial \mathbf{E}} : \mathrm{d}\mathbf{E} = \lambda\left(\mathbf{C}^{-1} : \mathrm{d}\mathbf{E}\right)\mathbf{C}^{-1} + 2\left(\mu - \lambda \log J\right)\mathbf{C}^{-1}\mathrm{d}\mathbf{E}\,\mathbf{C}^{-1}, \tag{2.42}$$

where we have used

$$\mathrm{d}\mathbf{C}^{-1} = \frac{\partial \mathbf{C}^{-1}}{\partial \mathbf{E}} : \mathrm{d}\mathbf{E} = -2\mathbf{C}^{-1}\mathrm{d}\mathbf{E}\,\mathbf{C}^{-1}. \tag{2.43}$$

2.5.3.4 St. Venant-Kirchoff

One can linearize Equation 2.38 around $\mathbf{E} = 0$, for which $\mathbf{C} = \mathbf{I}_3 + 2\mathbf{E} \to \mathbf{I}_3$ and $J \to 1 + \operatorname{trace}\mathbf{E}$, therefore Equation 2.38 reduces to

$$\mathbf{S} = \lambda(\operatorname{trace}\mathbf{E})\mathbf{I}_3 + 2\mu\mathbf{E}, \tag{2.44}$$

which is the St. Venant-Kirchoff model. This model has constitutive linearization without geometric linearization, as mentioned in Figure 2.13.

This model can be used for geometrically nonlinear mechanics such as snap-through of thin structures, but it is inappropriate for large strain.

2.5.3.5 Hyperelasticity at Small Strain

Alternatively, one can drop the geometric nonlinearities, $\mathbf{E} \to \boldsymbol{\epsilon}$ and $\mathbf{C} \to \mathbf{I}_3$, while retaining the nonlinear dependence on $J \to 1 + \operatorname{trace}\boldsymbol{\epsilon}$, thereby yielding the Neo-Hookean hyperelasticity at small strain.

In this case, the strain energy density function is given by

$$\Phi(\boldsymbol{\epsilon}) = \lambda(1 + \operatorname{trace}\boldsymbol{\epsilon})(\log(1 + \operatorname{trace}\boldsymbol{\epsilon}) - 1) + \mu\boldsymbol{\epsilon} : \boldsymbol{\epsilon} \tag{2.45}$$

and the corresponding constitutive law is given by

$$\boldsymbol{\sigma} = \lambda \log(1 + \operatorname{trace}\boldsymbol{\epsilon})\mathbf{I}_3 + 2\mu\boldsymbol{\epsilon}. \tag{2.46}$$

2.5.3.6 Linear Elasticity

The linear elasticity model can be derived by linearizing both the geometric and constitutive nonlinearities in the finite strain model, as shown in Figure 2.13, or independently derived from the static balance of linear momentum.

The strong form of the static balance of linear momentum at small strain for the three dimensional linear elasticity problem is given by [43] as

$$\nabla \cdot \boldsymbol{\sigma} + \boldsymbol{g} = \mathbf{0} \tag{2.47}$$

where $\boldsymbol{\sigma}$ is the stress function and $\boldsymbol{g}$ is the forcing function. This strong form has the corresponding weak form

$$\int_\Omega \nabla\mathbf{v} : \boldsymbol{\sigma} dV - \int_{\partial\Omega} \mathbf{v} \cdot (\boldsymbol{\sigma} \cdot \hat{\mathbf{n}})\, dS - \int_\Omega \mathbf{v} \cdot \mathbf{g} dV = 0, \forall \mathbf{v} \in \mathcal{V} \tag{2.48}$$

for some displacement $\mathbf{u} \in \mathcal{V} \subset H^1(\Omega)$, where : denotes contraction over both components and dimensions.

In the linear elasticity constitutive model, the symmetric strain tensor is given by

$$\boldsymbol{\epsilon} = \frac{1}{2}\left(\nabla\mathbf{u} + \nabla\mathbf{u}^T\right) \tag{2.49}$$

and the linear elasticity constitutive law is given by $\boldsymbol{\sigma} = \mathsf{C} : \boldsymbol{\epsilon}$ where

$$\mathsf{C} = \begin{bmatrix} \lambda + 2\mu & \lambda & \lambda & & & \\ \lambda & \lambda + 2\mu & \lambda & & & \\ \lambda & \lambda & \lambda + 2\mu & & & \\ & & & \mu & & \\ & & & & \mu & \\ & & & & & \mu \end{bmatrix}. \tag{2.50}$$

2.5.3.7 Ongoing Research

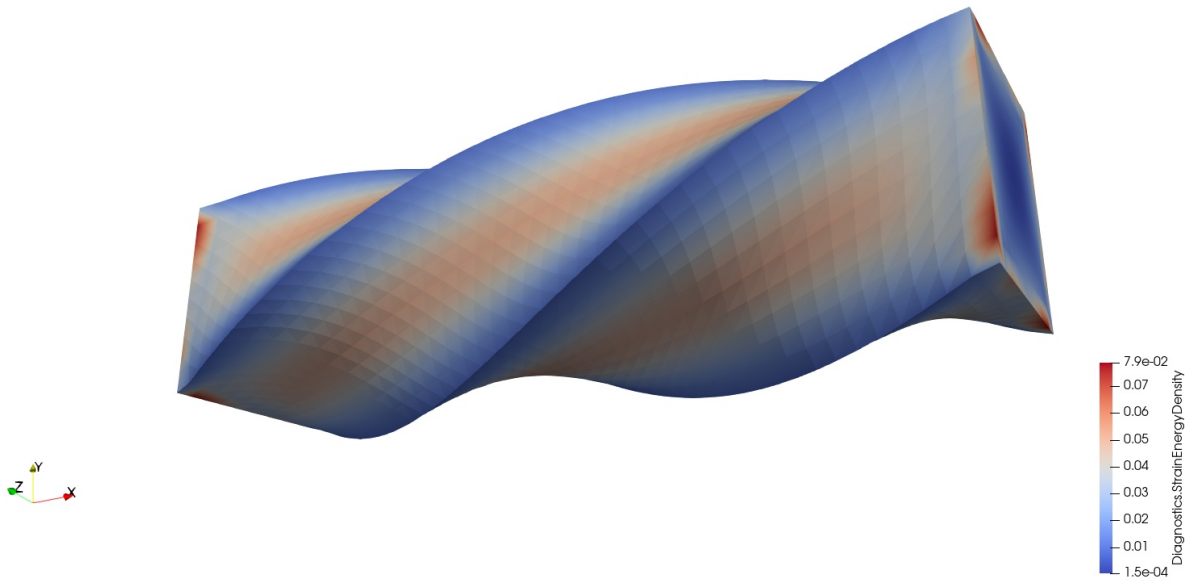


Figure 2.14: Strain Energy Density in Twisted Neo-Hookean Beam

Figure 2.14 shows the strain energy density for a beam undergoing a twist with Neo-Hookean hyperelastic modeling at finite strain. This mini-application can run on host or device processors. Areas of ongoing research include mixed finite element formulations for the displacement and pressure spaces, efficient load continuation techniques, and preconditioning improvements. For further information about the libCEED solid mechanics mini-application, see [4] and [64]; we summarize the key findings below.

2.5.3.8 Nearly Incompressible Linear Elasticity

In this section, we provide the results of performance studies for the linear elasticity formulation on a three dimensional unit cube using a manufactured solution for a range of Poisson's ratios that approach the incompressible limit of $\nu = 0.5$. The manufactured solution is reproducing a final displacement given by

$$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} e^{2x} \sin(3y) \cos(4z) \\ e^{3x} \sin(4y) \cos(2z) \\ e^{4x} \sin(2y) \cos(3z) \end{pmatrix} \tag{2.51}$$

with full Dirichlet boundary conditions on the outside of the box.

The performance study was conducted on a two-socket AMD EPYC 7452 machine. Each socket contains 32 CPU cores with a base clock speed of 2.35 GHz and 128 MB of L3 cache. The NPS4 BIOS configuration was used and processes were bound to cores. MPICH-3.3.2 was used with PETSc [6] version 3.14 and libCEED [12] version 0.7.

In Figure 2.15 and Figure 2.16 we investigate Pareto optimal configurations. A point is Pareto optimal if we cannot decrease error, along the y axis, without increasing cost in terms time, along the x axis. Note that as we approach the incompressible limit, we omit linear elements to avoid locking.

In Figure 2.15, the Pareto optimal configurations are toward the lower left of each pane, with $p = 3$ and $p = 4$ giving the fastest solutions for any error tolerance, while low-order elements ($p = 1$ and $p = 2$) are increasingly further from the Pareto front for larger values of ν. Each horizontal series of the same color represents strong scaling of a given resolution h and p.

In Figure 2.16, the Pareto optimal configurations are toward the lower left of each pane indicate higher order p is most cost-efficient. Each horizontal series of the same color represents a strong scaling study at fixed h and p, with perfect strong scaling manifesting when all the dots are collocated. The more expensive models tend to exhibit better strong scaling because they have more work over which to amortize the inherent communication costs, while small models are much more cost-efficient to run on a single core.

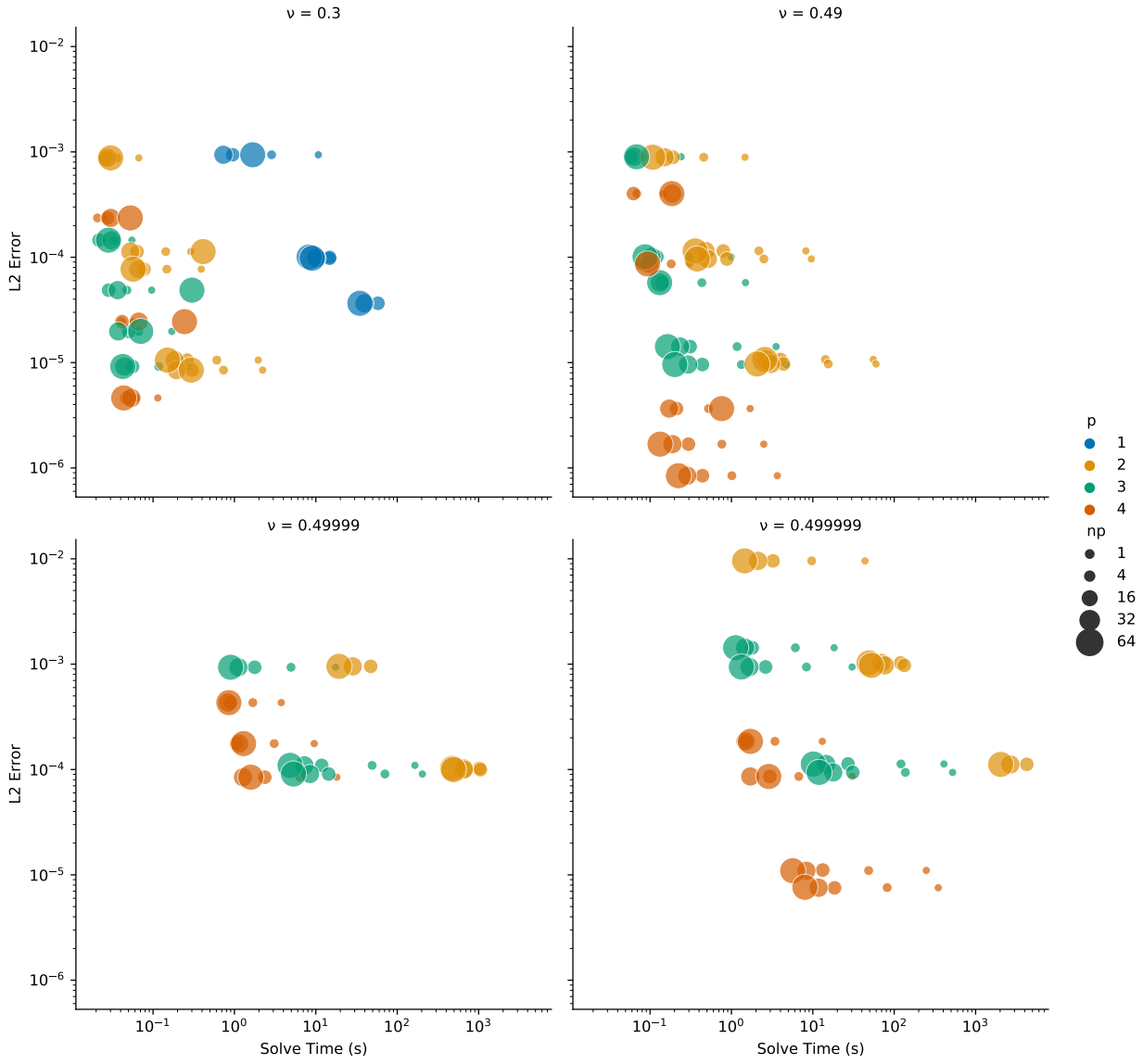


Figure 2.15: Error vs Time for $\nu = 0.3$, $\nu = 0.49$, $\nu = 0.49999$ and $\nu = 0.499999$ for Linear Elasticity

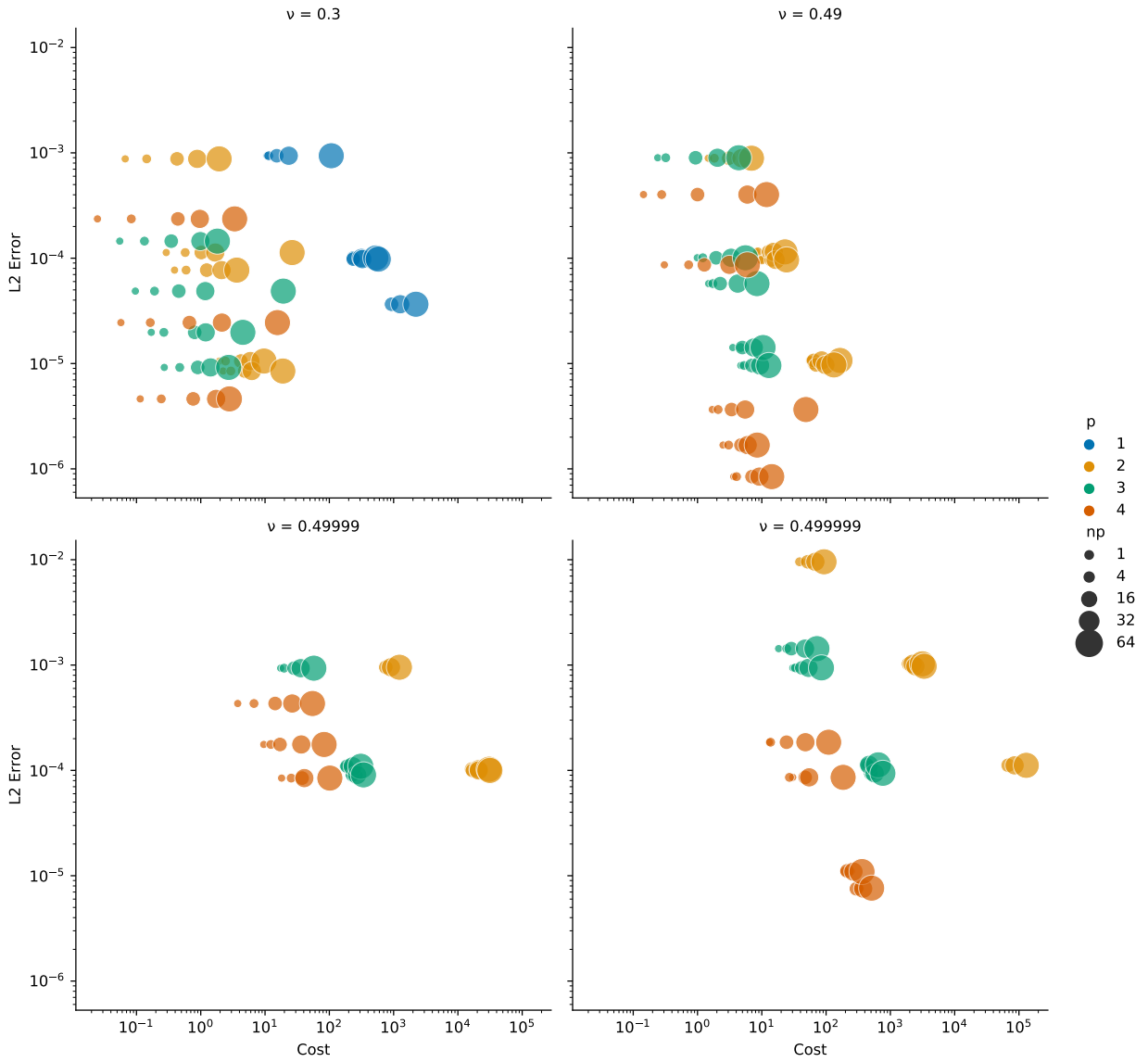


Figure 2.16: Error vs Cost for $\nu = 0.3$, $\nu = 0.49$, $\nu = 0.49999$ and $\nu = 0.499999$ for Linear Elasticity

In addition, we conducted an h-refinement study for polynomials of order 1 through 4 for the compressible case with $\nu = 0.3$ to determine the global rate of convergence of our implementation as the grid size is decreased. Figure 2.17 is a `log-log` plot of h versus L^2 error that represents the convergence of our implementation using grid sizes $h = 1/3$ to $h = 1/80$ for $\nu = 0.3$.

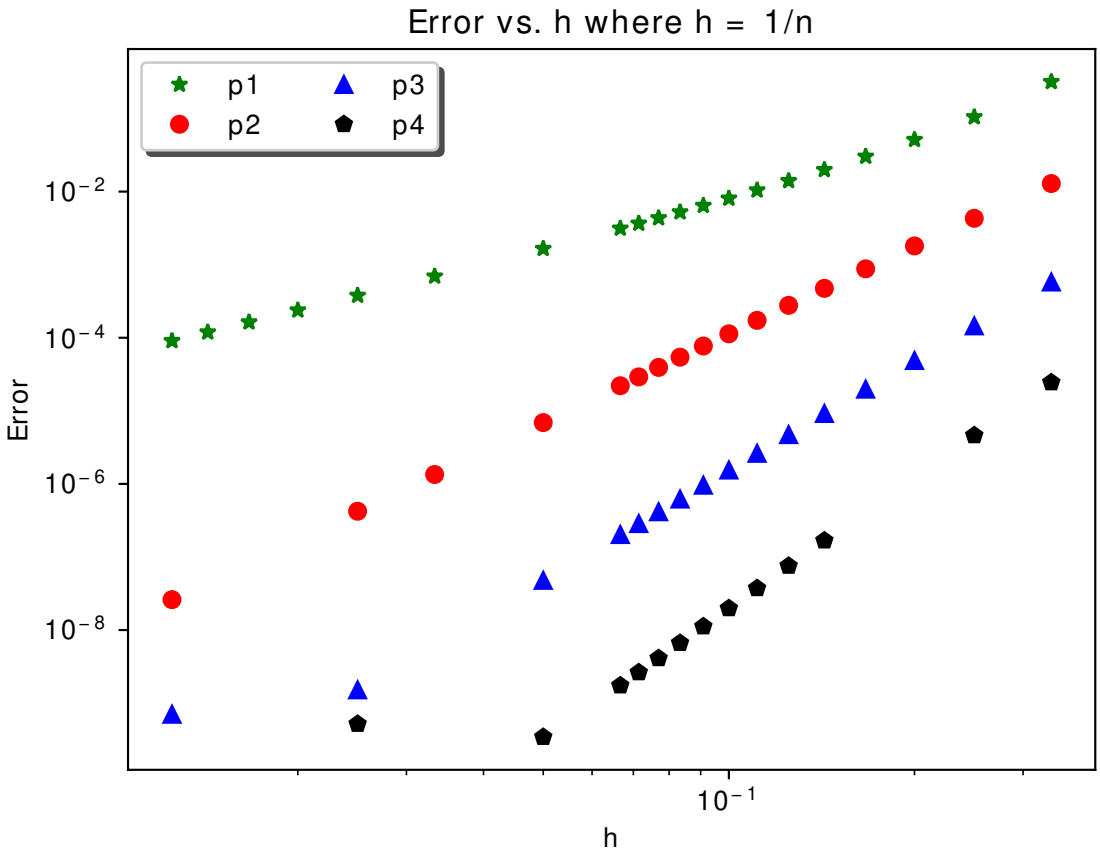


Figure 2.17: `log-log` Plot of L^2 Error vs h for Polynomial Orders 1-4 with $\nu = 0.3$ for Hyperelasticity at Finite Strain

In Figure 2.17, for each polynomial order, we calculate the slope of the line that is the best fit in the least square sense. The slopes of the lines of best fit in the least square sense in Figure 2.17 are 2.07, 4.00, 4.45, 4.75 for polynomials of order 1 through 4 respectively. We notice stagnation of the solver for $h = 1/40$ and $h = 1/80$ respectively with $p = 3$ and $p = 4$ due to the default tolerances that we used for the iterative solver. For polynomial of order 1, we notice accelerated

convergence behavior for coarse meshes, therefore they are not considered in the computation of the slope. We observed the expected spectral convergence; fixing the grid size constant while increasing the polynomial order provides the fastest time to converge to the desired solution given a target error tolerance.

2.5.3.9 Neo-Hookean Hyperelasticity at Finite Strain

In this section, we provide results for a performance study with Neo-Hookean hyperelasticity at finite strain.

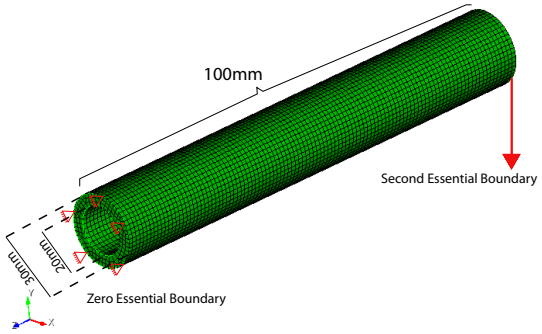


Figure 2.18: Hexahedral Mesh for Cylindrical Tube Bending Problem

For this test case, we simulate an assistive finger device shown in Figure 2.18. The length of the tube is 100mm, with circular cross-section with inner diameter 10mm and outer diameter 15mm. On the left end of the device we apply a zero Dirichlet boundary condition and on the right we apply a non-homogeneous Dirichlet boundary condition of 50mm in the negative y direction, downward, simulating a tube bending problem. We use a Poisson's ration of $\nu = 0.49$ and Young's Modulus of $E = 1.0$ MPa.

The performance study was conducted on a machine with a single AMD RYZEN 7 3700X 8-core 3.6GHz (4.4 GHz Max Boost) Socket AM4 65W 100-1000000711BOX Desktop Processor with G.SKILL Ripjaws V Series 32GB (2 x 16GB) 288-Pin DDR4 SDRAM DDR4 3200 (PC4 25600) Desktop Memory Model F4-3200C16D-32GV. PETSc [6] version 3.13 was used with libCEED [12]

version 0.6.

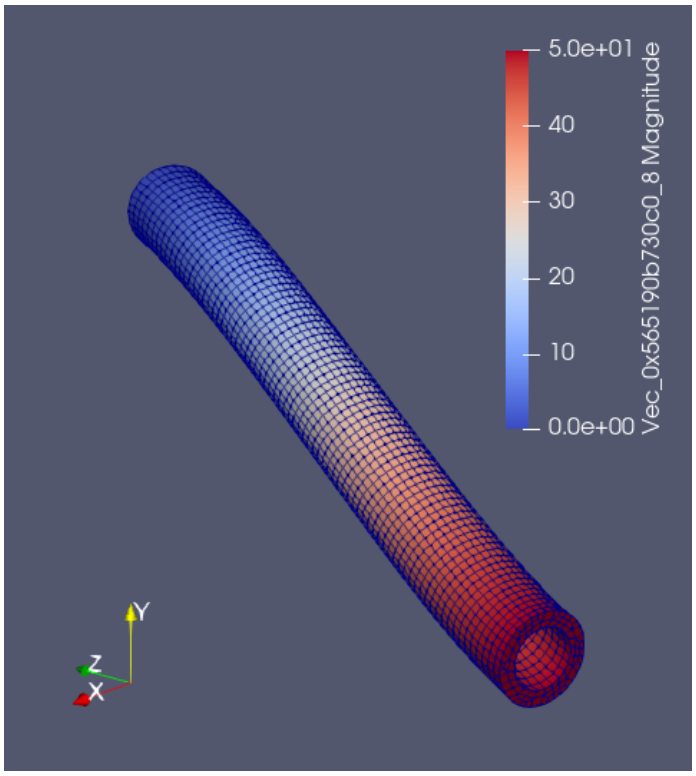


Figure 2.19: Displacement Magnitude in mm for Deformed Mesh

Figure 2.19 shows the final displacement of the tube in the simulation.

Elem	Nodes	dof	Load Increments	Time	np
736	7,434	22,302	500	474.05	8
5,550	52,920	158,760	750	8,749.68	8

Table 2.3: Total CPU Time with $\nu = 0.49$ for Hyperelasticity at Finite Strain

In Table 2.3, we see the the total CPU item required to complete the simulation. This simulation is currently performing poorly in the nearly incompressible regime, due to the large number of load increments required. Improved load increment strategies are an area of ongoing research for this mini-application.

Chapter 3

Local Fourier Analysis

Local Fourier Analysis (LFA) provides a tool to predict the convergence of preconditioning techniques for finite element and finite difference methods. LFA [7] was originally developed in the context of analyzing h-multigrid methods for finite difference methods, but since then LFA has been used to analyze finite element methods and a variety of preconditioning techniques. Our development of LFA of p-multigrid and Balancing Domain Decomposition by Constraints for single high-order finite element subdomains for general finite element operators is a novel addition to the field.

In this chapter, we introduce the notation required to describe LFA of high-order finite elements. In Section 3.1 we develop the notation for LFA of high-order finite element methods using the matrix-free notation from Chapter 2. This notation is used to develop LFA of smoothers for high-order finite element operators in Section 3.2. This notation will be used in Chapter 4 and Chapter 5 to analyze the performance of matrix-free implementations of Multigrid and Balancing Domain Decomposition by Constraints preconditioners for high-order finite element methods.

3.1 Local Fourier Analysis of High-Order Operators

Consider a scalar Toeplitz operator L_h on an infinite one dimensional uniform grid G_h,

$$\begin{aligned} L_h &\hat{=} [s_\kappa]_h \, (\kappa \in V) \\ L_h w_h(x) &= \sum_{\kappa \in V} s_\kappa w_h(x + \kappa h) \end{aligned} \tag{3.1}$$

where $V \subset \mathbb{Z}$ is a finite index set, $s_\kappa \in \mathbb{R}$ are constant coefficients, and $w_h(x)$ is a l^2 function on G_h. In terms of stencils, s_k are stencil weights that are nonzero on the neighborhood $\kappa \in V$.

Since L_h is Toeplitz, it can be diagonalized by the standard Fourier modes $\varphi(\theta, x) = e^{\imath\theta x/h}$. These Fourier modes alias according to $\varphi(\theta + 2\pi/h, x)$ on all grid points $x \in h\mathbb{Z}$.

Definition 3.1.1 (Symbol of L_h) If for all grid functions $\varphi(\theta, x)$ we have

$$L_h\varphi(\theta, x) = \tilde{L}_h(\theta)\varphi(\theta, x) \tag{3.2}$$

then we define $\tilde{L}_h(\theta) = \sum_{\kappa\in V} s_\kappa e^{\imath\theta\kappa}$ as the symbol of L_h, where $\imath^2 = -1$.

This definition can be extended to a $q \times q$ linear system of operators by

$$\mathbf{L}_h = \begin{bmatrix} L_h^{1,1} & \cdots & L_h^{1,q} \\ \vdots & \vdots & \vdots \\ L_h^{q,1} & \cdots & L_h^{q,q} \end{bmatrix} \tag{3.3}$$

where $L_h^{i,j}$, $i, j \in \{1, 2, \ldots, q\}$ is given by scalar Toeplitz operators describing how component j appears in the equation for component i. The symbol of $\mathbf{L}_h$, denoted $\tilde{\mathbf{L}}_h$, is a $q \times q$ matrix valued function of θ given by $\left[\tilde{\mathbf{L}}_h\right]_{i,j} = \tilde{L}_h^{i,j}(\theta)$. Note that for a system of equations representing an error propagation operator in a relaxation scheme, the spectral radius of the symbol matrix determines now rapidly the scheme decreases error at a target frequency.

These definitions are extended to d dimensions by taking the neighborhood $V \subset \mathbb{Z}^d$ and letting $\boldsymbol{\theta} \in [-\pi, \pi)^d$. For standard coarsening in the analysis of h-multigrid, low frequencies are given by $\boldsymbol{\theta} \in T^{\text{low}} = [-\pi/2, \pi/2)^d$ and high frequencies are given by $\boldsymbol{\theta} \in [-\pi, \pi)^d \setminus T^{\text{low}}$, or equivalently via periodicity, $\theta \in T^{\text{high}} = [-\pi/2, 3\pi/2)^d \setminus T^{\text{low}}$.

We can compute the symbol of a $p \times p$ linear system of operators representing a discretized PDE. We start with a system of equations representing a Galerkin operator, such as in Equation 2.6, but we omit boundary terms in this derivation, as they are not present on the infinite uniform grid G_h.

Using the algebraic representation of PDE operators given in Chapter 2, the PDE operator $\mathbf{A}$ is of the form

$$\begin{aligned} \mathbf{A} &= \mathbf{P}^T \mathbf{G}^T \mathbf{A}^e \mathbf{G} \mathbf{P} \\ \mathbf{A}^e &= \mathbf{B}^T \mathbf{D} \mathbf{B} \end{aligned} \tag{3.4}$$

where $\mathbf{P}$ and $\mathbf{G}$ represent the element assembly operators, $\mathbf{B}$ is a basis operator which computes the values and derivatives of the basis functions at the quadrature points, and $\mathbf{D}$ is a block diagonal operator which provides the pointwise application of the bilinear form on the quadrature points, to include quadrature weights and the change in coordinates between the physical and reference space.

We focus on LFA of the element operator, $\mathbf{A}^e$, on the mesh grid with points given by x_i, for $i \in \{1, 2, \ldots, p+1\}$, where p is the polynomial degree of the basis. We first develop the LFA in one dimension for scalar operators and then extend this analysis to more general operators.

In one dimension, the nodes on the left and right boundaries of the element map to the same Fourier mode when localized to nodes unique to a single finite element, so we can compute the symbol matrix as

$$\tilde{\mathbf{A}} = \mathbf{Q}^T \left(\mathbf{A}^e \odot \left[e^{\imath (x_j - x_i)\theta/h} \right] \right) \mathbf{Q} \tag{3.5}$$

where $\odot$ represents pointwise multiplication of the elements, h is the length of the element, and $i, j \in \{1, 2, \ldots, p+1\}$. In the pointwise product $\mathbf{A}^e \odot \left[e^{\imath (x_j - x_i)\theta/h} \right]$, the (i, j) entry is given by $[\mathbf{A}^e]_{i,j} \, e^{\imath (x_j - x_i)\theta/h}$. $\mathbf{Q}$ is a $(p+1) \times p$ matrix that localizes Fourier modes to each element, given by

$$\mathbf{Q} = \begin{bmatrix} \mathbf{I} \\ \mathbf{e}_0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 \\ 1 & 0 & \cdots & 0 \end{bmatrix}. \tag{3.6}$$

The computation of this symbol matrix extends to more complex PDE with multiple components and in higher dimensions.

Multiple components are supported by extending the $p \times p$ system of Toeplitz operators in Equation 3.5 to a $(n \cdot p) \times (n \cdot p)$ system of operators, where n is the number of components in the PDE. The localization operator $\mathbf{Q}$ for a multi-component PDE is given by $\mathbf{Q}_n = \mathbf{I}_n \otimes \mathbf{Q}$. In general, we omit the subscript indicating the number of components for the localization operator.

The infinite uniform grid G_h is extended into higher dimensions by taking the direct sum of the one dimensional grid. In a similar fashion to how tensor products are used to extend the one dimensional bases into higher dimensions, the localization of Fourier modes in two dimensions is given by

$$\mathbf{Q}_{2\text{d}} = \mathbf{Q} \otimes \mathbf{Q} \tag{3.7}$$

and the localization in three dimensions is given by

$$\mathbf{Q}_{3\text{d}} = \mathbf{Q} \otimes \mathbf{Q} \otimes \mathbf{Q}. \tag{3.8}$$

Again, we generally omit the subscript indicating the dimension of the Fourier mode localization operator.

Definition 3.1.2 (Symbol of High-Order Finite Element Operators) The symbol matrix of a finite element operator for an arbitrary second-order PDE with any number of components, basis order, and dimension is given by

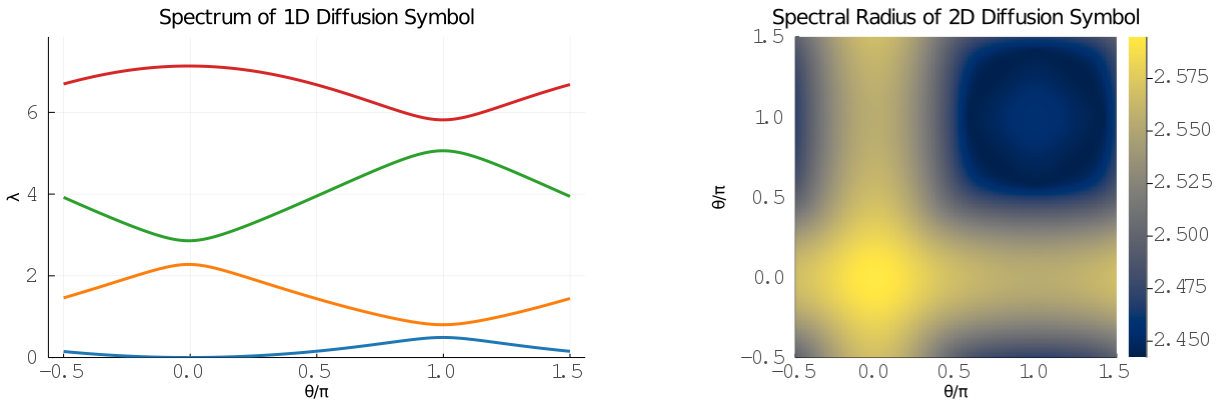
$$\tilde{\mathbf{A}}(\boldsymbol{\theta}) = \mathbf{Q}^T \left(\mathbf{A}^e \odot \left[e^{\imath(\mathbf{x}_j - \mathbf{x}_i)\cdot \boldsymbol{\theta}/\mathbf{h}} \right] \right) \mathbf{Q} \tag{3.9}$$

where $\odot$ represents pointwise multiplication of the elements, $\mathbf{h}$ is the length of the element in each dimension, $\boldsymbol{\theta}$ is the target frequency in each dimension, $i, j \in \left\{1, 2, \ldots, n \cdot (p+1)^d\right\}$, n is the number of components, p is the polynomial degree of the discretization, and d is the dimension of the finite element basis. $\mathbf{A}^e$ is the finite element operator for the element and $\mathbf{Q}$ is the localization operator for Fourier modes on an element.

Note that this LFA framework is applicable to any second-order PDE with a weak form that can be represented by the matrix-free representation given by Equation 2.5 or Equation 2.6. This

representation is used in LFAToolkit.jl, where the users provide the finite element basis **B**, the Fourier mode localization operator **Q**, and the pointwise representation of the weak form **D**, and the software provides the LFA of the PDE operator and various preconditioners.

As we develop LFA of various preconditioners and smoothers in this framework throughout this chapter and the following chapters, we will use the scalar diffusion operator as our standard test case to illustrate the properties these techniques with this analysis.



(a) Spectrum of Scalar Diffusion for $p = 4$

(b) Spectrum of Scalar Diffusion for $p = 4$

Figure 3.1: Spectrum of Scalar Diffusion Operator Symbol

We see the spectrum of the one dimensional scalar diffusion operator in Figure 3.1a and the spectral radius of the symbol of the two dimensional scalar diffusion operator in Figure 3.1b. In both cases, we used a fourth-order H^1 Lagrange finite element basis on the Gauss-Lobatto points. These plots were generated with LFAToolkit.jl. Various preconditioning techniques will reduce the spectral radius of this symbol, each with different effectiveness in different frequency ranges.

3.2 Local Fourier Analysis of Polynomial Smoothers

In this section we investigate LFA of two preconditioners often used as smoothers for multigrid methods. These smoothers can also be used as preconditioners for iterative solvers independently from multigrid methods.

The error propagation operator for a preconditioner is given by $\mathbf{S} = I - \mathbf{M}^{-1}\mathbf{A}$, where $\mathbf{M}^{-1}$

is determined by the particular preconditioner under investigation. We will define the symbol of the error propagation operator for weighted Jacobi preconditioner and the Jacobi preconditioned Chebyshev semi-iterative method.

The LFA-predicted convergence factor is given by the maximum spectral radius of the symbol of the error propagation operator across all frequencies

$$\mu = \max_{\boldsymbol{\theta} \in T^{\text{low}}, T^{\text{high}}} \rho\left(\tilde{\mathbf{S}}\left(\nu, \omega, \boldsymbol{\theta}\right)\right), \tag{3.10}$$

where $\rho\left(\tilde{\mathbf{S}}\left(\nu, \omega, \boldsymbol{\theta}\right)\right)$ denotes the spectral radius of the matrix symbol $\tilde{\mathbf{S}}$.

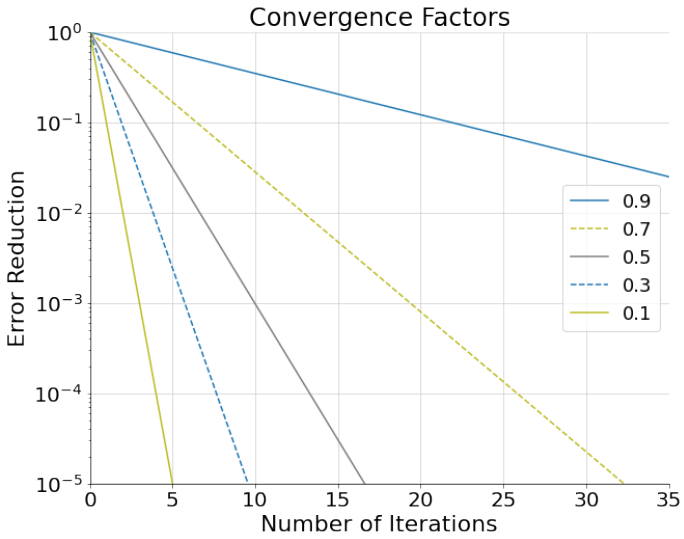


Figure 3.2: Richardson Iterations Required to Target Error Tolerances

Tolerance	$\rho = 0.9$	$\rho = 0.7$	$\rho = 0.5$	$\rho = 0.3$	$\rho = 0.1$
10^{-1}	22	7	4	2	1
10^{-5}	110	33	17	10	5
10^{-10}	219	65	34	20	10
10^{-15}	328	97	50	29	15

Table 3.1: Richardson Iterations Required to Target Error Tolerances

If we consider a Richardson iteration as an iterative solver for a system of equations, then the convergence factor for a preconditioner or smoother determines how many iterations would be

required for the Richardson iteration to reach a target reduction in error. Table 3.1 shows the number of iterations required to achieve a range of target error tolerances for different convergence factors. Figure 3.2 shows this same information in a visual format to emphasize the importance of reducing the convergence factor for these methods.

Note that Krylov subspace methods, such as the Conjugate Gradient (CG) method, converge faster than Richardson iterations; however, this convergence factor still provides valuable information about the effectiveness of these preconditioners. A smaller convergence factor indicates that the preconditioned operator is closer to the identity operator, which indicates that CG iterations will converge much faster.

3.2.1 Jacobi

In weighted Jacobi, the preconditioning matrix is given by $\mathbf{M}^{-1} = \omega \left(\operatorname{diag} \mathbf{A}\right)^{-1}$, where ω is the weighting factor. Following the derivation from Section 3.1, the symbol of the weighted Jacobi error propagation operator is therefore given by

$$\tilde{\mathbf{S}}\left(\omega, \boldsymbol{\theta}\right) = \mathbf{I} - \tilde{\mathbf{M}}^{-1}\left(\omega, \boldsymbol{\theta}\right) \tilde{\mathbf{A}}\left(\boldsymbol{\theta}\right) = \mathbf{I} - \omega \left(\mathbf{Q}^T \left(\operatorname{diag} \mathbf{A}^e\right)^{-1} \mathbf{Q}\right) \tilde{\mathbf{A}}\left(\boldsymbol{\theta}\right), \tag{3.11}$$

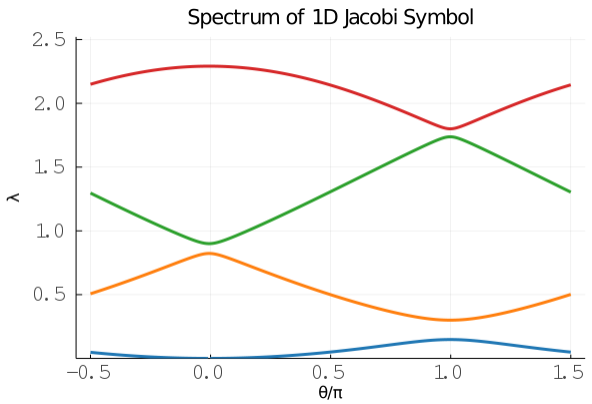
where this expression has been simplified by the fact that $e^{\imath(x_i - x_i)\theta/h} = 1$.

Definition 3.2.1 (Symbol of Jacobi Preconditioner Error Operator) The symbol of the error propagation operator for weighted Jacobi smoothing is given by

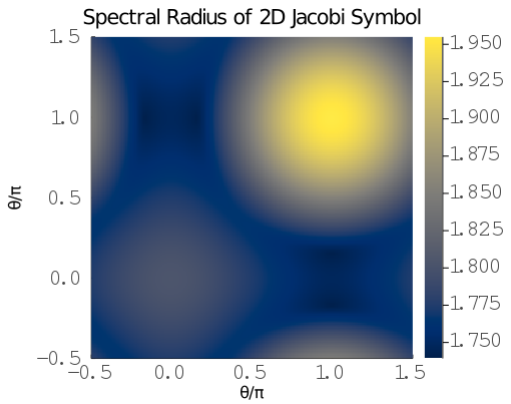
$$\tilde{\mathbf{S}}\left(\nu, \omega, \boldsymbol{\theta}\right) = \left(\mathbf{I} - \omega \left(\mathbf{Q}^T \left(\operatorname{diag} \mathbf{A}^e\right) \mathbf{Q}\right)^{-1} \tilde{\mathbf{A}}\left(\boldsymbol{\theta}\right)\right)^{\nu}, \tag{3.12}$$

where ν is the number of smoothing passes and ω is the weighting parameter.

Using Definition 3.2.1, in Figure 3.3a we see the spectrum of the symbol of the one dimensional Jacobi preconditioned operator $\tilde{\mathbf{M}}^{-1}\tilde{\mathbf{A}}$ for the one dimensional scalar diffusion operator and in 3.3b we see the spectral radius of the symbol of the two dimensional Jacobi preconditioned operator for the scalar diffusion problem. In both cases we use a fourth-order H^1 Lagrange finite element basis



(a) Spectrum of 1D Jacobi for $p = 4$



(b) Spectral Radius of 2D Jacobi for $p = 4$

Figure 3.3: Spectrum of Jacobi Preconditioner Symbol

on the Gauss-Lobatto points. In Figure 3.3a, we see that the unweighted Jacobi error symbol still has a spectral radius greater than 1.0; this spectral radius can be improved by changing the weighting factor, ω.

As discussed in [38], the classical optimal smoothing parameter is given by

$$\omega = \frac{2}{\lambda_{\text{min,high}} + \lambda_{\text{max,high}}} \tag{3.13}$$

for weighted Jacobi. Unfortunately, as noted by [38], this optimal smoothing parameter does not result in optimal two-grid convergence. Additionally, this method of estimating smoother parameter values is incompatible with this LFA framework.

3.2.2 Chebyshev

Damped Jacobi is based on the degree 1 polynomial with the root at $\alpha = 1/\omega > 0$. For $\nu = 1$ smoothing passes, this polynomial is minimal on the interval $[\alpha - c, \alpha + c]$, for any $c > 0$. However, for $\nu \geq 2$ sequential applications of the preconditioner, such as in multiple smoothing passes for a multigrid cycle, the Jacobi smoother is a degree ν polynomial that is not minimal on this interval. Chebyshev smoothers are based on a stable recurrence relation for any polynomial order that is minimal for a target interval on the positive real axis, which should include all eigenvalues corresponding to Fourier modes that the smoother is responsible for reducing.

It is well known that polynomial smoothers allow more aggressive coarsening than Jacobi [9], although this will not prove sufficient for our aggressive coarsening directly to linear elements. For further discussion of the error propagation properties of the Chebyshev semi-iterative method, see [37].

We use the Jacobi preconditioned operator, $(\operatorname{diag} \mathbf{A})^{-1} \mathbf{A}$, in this Chebyshev iteration instead of the finite element operator $\mathbf{A}$, similar to the discussion in [2].

The terms in the Chebyshev semi-iterative method can be modeled by the three term recurrence relation given by

$$\mathbf{u}_k = -\left(\mathbf{r}_{k-1} + \alpha \mathbf{u}_{k-1} + \beta_{k-2} \mathbf{u}_{k-2}\right) / \gamma_{k-1} \tag{3.14}$$

where the spectrum of $(\operatorname{diag} \mathbf{A})^{-1} \mathbf{A}$ lies on the line segment $[\alpha - c, \alpha + c]$ and the parameters β and γ are given by the recurrence

$$\begin{array}{cc} \beta_0 = -\frac{c^2}{2\alpha} & \gamma_0 = -\alpha \\ \beta_k = \frac{c}{2} \frac{T_k(\eta)}{T_{k+1}(\eta)} = \left(\frac{c}{2}\right)^2 \frac{1}{\gamma_k} & \gamma_k = \frac{c}{2} \frac{T_{k+1}(\eta)}{T_k(\eta)} = -\left(\alpha + \beta_{k-1}\right). \end{array} \tag{3.15}$$

In this equation, $T_i(\zeta) = 2\zeta T_{i-1}(\zeta) - T_{i-2}(\zeta)$ are the classical Chebyshev polynomials, which are evaluated at the point $\eta = -\alpha / c$.

The residual in the Chebyshev semi-iterative method can therefore be modeled by the three term recurrence

$$\mathbf{r}_k = \left((\operatorname{diag} \mathbf{A})^{-1} \mathbf{A} \mathbf{r}_{k-1} - \alpha \mathbf{r}_{k-1} - \beta_{k-2} \mathbf{r}_{k-2} \right) / \gamma_{k-1}. \tag{3.16}$$

Using the recurrence relation given in Equation 3.16, we can define the error propagation of the kth order Chebyshev smoother in terms of the error in the first term:

$$\begin{aligned} \mathbf{E}_0 &= \mathbf{I} \\ \mathbf{E}_1 &= \mathbf{I} - \tfrac{1}{\alpha} (\operatorname{diag} \mathbf{A})^{-1} \mathbf{A} \\ \mathbf{E}_k &= \left((\operatorname{diag} \mathbf{A})^{-1} \mathbf{A} \mathbf{E}_{k-1} - \alpha \mathbf{E}_{k-1} - \beta_{k-2} \mathbf{E}_{k-2} \right) / \gamma_{k-1} \end{aligned} \tag{3.17}$$

With this recursive definition of the error propagation operator, we can define the symbol for Chebyshev smoothing.

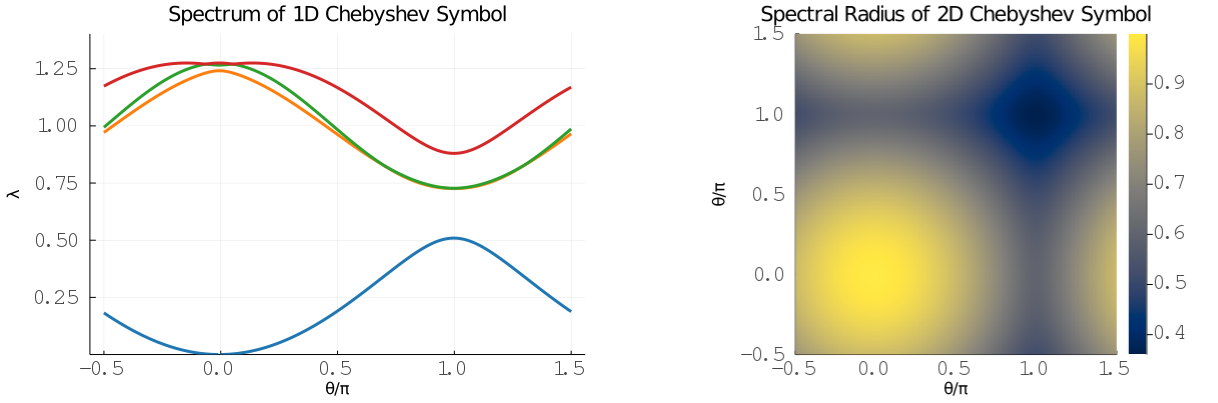
Definition 3.2.2 (Symbol of Chebyshev Preconditioner Error Operator) The symbol of the error propagation operator for kth order Chebyshev smoothing based on the Jacobi preconditioned operator is given by

$$\tilde{\mathbf{S}}(\nu, n, \boldsymbol{\theta}) = \left(\tilde{\mathbf{E}}_k\right)^{\nu}, \tag{3.18}$$

where ν is the number of smoothing passes and $\tilde{\mathbf{E}}_k(\boldsymbol{\theta})$ is given by the recursive definition

$$\begin{aligned} \tilde{\mathbf{E}}_0(\boldsymbol{\theta}) &= \mathbf{I} \\ \tilde{\mathbf{E}}_1(\boldsymbol{\theta}) &= \mathbf{I} - \tfrac{1}{\alpha}\tilde{\mathbf{A}}_J\tilde{\mathbf{A}}(\boldsymbol{\theta}) \\ \tilde{\mathbf{E}}_k(\boldsymbol{\theta}) &= \left(\tilde{\mathbf{A}}_J\tilde{\mathbf{A}}(\boldsymbol{\theta})\tilde{\mathbf{E}}_{k-1}(\boldsymbol{\theta}) - \alpha\tilde{\mathbf{E}}_{k-1}(\boldsymbol{\theta}) - \beta_{k-2}\tilde{\mathbf{E}}_{k-2}(\boldsymbol{\theta})\right)/\gamma_{k-1} \end{aligned} \tag{3.19}$$

with $\tilde{\mathbf{A}}_J = \left(\mathbf{Q}^T \operatorname{diag}(\mathbf{A}^e)\mathbf{Q}\right)^{-1}$ giving the symbol of the Jacobi smoother.



(a) Spectrum of 1D Chebyshev for $p = 4$

(b) Spectral Radius of 2D Chebyshev for $p = 4$

Figure 3.4: Spectrum of Chebyshev Preconditioner Symbol

Using Definition 3.2.2, in Figure 3.4a we see the spectrum of the symbol of third-order Chebyshev polynomial smoothing for the one dimensional scalar diffusion operator and in 3.4b we see the spectral radius of the symbol of third-order Chebyshev polynomial smoothing for the two dimensional scalar diffusion operator. Again, in both cases we use a fourth-order H^1 Lagrange finite element basis on the Gauss-Lobatto points. When compared to Figures 3.3a and 3.3b, we can see that smoothing based upon the Chebyshev semi-iterative method offers better error reduction than weighted or unweighted Jacobi, albeit at a higher computational cost.

Note that first-order Chebyshev smoothing is equivalent to Jacobi smoothing with the classical choice of parameter value of $\omega = 1/\alpha$.

The Chebyshev semi-iterative method is known to be sensitive to the quality of the estimates of the extremal eigenvalues $\lambda_{\min}$ and $\lambda_{\max}$. Large eigenvalues are associated with higher frequencies for differential operators, so $\lambda_{\min}$ for effective smoothing as a part of multigrid methods is not the minimal eigenvalue. Additionally, the true minimal eigenvalue is difficult to compute and goes to zero under grid refinement.

PETSc [6] estimates the eigenvalues of the preconditioned operator via Krylov iterations, yielding an estimate of the maximal eigenvalue, $\hat{\lambda}_{\max}$. This estimate is used to target the upper part of the spectrum of the error, with $\lambda_{\min} = 0.1\hat{\lambda}_{\max}$ and $\lambda_{\max} = 1.1\hat{\lambda}_{\max}$. This default lower bound was chosen based upon empirical experiments for rapid coarsening with smoothed aggregation algebraic multigrid (AMG), and the upper bound incorporates a safety factor due to the fact that the maximum eigenvalue estimate from the Krylov iterations tends to be an underestimate of the true maximum eigenvalue.

In LFAToolkit.jl, we also want to target the upper part of the spectrum of the error; we estimate the spectral radius of the symbol of the Jacobi preconditioned operator by sampling the the frequencies at a small number of values to compute $\hat{\lambda}_{\max}$. We then take $\lambda_{\min} = 0.1\hat{\lambda}_{\max}$ and $\lambda_{\max} = 1.0\hat{\lambda}_{\max}$. Adjusting this lower bound has different consequences when using conservative and aggressive coarsening strategies with p-multigrid, as we will investigate in the next chapter.

Chapter 4

Multigrid Methods

Multigrid methods are popular multi-level techniques that provide resolution independent convergence rates. p-type multigrid, developed by Ronquist and Patera [74], is a natural choice for high-order finite elements on an unstructured mesh and can be implemented with operators represented in the matrix-free form given in Equation 2.7. Ronquist and Patera declared p-multigrid *sensibly independent* of the number of elements and polynomial degree of the element bases.

In a recent publication, Davydow et al. use h-multigrid for matrix-free finite elements in solid mechanics; however p-multigrid can offer more flexibility with respect to meshes in comparison to h-multigrid as it does not require aggregation of multiple elements into larger elements, which can be difficult on more complex geometry. There has been work by Heys, Manteuffel, McCormick, and Olson demonstrating the feasibility of algebraic multigrid (AMG) for high-order finite elements [40]; however, AMG requires assembly of the finite element operator, which defeats the benefits of matrix-free implementations.

In this chapter we introduce LFA of p-multigrid and generalize this analysis to reproduce previous work on h-multigrid. We provide convergence factors for aggressive coarsening with Jacobi and Chebyshev semi-iterative smoothers as a baseline to compare with Balancing Domain Decomposition by Constraints smoothing.

4.1 Matrix-Free Multigrid Methods

Both h-multigrid and p-multigrid follow the same general algorithm, with differences in the grid transfer operators and coarse grid representation. Application of multigrid follows the algorithm given by Definition 4.1.1, similar to the presentation in [8].

Definition 4.1.1 (Multigrid Algorithm)

1: Compute $\mathbf{u}_k$

2: $\mathbf{u}_k \leftarrow \mathbf{u}_k + \hat{\mathbf{M}}^{-1}\left(\mathbf{b} - \mathbf{A}_f \mathbf{u}_k\right)$ ▷ pre-smooth ν times

3: $\mathbf{r} = \mathbf{R}_{\text{ftoc}}\left(\mathbf{b} - \mathbf{A}_f \mathbf{u}_k\right)$ ▷ restrict the residual

4: $\mathbf{A}_c \mathbf{e} = \mathbf{r}$ ▷ Solve on coarse grid (may involve additional levels)

5: $\mathbf{u}_k \leftarrow \mathbf{u}_k + \mathbf{P}_{\text{ctof}} \mathbf{e}$ ▷ prolong error correction

6: $\mathbf{u}_k \leftarrow \mathbf{u}_k + \hat{\mathbf{M}}^{-1}\left(\mathbf{b} - \mathbf{A}_f \mathbf{u}_k\right)$ ▷ post-smooth ν times

In this algorithm, $\mathbf{A}_f$ is the operator on the fine grid, $\mathbf{P}_{\text{ctof}}$ is the coarse to fine grid prolongation operator, $\mathbf{R}_{\text{ftoc}}$ the fine to coarse grid restriction operator, $\hat{\mathbf{M}}$ a separate preconditioner used for smoothing, and $\mathbf{A}_c$ represents solving the error correction problem on the coarse grid, which may involve recursively applying the multigrid algorithm.

For h-multigrid, the restriction operator represents aggregating multiple fine grid elements into larger coarse grid elements and the prolongation operator is given by the transpose. This process requires knowledge about neighboring elements on the mesh and can be complex on fully unstructured meshes and complex geometry.

For p-multigrid with nodal bases, the prolongation operator interpolates from the coarse grid to the fine grid, evaluating the lower-order basis functions on the higher-order basis nodes. Prolongation to a basis with a polynomial order that is a single order higher than the coarse grid basis is defined by

$$\mathbf{P}_{p-1}^{p} = \Lambda\left(m_p^{-1}\right) \mathcal{E}_p^T \sum_e \mathbf{N}_{p-1}^{p} \mathcal{E}_{p-1}^{e}, \tag{4.1}$$

where $\mathbf{N}_{p-1}^{p}$ interpolates from a basis of degree $p-1$ to degree p and $m_p = \mathcal{E}_p^T \mathcal{E}_p 1$ counts the multiplicity of shared nodes between elements. To preserve symmetry and prevent aliasing, the restriction operator is defined as the transpose of the prolongation operator, $\mathbf{R}_{p-1}^{p} = \left(\mathbf{P}_{p-1}^{p}\right)^T$. These operators can be implemented in a matrix-free fashion.

Even with modest order finite elements, such as degree 4, p-multigrid can substantially reduce the size of the global solution vector, by a factor of approximately $(p+1)^3/8$. Thus, assembly of the finite element operator on the coarse grid, $\mathbf{A}_c$, becomes tractable and direct solvers such as AMG can be used to solve the coarse problem.

4.2 Local Fourier Analysis of Multigrid Methods

With the representation of the symbol of high-order PDE operators given in Definition 3.1.2, we can derive the symbol of multigrid error propagation operators.

The total multigrid error propagation operator is given by

$$\mathbf{E}_{\text{2MG}} = \mathbf{S}_f \left(\mathbf{I} - \mathbf{P}_{\text{ctof}} \mathbf{A}_c^{-1} \mathbf{R}_{\text{ftoc}} \mathbf{A}_f\right) \mathbf{S}_f \tag{4.2}$$

where $\mathbf{A}_f$ represents the fine grid operator, $\mathbf{A}_c^{-1}$ represents the coarse grid solve, $\mathbf{S}_f$ represents the smoother error propagation operator, while $\mathbf{P}_{\text{ctof}}$ and $\mathbf{R}_{\text{ftoc}}$ represent the grid prolongation and restriction operators between the coarse and fine grids, respectively.

Definition 4.2.1 (Symbol of Multigrid Error Operator) The symbol of total multigrid error propagation operator for a finite element operator $\mathbf{A}$ is given by

$$\tilde{\mathbf{E}}_{\text{2MG}}(\boldsymbol{\theta}, \omega) = \tilde{\mathbf{S}}_f(\boldsymbol{\theta}, \omega) \left(\mathbf{I} - \tilde{\mathbf{P}}_{\text{ctof}}(\boldsymbol{\theta}) \left(\tilde{\mathbf{A}}_c(\boldsymbol{\theta})\right)^{-1} \tilde{\mathbf{R}}_{\text{ftoc}}(\boldsymbol{\theta}) \tilde{\mathbf{A}}_f(\boldsymbol{\theta})\right) \tilde{\mathbf{S}}_f(\boldsymbol{\theta}, \omega) \tag{4.3}$$

where $\tilde{\mathbf{A}}_f$ represents the fine grid operator symbol, $\tilde{\mathbf{A}}_c^{-1}$ represents the symbol of the coarse grid solve, $\tilde{\mathbf{S}}_f$ represents the smoother error propagation symbol with any additional parameters ω, while $\tilde{\mathbf{P}}_{\text{ctof}}$ and $\tilde{\mathbf{R}}_{\text{ftoc}}$ represent the symbols of the grid prolongation and restriction operators, respectively.

This error propagation operator can represent both h-multigrid and p-multigrid, depending upon the grid transfer operators and coarse grid representation chosen.

4.2.1 *P*-Multigrid

In p-multigrid, the different grids have the same number and size of finite elements but with different order basis functions. The grid transfer operators can be represented elementwise and can thus be easily represented in the matrix-free form of Equation 2.7.

The prolongation operator from the coarse to the fine grid interpolates low-order basis functions at the nodes for the high-order basis functions. Figure 4.1 shows the evaluation of second order basis function on the Gauss-Lobatto nodes for a fourth-order basis. This basis evaluation operation will be extended to provide an alternate representation for LFA of h-multigrid.

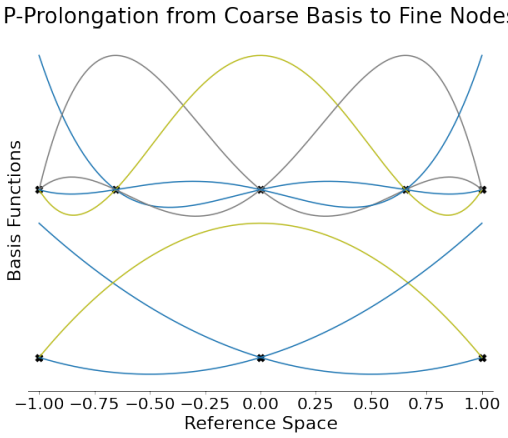


Figure 4.1: P-Prolongation from Coarse Basis to Fine Basis

With this coarse to fine basis interpolation, the matrix-free formulation of the prolongation operator can therefore be represented by

$$\begin{aligned} \mathbf{P}_{\text{ctof}} &= \mathbf{P}_f^T \mathbf{G}_f^T \mathbf{P}^e \mathbf{G}_c \mathbf{P}_c \\ \mathbf{P}^e &= \mathbf{I}\mathbf{D}_{\text{scale}} \mathbf{B}_{\text{ctof}} \end{aligned} \tag{4.4}$$

where $\mathbf{B}_{\text{ctof}}$ is an interpolation operator from the coarse grid basis to the fine grid basis, $\mathbf{P}_f$ and $\mathbf{G}_f$ are the fine grid element assembly operators, $\mathbf{P}_c$ and $\mathbf{G}_c$ are the coarse grid element assembly

operators, and $\mathbf{D}_{\text{scale}}$ is a pointwise scaling operator to account for node multiplicity across element interfaces. Restriction from the fine grid to the coarse grid is given by the transpose, $\mathbf{R}_{\text{ftoc}} = \mathbf{P}_{\text{ctof}}^T$.

It is useful to think of the p-prolongation operation as an interpolation operation between the coarse and fine grids, but in practice it can be easier to construct the prolongation basis $\mathbf{B}_{\text{ctof}}$ from the coarse and fine grid interpolation operators, provided that both bases use the same quadrature space.

$$\mathbf{B}_f = \mathbf{QR}, \quad \mathbf{B}_{\text{ctof}} = \mathbf{R}^{-1}\mathbf{Q}^T\mathbf{B}_c \tag{4.5}$$

In Equation 4.5, we form the interpolation operation between the coarse grid from the coarse grid interpolation operator and the QR factorization of the fine grid interpolation operator, assuming that the coarse and fine grid bases share the same quadrature space. Note that in the case of H^1 Lagrange bases, this factorization will produce the same coarse to fine grid interpolation operator as evaluating the coarse grid basis functions at the fine grid nodes; however, this formulation has the benefit of generalizing to a wider range of finite element bases.

Following the derivation from Section 3.1, we can derive the symbols of $\mathbf{P}_{\text{ctof}}$ and $\mathbf{R}_{\text{ftoc}}$.

Definition 4.2.2 (Symbol of P-prolongation Operator) The symbol of the p-prolongation operator is given by

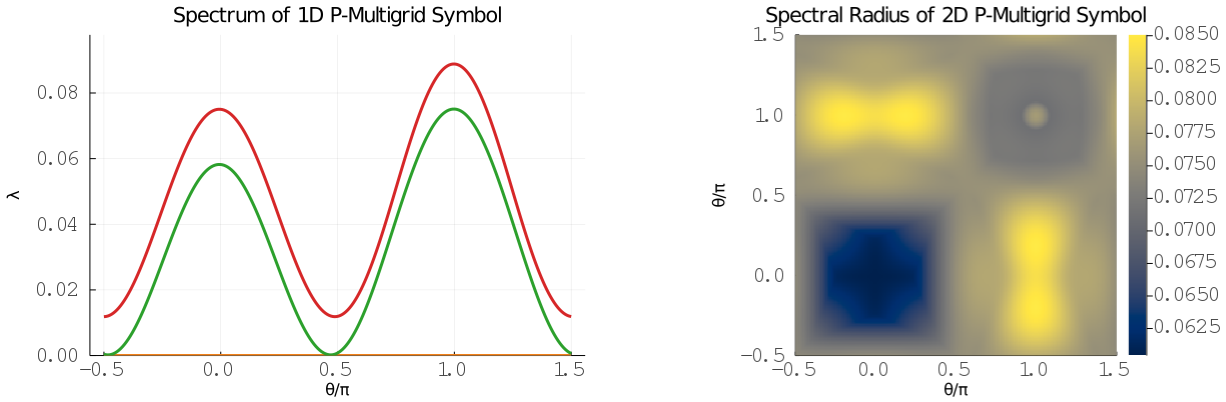
$$\tilde{\mathbf{P}}_{\text{ctof}}(\boldsymbol{\theta}) = \mathbf{Q}_f^T \left(\mathbf{P}^e \odot \left[e^{\imath \left(\mathbf{x}_{j,c} - \mathbf{x}_{i,f} \right) \cdot \boldsymbol{\theta} / \mathbf{h}} \right] \right) \mathbf{Q}_c \tag{4.6}$$

where $i \in \left\{ 1, 2, \ldots, n\left(p_{\text{fine}} + 1\right)^d \right\}$, $\mathbf{h}$ is the length of the element, and $j \in \left\{ 1, 2, \ldots, n\left(p_{\text{coarse}} + 1\right)^d \right\}$, n is the number of components, p_{fine} and p_{coarse} are the polynomial orders of the fine and coarse grid discretizations, and d is the dimension of the finite element basis. The matrices $\mathbf{Q}_f$ and $\mathbf{Q}_c$ are the localization operators for the fine and coarse grid, respectively, and the element p-prolongation operator is given by $\mathbf{P}^e = \mathbf{I}\mathbf{D}_{\text{scale}}\mathbf{B}_{\text{ctof}}$. The nodes $\mathbf{x}_{j,f}$ and are $\mathbf{x}_{i,c}$ are on the fine and coarse grids, respectively.

Definition 4.2.3 (Symbol of P-restriction Operator) The symbol of p-restriction operator is given by the expression

$$\tilde{\mathbf{R}}_{\text{ftoc}}(\boldsymbol{\theta}) = \mathbf{Q}_c^T \left(\mathbf{R}^e \odot \left[e^{\imath \left(\mathbf{x}_{j,f} - \mathbf{x}_{i,c} \right) \cdot \boldsymbol{\theta} / \mathbf{h}} \right] \right) \mathbf{Q}_f \tag{4.7}$$

where $i \in \left\{1, 2, \ldots, n\left(p_{\text{coarse}}+1\right)^d\right\}$, $\mathbf{h}$ is the length of the element, and $j \in \left\{1, 2, \ldots, n\left(p_{\text{fine}}+1\right)^d\right\}$, n is the number of components, p_{fine} and p_{coarse} are the polynomial orders of the fine and coarse grid discretizations, and d is the dimension of the finite element basis. The matrices $\mathbf{Q}_f$ and $\mathbf{Q}_c$ are the localization operators for the fine and coarse grid, respectively, and the element p-restriction operator is given by $\mathbf{R}^e = \mathbf{P}^{e,T} = \mathbf{B}^T_{\text{ctof}}\mathbf{D}_{\text{scale}}\mathbf{I}$. The nodes $\mathbf{x}_{j,c}$ and are $\mathbf{x}_{i,f}$ are on the coarse and fine grids, respectively.



(a) Spectrum of P-Multigrid for $p_f = 4$, $p_c = 2$

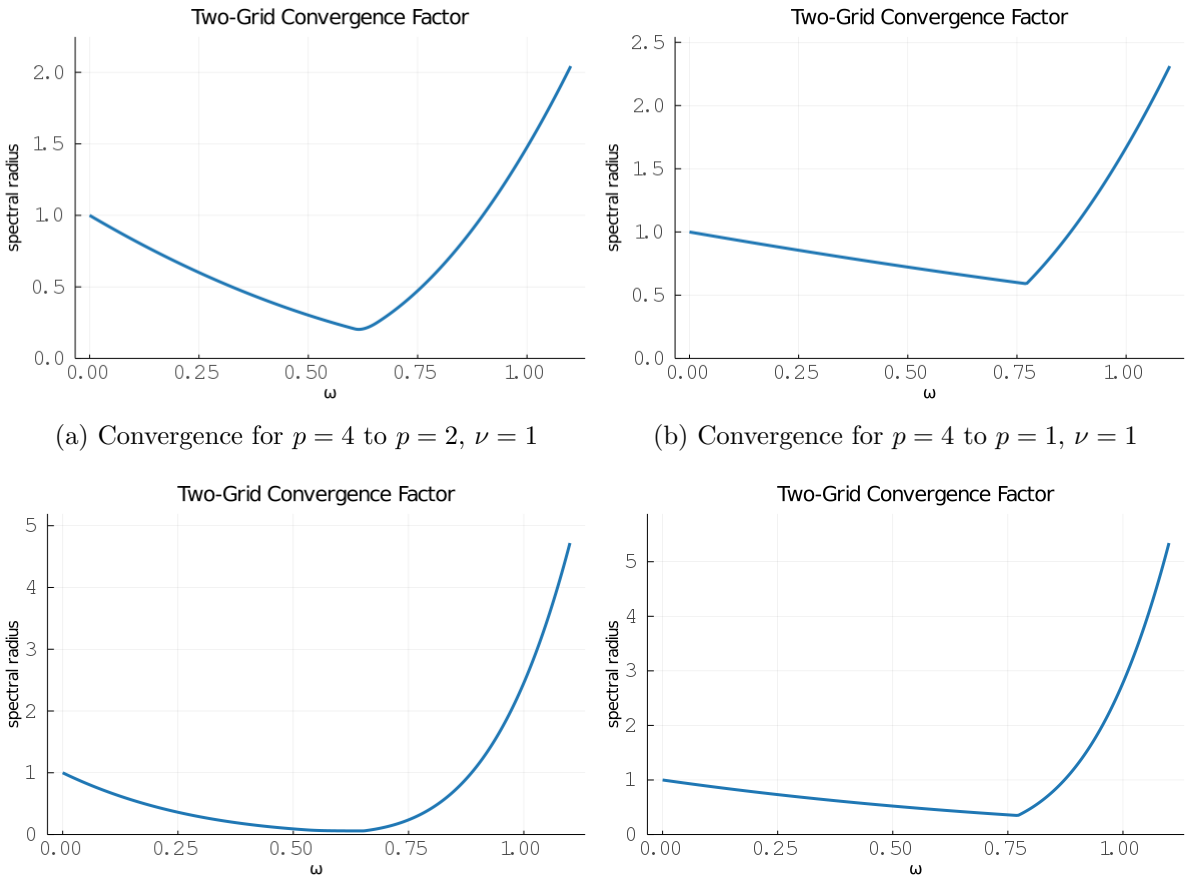
(b) Spectrum of P-Multigrid for $p_f = 4$, $p_c = 2$

Figure 4.2: Spectral Radius of P-Multigrid Symbol for $p_f = 4$, $p_c = 2$

In Figures 4.2a and 4.2b, we see the spectral radius of the symbol of p-multigrid for the scalar diffusion operator with third-order Chebyshev smoothing on a fine grid with a fourth-order H^1 Lagrange finite element basis and a coarse grid with a second-order H^1 Lagrange finite element basis on the Gauss-Lobatto points in one and two dimensions. Different smoothing techniques will result in different spectral radaii and different effectiveness of the multigrid algorithm in different frequency ranges.

4.2.2 *P*-Multigrid Numerical Results

In this section, we present numerical results for this analysis for the scalar Laplacian in one and two dimensions with H^1 Lagrange bases on Gauss-Lobatto points with Gauss-Legendre quadrature. Then we consider linear elasticity in three dimensions.



(a) Convergence for $p = 4$ to $p = 2$, $\nu = 1$

(b) Convergence for $p = 4$ to $p = 1$, $\nu = 1$

(c) Convergence for $p = 4$ to $p = 2$, $\nu = 2$

(d) Convergence for $p = 4$ to $p = 1$, $\nu = 2$

Figure 4.3: Two-Grid Analysis for P-Multigrid with Jacobi Smoothing for the 1D Laplacian

4.2.2.1 Scalar Laplacian - 1D Convergence Factors

In Figure 4.3a and Figure 4.3b we plot the two-grid convergence factor for p-multigrid with a single iteration of Jacobi pre and post-smoothing for the one dimensional Laplacian as a function of the Jacobi smoothing parameter ω, and in Figure 4.3c and Figure 4.3d we plot the two-grid convergence factor for p-multigrid with two iterations of Jacobi pre and post-smoothing for the one dimensional Laplacian as a function of the Jacobi smoothing parameter ω. On the left we show conservative coarsening from quartic to quadratic elements and on the right we show more aggressive coarsening from quartic to linear elements. As expected, the two-grid convergence factor decreases as we coarsen more rapidly. Also, the effect of underestimating the optimal Jacobi smoothing pa-

rameter, ω, is less pronounced than the effect of overestimating the smoothing parameter, especially with a higher number of pre and post-smooths.

In contrast to the previous work on h-multigrid for high-order finite elements, [38], poorly chosen values of $\omega < 1.0$ can result in a spectral radius of the p-multigrid error propagation symbol that is greater than 1, indicating that application of p-multigrid with Jacobi smoothing at these parameter values will result in increased error.

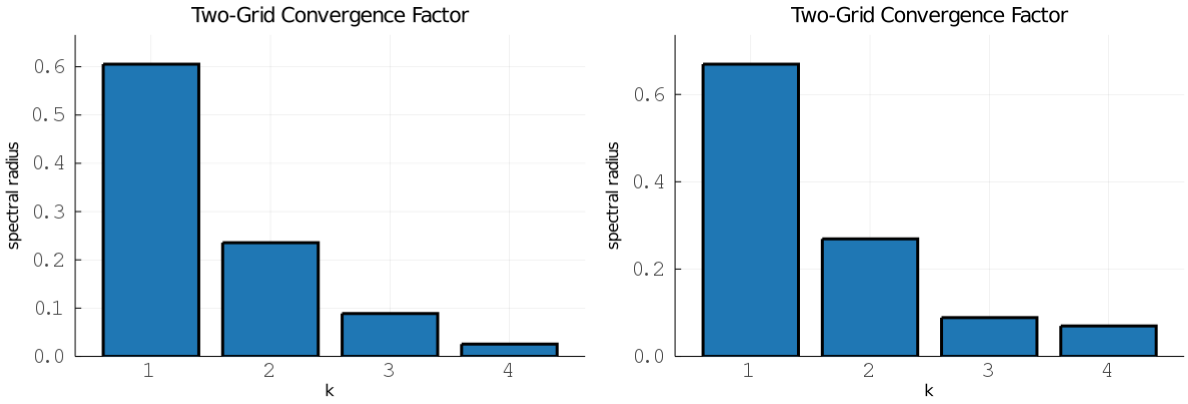
p_{fine} to p_{coarse}	$\nu = 1$		$\nu = 2$		$\nu = 3$	
	$\rho_{\min}$	ω_{opt}	$\rho_{\min}$	ω_{opt}	$\rho_{\min}$	ω_{opt}
$p = 2$ to $p = 1$	0.137	0.63	0.060	0.69	0.041	0.72
$p = 4$ to $p = 2$	0.204	0.62	0.059	0.64	0.045	0.70
$p = 4$ to $p = 1$	0.591	0.77	0.350	0.77	0.207	0.77
$p = 8$ to $p = 4$	0.250	0.60	0.068	0.60	0.033	0.63
$p = 8$ to $p = 2$	0.668	0.73	0.446	0.73	0.298	0.73
$p = 8$ to $p = 1$	0.874	0.78	0.764	0.78	0.668	0.78
$p = 16$ to $p = 8$	0.300	0.57	0.090	0.57	0.035	0.58
$p = 16$ to $p = 4$	0.719	0.69	0.517	0.69	0.371	0.69
$p = 16$ to $p = 2$	0.906	0.73	0.820	0.73	0.743	0.73
$p = 16$ to $p = 1$	0.968	0.74	0.936	0.74	0.906	0.74

Table 4.1: Two-Grid Convergence Factor and Optimal Jacobi Smoothing Parameter for P-Multigrid with Jacobi Smoothing for the 1D Laplacian

The results in Table 4.1 provide the LFA convergence factor and optimal values of ω for two-grid high-order p-multigrid for a variety of polynomial orders and coarsening factors.

Traditional estimates of the optimal smoothing parameter based upon extremal eigenvalues of the preconditioned operator are incompatible with this LFA framework. Optimal parameter estimation is an open question for high-order p-multigrid, but optimization techniques, such as those discussed in [14], can be used to tune these parameters, especially for more complex PDEs.

In Figure 4.4a and Figure 4.4b we plot the two-grid convergence factor for p-multigrid with Chebyshev pre and post-smoothing for the one dimensional Laplacian as a function of the Chebyshev order, k. On the left we show conservative coarsening from quartic to quadratic elements and



(a) Convergence for $p = 4$ to $p = 2$, $\nu = 1$

(b) Convergence for $p = 4$ to $p = 1$, $\nu = 1$

Figure 4.4: Two-Grid Analysis for P-Multigrid with Chebyshev Smoothing for the 1D Laplacian

on the right we show more aggressive coarsening from quartic to linear elements. As expected, the two-grid convergence factor decreases as we coarsen more rapidly.

p_{fine} to p_{coarse}	$k=1$	$k=2$	$k=3$	$k=4$
$p=2$ to $p=1$	0.545	0.220	0.063	0.017
$p=4$ to $p=2$	0.576	0.222	0.089	0.025
$p=4$ to $p=1$	0.623	0.269	0.089	0.070
$p=8$ to $p=4$	0.638	0.244	0.074	0.022
$p=8$ to $p=2$	0.657	0.260	0.097	0.059
$p=8$ to $p=1$	0.881	0.674	0.510	0.393
$p=16$ to $p=8$	0.664	0.253	0.075	0.022
$p=16$ to $p=4$	0.714	0.328	0.135	0.059
$p=16$ to $p=2$	0.907	0.741	0.602	0.496
$p=16$ to $p=1$	0.970	0.912	0.857	0.809

Table 4.2: Two-Grid Convergence Factor for P-Multigrid with Chebyshev Smoothing for the 1D Laplacian

The results in Table 4.2 provide the LFA convergence factor and optimal values of k for two-grid high-order p-multigrid for a variety of coarsening rates and orders of Chebyshev smoother. From this table, we can see that the effectiveness of higher order Chebyshev smoothers degrades as we coarsen more aggressively, but Chebyshev smoothing still provides better two-grid convergence than multiple pre and post-smoothing Jacobi iterations.

The results in Table 4.3 provide the LFA-predicted convergence factor for two-grid high-order p-multigrid for a variety of coarsening rates and orders of Chebyshev smoother with different scaling factors for the minimum eigenvalue estimate used in the Chebyshev iterations. Increasing the lower eigenvalue estimate results in the Chebyshev method better targeting high frequency error modes, which results in improved two-grid convergence when halving the polynomial degree of the basis functions. However, increasing the lower eigenvalue estimate results in worse two-grid convergence for aggressive coarsening directly to linear elements.

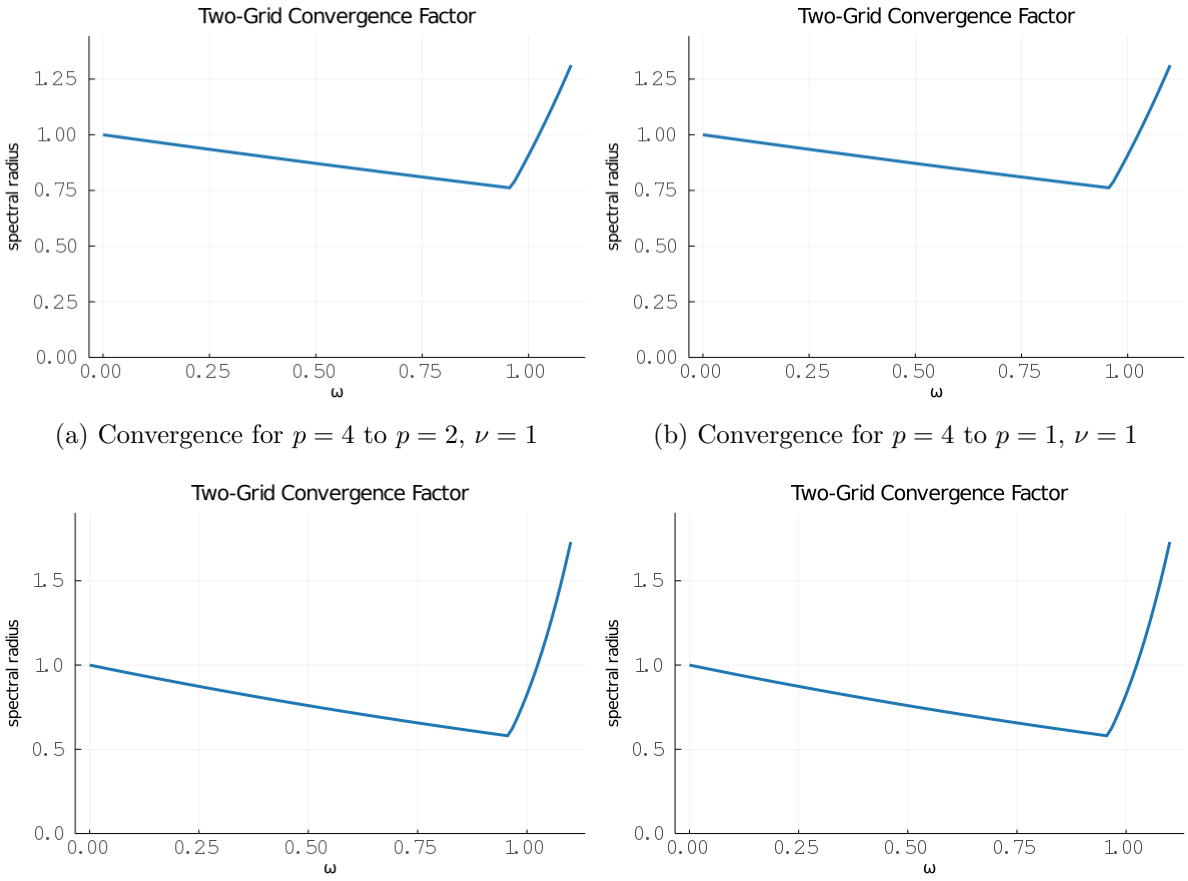
$\lambda_{\min} = 0.2\hat{\lambda}_{\max}$				
p_{fine} to p_{coarse}	$k=1$	$k=2$	$k=3$	$k=4$
$p=4$ to $p=2$	0.410	0.093	0.043	0.024
$p=4$ to $p=1$	0.611	0.250	0.106	0.071
$p=8$ to $p=4$	0.435	0.081	0.016	0.007
$p=8$ to $p=1$	0.891	0.739	0.623	0.529
$p=16$ to $p=8$	0.443	0.081	0.015	0.006
$p=16$ to $p=1$	0.973	0.931	0.894	0.861
$\lambda_{\min} = 0.3\hat{\lambda}_{\max}$				
p_{fine} to p_{coarse}	$k=1$	$k=2$	$k=3$	$k=4$
$p=4$ to $p=2$	0.279	0.070	0.042	0.031
$p=4$ to $p=1$	0.638	0.332	0.184	0.104
$p=8$ to $p=4$	0.289	0.050	0.023	0.012
$p=8$ to $p=1$	0.899	0.777	0.682	0.599
$p=16$ to $p=8$	0.294	0.055	0.020	0.010
$p=16$ to $p=1$	0.975	0.942	0.913	0.885

Table 4.3: Two-Grid Convergence Factor for P-Multigrid with Chebyshev Smoothing for the 1D Laplacian with Modified Lower Eigenvalue Bound

4.2.2.2 Scalar Laplacian - 2D Convergence Factors

In Figure 4.5b and Figure 4.5a we plot the two-grid convergence factor for p-multigrid with a single iteration of Jacobi pre and post-smoothing for the two dimensional Laplacian as a function of the Jacobi smoothing parameter ω, and in Figure 4.5d and Figure 4.5c we plot the two-grid convergence factor for p-multigrid with two iterations of Jacobi pre and post-smoothing for the two dimensional Laplacian as a function of the Jacobi smoothing parameter ω. On the left we show conservative coarsening from quartic to quadratic elements and on the right we show more aggressive coarsening from quartic to linear elements. As we saw with one dimension, the two-grid convergence factor decreases as we coarsen more rapidly. Also, the effect of underestimating the optimal Jacobi smoothing parameter, ω, is less pronounced than the effect of overestimating the smoothing parameter, especially with a higher number of pre and post-smooths.

The results in Table 4.4 provide the LFA convergence factor and optimal values of ω for



(a) Convergence for $p = 4$ to $p = 2$, $\nu = 1$

(b) Convergence for $p = 4$ to $p = 1$, $\nu = 1$

(c) Convergence for $p = 4$ to $p = 2$, $\nu = 2$

(d) Convergence for $p = 4$ to $p = 1$, $\nu = 2$

Figure 4.5: Two-Grid Convergence for P-Multigrid with Jacobi Smoothing for the 2D Laplacian

p_{fine} to p_{coarse}	$\nu = 1$		$\nu = 2$		$\nu = 3$	
	$\rho_{\min}$	ω_{opt}	$\rho_{\min}$	ω_{opt}	$\rho_{\min}$	ω_{opt}
$p = 2$ to $p = 1$	0.230	0.95	0.091	0.99	0.061	1.03
$p = 4$ to $p = 2$	0.388	0.82	0.151	0.82	0.078	0.83
$p = 4$ to $p = 1$	0.763	0.95	0.582	0.95	0.444	0.95
$p = 8$ to $p = 4$	0.646	0.79	0.418	0.79	0.272	0.79
$p = 8$ to $p = 2$	0.858	0.84	0.737	0.84	0.633	0.84
$p = 8$ to $p = 1$	0.952	0.87	0.907	0.87	0.864	0.87

Table 4.4: Two-Grid Convergence Factor and Optimal Jacobi Smoothing Parameter for P-Multigrid with Jacobi Smoothing for the 2D Laplacian

two-grid high-order p-multigrid for a variety of polynomial orders and coarsening factors.

p_{fine} to p_{coarse}	$k = 1$	$k = 2$	$k = 3$	$k = 4$
$p = 2$ to $p = 1$	0.621	0.252	0.075	0.039
$p = 4$ to $p = 2$	0.607	0.281	0.085	0.047
$p = 4$ to $p = 1$	0.768	0.424	0.219	0.127
$p = 8$ to $p = 4$	0.669	0.278	0.110	0.055
$p = 8$ to $p = 2$	0.864	0.633	0.456	0.336
$p = 8$ to $p = 1$	0.956	0.873	0.795	0.730
$p = 16$ to $p = 8$	0.855	0.613	0.435	0.319
$p = 16$ to $p = 4$	0.938	0.822	0.719	0.634
$p = 16$ to $p = 2$	0.976	0.928	0.882	0.842
$p = 16$ to $p = 1$	0.992	0.975	0.959	0.944

Table 4.5: Two-Grid Convergence Factor for P-Multigrid with Chebyshev Smoothing for the 2D Laplacian

The results in Table 4.5 provide the LFA convergence factor and optimal values of k for two-grid high-order p-multigrid for a variety of coarsening rates and orders of Chebyshev smoother. The two-grid convergence factor still degrades and the effectiveness of higher order Chebyshev smoothers is again reduced as we coarsen more aggressively.

The results in Table 4.6 provide the LFA-predicted convergence factor and optimal values of k for two-grid high-order p-multigrid for a variety of coarsening rates and orders of Chebyshev

$\lambda_{\min} = 0.2\hat{\lambda}_{\max}$				
p_{fine} to p_{coarse}	$k=1$	$k=2$	$k=3$	$k=4$
$p=4$ to $p=2$	0.450	0.137	0.067	0.050
$p=4$ to $p=1$	0.786	0.525	0.362	0.255
$p=8$ to $p=4$	0.668	0.330	0.172	0.106
$p=8$ to $p=1$	0.960	0.899	0.848	0.801
$\lambda_{\min} = 0.3\hat{\lambda}_{\max}$				
p_{fine} to p_{coarse}	$k=1$	$k=2$	$k=3$	$k=4$
$p=4$ to $p=2$	0.407	0.106	0.073	0.059
$p=4$ to $p=1$	0.803	0.590	0.447	0.341
$p=8$ to $p=4$	0.691	0.409	0.256	0.164
$p=8$ to $p=1$	0.963	0.915	0.874	0.835

Table 4.6: Two-Grid Convergence Factor for P-Multigrid with Chebyshev Smoothing for the 2D Laplacian with Modified Lower Eigenvalue Bound

smoother with different scaling factors for the minimum eigenvalue estimate used in the Chebyshev iterations. As in one dimension, increasing the lower eigenvalue estimate results in better two-grid convergence when conventional coarsening by halving the polynomial degree of the basis functions but results in worse two-grid convergence with aggressive coarsening.

4.2.2.3 Linear Elasticity - 3D Convergence Factors

To demonstrate the suitability of this LFA formulation for more complex PDE, we consider linear elasticity in three dimensions. The derivation of the linear elasticity problem was given in Section 2.5.3.

The results in Table 4.7 provide the LFA convergence factor and optimal values of k for two-grid high-order p-multigrid for a variety of coarsening rates and orders of Chebyshev smoother.

4.2.3 *P*-Multigrid Validation with Numerical Experiments

In this section, we compare the LFA two-grid convergence factors to numerical results. Our numerical experiments were conducted using the libCEED [1] with PETSc [6] multigrid example

p_{fine} to p_{coarse}	$k=1$	$k=2$	$k=3$	$k=4$
$p=2$ to $p=1$	0.621	0.252	0.075	0.039
$p=4$ to $p=2$	0.607	0.281	0.085	0.047
$p=4$ to $p=1$	0.768	0.424	0.219	0.127
$p=8$ to $p=4$	0.669	0.278	0.110	0.055
$p=8$ to $p=2$	0.864	0.633	0.456	0.336
$p=8$ to $p=1$	0.956	0.873	0.795	0.730

Table 4.7: Two-Grid Convergence Factor for P-Multigrid with Chebyshev Smoothing for 3D Linear Elasticity

found in the libCEED repository. PETSc provides the mesh management, linear solvers, and multigrid preconditioner while libCEED provides the matrix-free operator evaluation.

We recover the manufactured solution given by

$$f(x,y,z) = xyz\sin(\pi x)\sin(\pi(1.23+0.5y))\sin(\pi(2.34+0.25z)) \tag{4.8}$$

on the domain $[-3,3]^3$ with Dirichlet boundary conditions for finite element discretizations with varying orders on approximately 8 million degrees of freedom for a variety of test cases. Although LFA is defined on infinite grids, which most naturally translates to periodic problems, LFA-predicted convergence factors are also accurate for appropriate problems with other boundary conditions on finite grids [73].

Since the Chebyshev smoothing is based upon the Jacobi preconditioned operator, it is important to validate the LFA of the Jacobi smoothing before considering Chebyshev smoothing. We use simple Jacobi smoothing with a weight of $\omega = 1.0$ to validate the LFA.

The results in Table 4.8 provide the LFA and experimental convergence factors for the test problem. As expected, the high-order fine grid problems diverge with a smoothing factor of $\omega = 1.0$; however, the LFA provides reasonable upper bounds on the true convergence factor seen in the experimental results.

We used the LFA estimates of the maximal eigenvalue to set the extremal eigenvalues used the Chebyshev iteration in PETSc, using $\lambda_{\min} = 0.1\hat{\lambda}_{\max}$ and $\lambda_{\max} = 1.0\hat{\lambda}_{\max}$, where $\hat{\lambda}_{\max}$ is the

p_{fine} to p_{coarse}	LFA	libCEED
$p = 2$ to $p = 1$	0.312	0.301
$p = 4$ to $p = 2$	1.436	1.402
$p = 4$ to $p = 1$	1.436	1.401
$p = 8$ to $p = 4$	1.989	1.885
$p = 8$ to $p = 2$	1.989	1.874
$p = 8$ to $p = 1$	1.989	1.875

Table 4.8: LFA and Experimental Two-Grid Convergence Factor for P-Multigrid with Jacobi Smoothing for the 3D Laplacian with $\omega = 1.0$

estimated maximal eigenvalue of the symbol of the Jacobi preconditioned operator.

p_{fine} to p_{coarse}	$k = 2$		$k = 3$		$k = 4$	
	LFA	libCEED	LFA	libCEED	LFA	libCEED
$p = 2$ to $p = 1$	0.253	0.222	0.076	0.058	0.041	0.033
$p = 4$ to $p = 2$	0.277	0.251	0.111	0.097	0.062	0.050
$p = 4$ to $p = 1$	0.601	0.587	0.416	0.398	0.295	0.276
$p = 8$ to $p = 4$	0.398	0.391	0.197	0.195	0.121	0.110
$p = 8$ to $p = 2$	0.748	0.743	0.611	0.603	0.506	0.469
$p = 8$ to $p = 1$	0.920	0.914	0.871	0.861	0.827	0.814

Table 4.9: LFA and Experimental Two-Grid Convergence Factor for P-Multigrid with Chebyshev Smoothing for the 3D Laplacian

The LFA provides reasonable upper bounds on the true convergence factor seen in the experimental results. As with the one and two dimensional results, rapid coarsening of the polynomial order of the bases decreases the effectiveness of higher order Chebyshev smoothing.

4.2.4 *H*-Multigrid

In h-multigrid, the different grids have the same polynomial order of finite elements but these elements are amalgamated into increasingly larger elements. For LFA of h-multigrid, we introduce the concept of macro-element bases. With macro-element bases, the grid transfer operators can be still represented elementwise and can thus be easily represented in the matrix-free form of Equation

2.7.

A macro-element basis is formed by amalgamating two or more finite element bases. Each sub-element has separate quadrature spaces, so the basis functions for each sub-element, are defined as zero on the portions of the domain not included in the given sub-element. In effect, the basis interpolation and gradient operations for the macro-element provide a sub-element restriction and sub-element basis operation together.

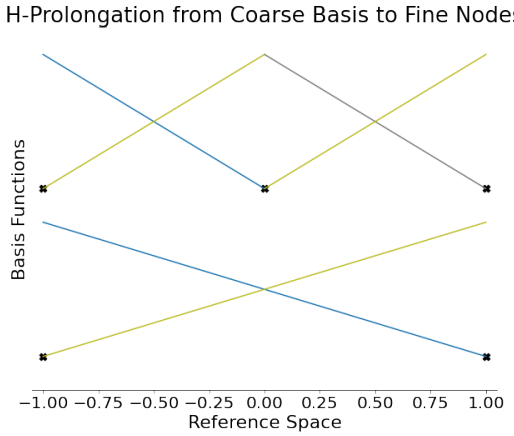


Figure 4.6: H-Prolongation from Coarse Basis to Fine Basis

In Figure 4.6, we see an example of interpolation from a coarse grid basis to a fine grid macro-element basis. The linear shape functions are evaluated at the nodes of the fine grid macro-element, which is a pair of linear sub-elements. On the left linear sub-element, there are two basis functions which are zero over the domain of the right sub-element, and the reverse is also true. The element level operator, $\mathbf{A}_e$, on the fine grid macro-element assembles the element level operator for each sub-element into the action of the PDE operator on the full macro-element.

With this macro-element structure, we can form the Fourier mode localization operator for the macro-element in the same fashion as the Fourier mode localization operator for high-order elements given in Equation 3.6. With macro-elements, there are mp columns in the localization operator for a one dimensional scalar PDE on a macro-element with m sub-elements instead of p

columns for a one dimensional scalar PDE on a single high order element.

$$\mathbf{Q} = \begin{bmatrix} \mathbf{I} \\ \mathbf{e}_0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 \\ 1 & 0 & \cdots & 0 \end{bmatrix} \tag{4.9}$$

Following the derivation from Section 3.1, we can derive the symbols of $\mathbf{P}_{\text{ctof}}$ and $\mathbf{R}_{\text{ftoc}}$.

Definition 4.2.4 (Symbol of H-prolongation Operator) The symbol of the h-prolongation operator is given by

$$\tilde{\mathbf{P}}_{\text{ctof}}(\boldsymbol{\theta}) = \mathbf{Q}_f^T \left(\mathbf{P}^e \odot \left[e^{\imath \left(\mathbf{x}_{j,c} - \mathbf{x}_{i,f} \right) \cdot \boldsymbol{\theta} / \mathbf{h}} \right] \right) \mathbf{Q}_c \tag{4.10}$$

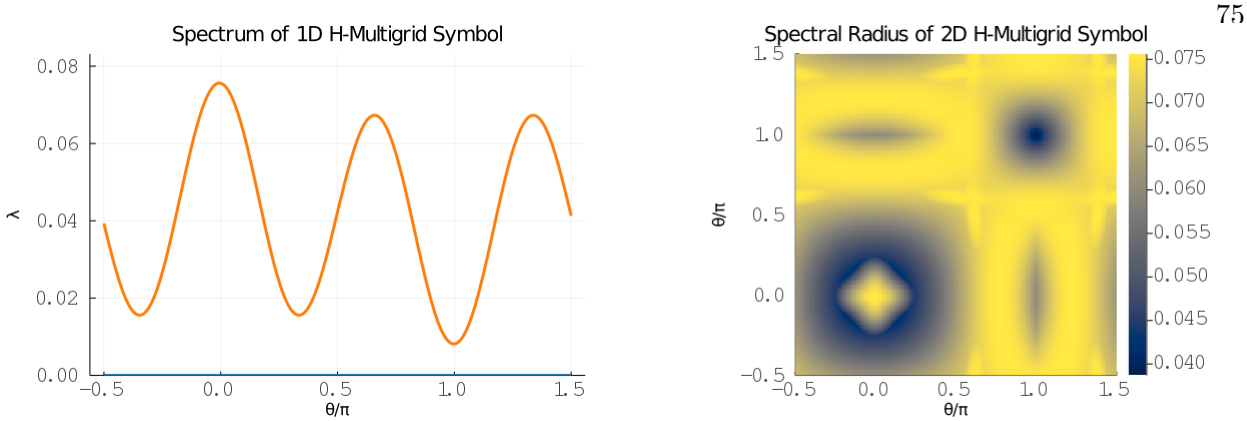
where $i \in \left\{ 1, 2, \ldots, n\left(mp+1\right)^d \right\}$, $j \in \left\{ 1, 2, \ldots, n\left(p+1\right)^d \right\}$, $\mathbf{h}$ is the length of the macro-element, d is the dimension of the finite element bases, n is the number of components, and m is the number of sub-elements in each fine grid macro-element. The matrices $\mathbf{Q}_f$ and $\mathbf{Q}_c$ are the localization operators for the fine and coarse grid, respectively, and the macro-element h-prolongation operator is given by $\mathbf{P}^e = \mathbf{I}\mathbf{D}_{\text{scale}}\mathbf{B}_{\text{ctof}}$.

Definition 4.2.5 (Symbol of H-restriction Operator) The symbol of h-restriction operator is given by the expression

$$\tilde{\mathbf{R}}_{\text{ftoc}}(\boldsymbol{\theta}) = \mathbf{Q}_c^T \left(\mathbf{R}^e \odot \left[e^{\imath \left(\mathbf{x}_{j,f} - \mathbf{x}_{i,c} \right) \cdot \boldsymbol{\theta} / \mathbf{h}} \right] \right) \mathbf{Q}_f \tag{4.11}$$

where $i \in \left\{ 1, 2, \ldots, n\left(p+1\right)^d \right\}$, $j \in \left\{ 1, 2, \ldots, n\left(mp+1\right)^d \right\}$, $\mathbf{h}$ is the length of the macro-element, d is the dimension of the finite element bases, n is the number of components, and m is the number of sub-elements in each fine grid macro-element. The matrices $\mathbf{Q}_f$ and $\mathbf{Q}_c$ are the localization operators for the fine and coarse grid, respectively, and the macro-element h-restriction operator is given by $\mathbf{R}^e = \mathbf{P}^{e,T} = \mathbf{B}_{\text{ctof}}^T \mathbf{D}_{\text{scale}} \mathbf{I}$.

In Figures 4.7a and 4.7b, we see the spectral radius of the symbol of h-multigrid for the scalar diffusion operator with third-order Chebyshev smoothing on a fine grid with a fourth-order



(a) Spectrum of H-Multigrid for $p = 1$

(b) Spectrum of H-Multigrid for $p = 1$

Figure 4.7: Spectrum of H-Multigrid Symbol for $p = 1$

H^1 Lagrange finite element basis and a coarse grid with a second-order H^1 Lagrange finite element basis on the Gauss-Lobatto points in one and two dimensions. Various preconditioning techniques will reduce this spectral radius, with different effectiveness in different frequency ranges.

4.2.5 *H*-Multigrid Validation with Previous Work

By representing the fine grid with macro-elements and the prolongation operator with this interpolation, this LFA of p-multigrid exactly reproduces the results of He and Maclachlan [38] for LFA of high-order h-multigrid even though this previous work did not use the concept of macro-elements. The results from this LFA can be seen in Table 4.10.

On uniform rectangular meshes, linear finite elements produce the same discretized operator as finite differencing. The nine-point stencil for the Laplace operator in two dimensions is given by

$$\frac{1}{3}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{4.12}$$

with a corresponding local Fourier analysis symbol given by

p, d	$\nu = (0,1)$		$\nu = (1,1)$		$\nu = (2,2)$	
	ρ	ω	ρ	ω	ρ	ω
$p = 2, d = 1$	0.821	1.000	0.821	1.000	1.279	1.000
$p = 2, d = 1$	0.526	0.838	0.495	0.838	0.302	0.838
$p = 2, d = 1$	0.291	0.709	0.249	0.709	0.064	0.709
$p = 3, d = 1$	0.491	0.650	0.337	0.650	0.131	0.650
$p = 4, d = 1$	0.608	0.640	0.559	0.640	0.331	0.640
$p = 2, d = 2$	0.452	1.000	0.288	1.000	0.091	1.000

Table 4.10: Two-Grid Convergence Factor and Jacobi Smoothing Parameter for High-Order H-Multigrid

$$\tilde{A}(\theta_1, \theta_2) = \frac{8}{3} - \frac{2}{3}\cos(\theta_1) - \frac{2}{3}\cos(\theta_2) - \frac{4}{3}\cos(\theta_1)\cos(\theta_2) \tag{4.13}$$

The assembled matrix for a single linear element is given by

$$\mathbf{A}^e = \frac{1}{3}\begin{bmatrix} 2 & -1/2 & -1/2 & -1 \\ -1/2 & 2 & -1 & -1/2 \\ -1/2 & -1 & 2 & -1/2 \\ -1 & -1/2 & -1/2 & 2 \end{bmatrix} \tag{4.14}$$

with a corresponding local Fourier analysis symbol given by

$$\begin{aligned} \tilde{\mathbf{A}}(\theta_1, \theta_2) &= \mathbf{Q}^T \left(\mathbf{A}^e \odot \left[e^{\imath(\mathbf{x}_j - \mathbf{x}_i)\cdot\boldsymbol{\theta}} \right] \right) \mathbf{Q} \\ &= \frac{8}{3} - \frac{2}{3}\cos(\theta_1) - \frac{2}{3}\cos(\theta_2) - \frac{4}{3}\cos(\theta_1)\cos(\theta_2). \end{aligned} \tag{4.15}$$

We can use the LFA of multigrid given by Definition 4.2.1 with the symbols of the h-multigrid transfer operators givn by Definition 4.2.4 and Definition 4.2.5 to reproduce LFA of h-multigrid methods for finite differencing where the stencil can be represented by a finite element discretization. This LFA of arbitrary second-order PDEs with high-order finite element discretizations agrees with previous work on LFA of PDE operators derived with finite differencing with analogus stencils.

Our LFA of h-multigrid presented here is based on a specific basis of the Fourier space used in [51] rather than the commonly used Fourier modes, see [82, 86], where the symbol of each component in multigrid methods are formed based on different harmonic frequencies. The symbols of grid-transfer operators described in Definition 4.2.4 and Definition 4.2.5 give a general way for multigrid coarsening with factor m, and this framework is simpler, especially for high-order discretizations described in [38]. Also, our LFA for h-multigrid is suitable for finite element discretizations without requiring bases to have uniformly spaced nodes.

As mentioned before, our focus of this work is p-multigrid, so we will not expand the discussion of h-multigrid further method here. Applying this LFA framework to h-multigrid or hp-multigrid methods are topics for future research.

Chapter 5

Balancing Domain Decomposition by Constraints

Balancing Domain Decomposition by Constraints (BDDC) is a relatively new technique created by Dohrmann in 2003 [20]. BDDC is an improvement upon Balancing Domain Decomposition (BDD) [60], which requires modifications to be effective for high-order problems, especially for three-dimensional problems. The dual primal version of finite element tearing and interconnecting (FETI-DP) [26] is closely related to BDDC and the two are effectively the same technique, under certain assumptions [61].

Domain decomposition techniques decompose the domain Ω into a series of subdomains Ω^i. Some techniques use overlapping subdomains, such as additive Schwartz methods, which are popular for high-order and spectral finite elements [29] and have been used as smoothers in multigrid methods [30]. In BDDC, we instead use non-overlapping subdomains with an energy minimization problem to resolve jumps in the values of degrees of freedom over subdomain interfaces.

Local Fourier Analysis (LFA) of BDDC was introduced by Brown, He, and MacLachlan [13]. We discuss implementing BDDC in a matrix-free fashion and re-derive the LFA of BDDC in the context of high-order elements, with single element and macro-element subdomains.

5.1 Matrix-Free Balancing Domain Decomposition by Constraints

For high-order matrix-free BDDC, we decompose the domain Ω into a series of subdomains Ω^e given by the individual elements, and we utilize four non-exclusive spaces on each subdomain. Figure 5.1a shows a global mesh before decomposition, consisting of four quadratic elements in two

dimensions in this case.

The interface between subdomains is given by $\Gamma = \bigcup \partial\Omega^e \backslash \partial\Omega$, and the interface of subdomain Ω^e is given by $\Gamma^e = \delta\Omega^e \bigcap \Gamma$. The interior of each subdomain, I, is given by the remaining degrees of freedom on the element. In Figure 5.1b, the interface degrees of freedom, Γ, for each subdomain have been highlighted and the interior I is given by the remainder of each subdomain.

Each subdomain problem could therefore be written as

$$\mathbf{A}^e = \begin{bmatrix} \mathbf{A}^e_{\mathrm{I,I}} & \mathbf{A}^{e,T}_{\Gamma,\mathrm{I}} \\ \mathbf{A}^e_{\Gamma,\mathrm{I}} & \mathbf{A}^e_{\Gamma,\Gamma} \end{bmatrix} \tag{5.1}$$

where $\mathbf{A}^e = \mathbf{B}^T\mathbf{D}\mathbf{B}$, as shown in Equation 3.4.

This subdomain problem can be assembled into the global problem in the typical finite element approach given by the matrix-free form in Equation 2.7, but in BDDC we instead create a partially subassembled problem that is easier to invert than the global problem by duplicating broken degrees of freedom along the subdomain boundaries. Only the corner, or vertex, degrees of freedom from each element are assembled into a global coarse problem while the remaining degrees of freedom on the subdomain interfaces are duplicated in each subdomain. The solutions on the global coarse problem and broken subdomain problems are used to construct an approximate solution to the true assembled global problem.

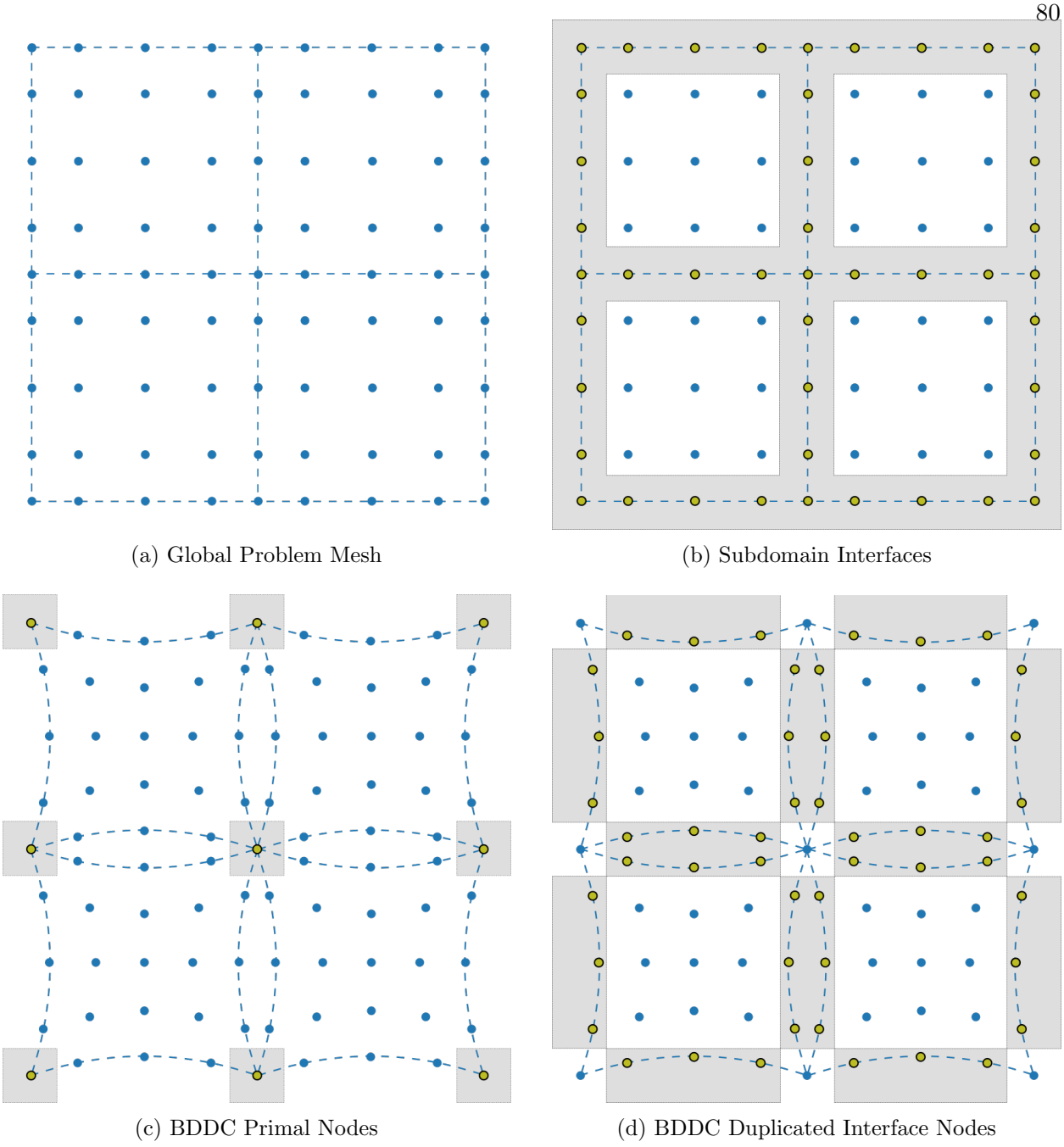
The coarse grid, or primal, nodes Π^e are given by the corners of the subdomain, so we have 4 primal nodes for each element in two dimensions and 8 primal nodes in three dimensions. The remainder of the subdomain we denote with an r. In Figure 5.1c the primal nodes have been highlighted.

The partially subassembled problem is therefore given by

$$\hat{\mathbf{A}} = \sum_{e=1}^{N} \mathbf{R}^{e,T} \hat{\mathbf{A}}^e \mathbf{R}^e \tag{5.2}$$

where

$$\hat{\mathbf{A}}^e = \begin{bmatrix} \mathbf{A}^e_{\mathrm{r,r}} & \hat{\mathbf{A}}^{e,T}_{\Pi,\mathrm{r}} \\ \hat{\mathbf{A}}^e_{\Pi,\mathrm{r}} & \hat{\mathbf{A}}^e_{\Pi,\Pi} \end{bmatrix} \tag{5.3}$$



(a) Global Problem Mesh

(b) Subdomain Interfaces

(c) BDDC Primal Nodes

(d) BDDC Duplicated Interface Nodes

Figure 5.1: Non-Overlapping Decomposition of Domain for BDDC

and $\mathbf{R}^e$ is the subdomain injection operator. In this formulation, the broken degrees of freedom found on the portion of subdomain interface given by $\Gamma^e - \Pi^e$ are duplicated for each subdomain. In Figure 5.1d the duplicated degrees of freedom across each subdomain interface have

been highlighted. This duplication reduces the amount of global communication required to solve the subassembled problem, which makes BDDC attractive as a preconditioner for high-order finite element methods.

There are BDDC formulations which enrich the primal space with edge or face averages to improve convergence. We forgo these formulations to simplify the analysis; however these formulations can become increasingly attractive with large, high-order subdomains in three dimensions. There is no fundamental barrier to extending the LFA of BDDC described below to formulations that use edge or face averages to improve convergence.

Definition 5.1.1 (Balancing Domain Decomposition by Constraints Preconditioner) The action of the BDDC preconditioner is given by

$$\mathbf{M}^{-1} = \mathbf{R}^T \hat{\mathbf{A}}^{-1} \mathbf{R} \tag{5.4}$$

where $\mathbf{R}$ is the subassembly injection operator. The inverse of the partially subassembled problem is computed by Schur complement

$$\hat{\mathbf{A}}^{-1} = \begin{bmatrix} \mathbf{I} & -\mathbf{A}_{\mathrm{r,r}}^{-1} \hat{\mathbf{A}}_{\Pi,\mathrm{r}}^T \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{\mathrm{r,r}}^{-1} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{S}}_{\Pi}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\hat{\mathbf{A}}_{\Pi,\mathrm{r}} \mathbf{A}_{\mathrm{r,r}}^{-1} & \mathbf{I} \end{bmatrix} \tag{5.5}$$

where $\hat{\mathbf{S}}_{\Pi} = \hat{\mathbf{A}}_{\Pi,\Pi} - \hat{\mathbf{A}}_{\Pi,\mathrm{r}} \mathbf{A}_{\mathrm{r,r}}^{-1} \hat{\mathbf{A}}_{\Pi,\mathrm{r}}^T$. Π denotes degrees of freedom on the assembled subdomain vertex space, while r denotes degrees of freedom on the broken space given by the subdomain interior and duplicated portions of the subdomain interface.

5.1.1 Injection Operators

The global subassembly injection operator, $\mathbf{R}$, maps the global degrees of freedom to the broken spaces of the partially subassembled problem on each subdomain. We consider the two injection operators discussed by Brown, He, and MacLachlan in [13].

The first injection operator, $\mathbf{R}_1$, the scaled injection operator, results in the lumped BDDC preconditioner, $\mathbf{M}_1^{-1}$.

Let $\mathcal{N}(x)$ denote the set of subdomains that contain a broken copy of degree of freedom x in the broken space. If $x \in \Pi^e$ or $x \in \mathrm{I}^e$, then $|\mathcal{N}(x)| = 1$, otherwise $|\mathcal{N}(x)|$ is the number of subdomains that contain x. The scaled injection operator, $\mathbf{R}_1$ is defined as the mapping between the global degrees of freedom and the broken space where each element is scaled by $1/|\mathcal{N}(x)|$. For interior and vertex degrees of freedom, the sparse matrix representation of $\mathbf{R}_1$ would contain a single entry with a value of 1, while each column corresponding to a duplicated degree of freedom has $|\mathcal{N}(x)|$ entries, each given by $1/|\mathcal{N}(x)|$.

Definition 5.1.2 (Scaled Injection Operator) The scaled injection operator, $\mathbf{R}_1$, is given by the mapping from the global problem space to the broken space where each degree of freedom is scaled by the reciprocal of its multiplicity in the broken space, denoted by $|\mathcal{N}(x)|^{-1}$.

Definition 5.1.3 (Lumped Balanced Domain Decomposition by Constraints) Let $\mathbf{R}_1$ be given by the scaled injection operator from Definition 5.1.2. The lumped BDDC preconditioner is given by

$$\mathbf{M}_1^{-1} = \mathbf{R}_1^T \hat{\mathbf{A}} \mathbf{R}_1. \tag{5.6}$$

An operator preconditioned with the lumped BDDC preconditioner has the same eigenvalues as the lumped FETI-DP preconditioned operator, with the exception of some eigenvalues that are equal to 0 and 1 [54].

The second injection operator, $\mathbf{R}_2$, the harmonic injection operator, results in the Dirichlet BDDC preconditioner, $\mathbf{M}_2^{-1}$.

This injection operator uses a discrete harmonic extension of the interface values to minimize the energy norm of the result, which leads to a better stability bound, as shown by Toselli and Widlund [81]. Let $\boldsymbol{\mathcal{H}}$ be given by a direct sum of local operators $\boldsymbol{\mathcal{H}}^e = -\mathbf{A}_{\mathrm{I,I}}^{e,-1} \mathbf{A}_{\Gamma,\mathrm{I}}^{e,T}$. This operator solves a local Dirichlet problem given by mapping the jump over subdomain interfaces for duplicated

values to the subdomain interior. The jump mapping is given by

$$\mathbf{J}^{e,T} v\left(x\right) = \sum_{i \in \mathcal{N}(x)} \left(\frac{v^e\left(x\right)}{|\mathcal{N}\left(x\right)|} - \frac{v^i\left(x\right)}{|\mathcal{N}\left(x\right)|} \right), \forall x \in \Gamma^e. \tag{5.7}$$

Definition 5.1.4 (Harmonic Injection Operator) The harmonic injection operator is given by $\mathbf{R}_2 = \mathbf{R}_1 - \mathbf{J}^T \boldsymbol{\mathcal{H}}^T$, where $\boldsymbol{\mathcal{H}}$ is given by a direct sum of local operators $\boldsymbol{\mathcal{H}}^e = -\mathbf{A}_{\mathrm{I,I}}^{e,-1} \mathbf{A}_{\Gamma,\mathrm{I}}^{e,T}$ and

$$\mathbf{J}^{e,T} v\left(x\right) = \sum_{i \in \mathcal{N}(x)} \left(\frac{v^e\left(x\right)}{|\mathcal{N}\left(x\right)|} - \frac{v^i\left(x\right)}{|\mathcal{N}\left(x\right)|} \right), \forall x \in \Gamma^e. \tag{5.8}$$

Definition 5.1.5 (Dirichlet Balanced Domain Decomposition by Constraints) Let $\mathbf{R}_2$ be given by the harmonic injection operator from Definition 5.1.4. The Dirichlet BDDC preconditioner is given by

$$\mathbf{M}_2^{-1} = \mathbf{R}_2^T \hat{\mathbf{A}} \mathbf{R}_2. \tag{5.9}$$

An operator preconditioned with the Dirichlet BDDC preconditioner, $\mathbf{M}_2^{-1}\mathbf{A}$, has the same eigenvalues as the BDDC preconditioned operator with original formulation of the BDDC preconditioner given by Dohrmann [20], with the exception of some eigenvalues that are equal to 1 [54].

5.1.2 Subdomain Solver with Fast Diagonalization

Inexact solvers for the subdomain problems $\mathbf{A}_{\mathrm{r,r}}^{-1}$ were discussed by Li and Widlund in [54]. Here we introduce an inexact subdomain solver based on the Fast Diagonalization Method (FDM).

The FDM is a fast exact solver for separable problems with a tensor product representation introduced by Lynch, Rice, and Thomas [57]. Fast diagonalization has been an effective inexact solver as part of additive overlapping Schwarz problems for spectral element discretizations for Poisson and Helmholtz equations, as well as Naiver-Stokes problems, as shown by Fischer and Lottes [30].

Let M and K be the one-dimensional mass and Laplacian matrices, respectively, given by

$$M = N^T W N, \quad K = D^T W D \tag{5.10}$$

where N, D, and W are the element interpolation, derivative, and quadrature weight matrices, as given in Section 2.3.1. Mass and Laplacian matrices in higher dimensions can be given by tensor products.

$$\begin{aligned} \mathbf{M} &= (M \otimes M \otimes M) \\ \mathbf{K} &= (K \otimes M \otimes M) + (M \otimes K \otimes M) + (M \otimes M \otimes K) \end{aligned} \tag{5.11}$$

Because M is symmetric positive definite and K is symmetric, the one-dimensional mass and Laplacian matrices can be simultaneously diagonalized. This means that there exists a non-singular matrix S and diagonal matrix Λ such that

$$SMS^T = I, \quad SKS^T = \Lambda. \tag{5.12}$$

In higher dimensions this diagonalization is given by tensor products of the one-dimensional diagonalization, and we have

$$\mathbf{M} = \mathbf{S}^{-1}\mathbf{I}\mathbf{S}^{-T}, \quad \mathbf{K} = \mathbf{S}^{-1}\mathbf{\Lambda}\mathbf{S}^{-T} \tag{5.13}$$

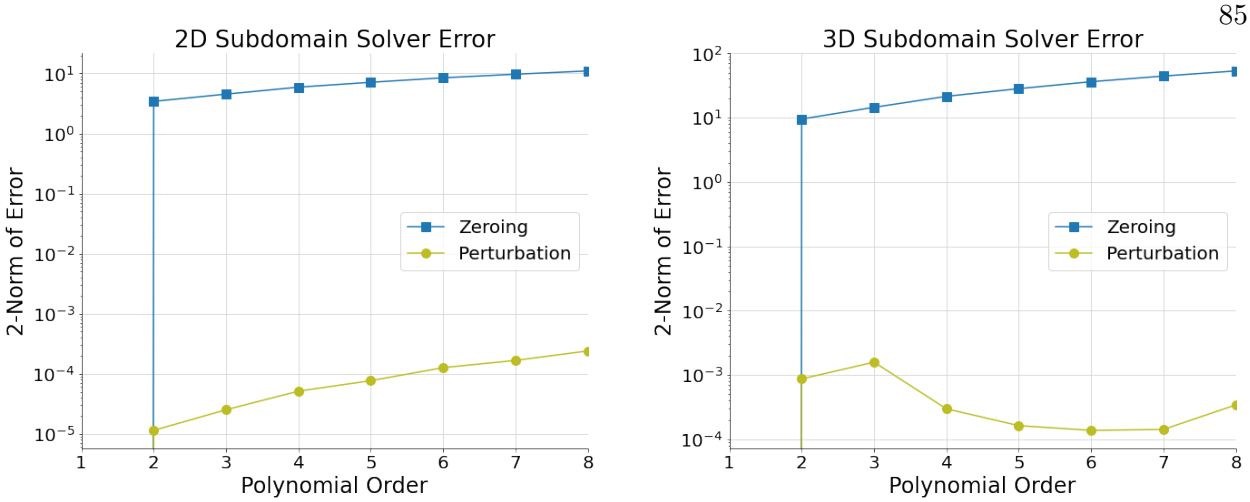
where

$$\begin{aligned} \mathbf{S} &= (S \otimes S \otimes S) \\ \mathbf{I} &= (I \otimes I \otimes I) \\ \mathbf{\Lambda} &= (\Lambda \otimes I \otimes I) + (I \otimes \Lambda \otimes I) + (I \otimes I \otimes \Lambda)\,. \end{aligned} \tag{5.14}$$

If $\mathbf{M}$ and $\mathbf{K}$ are invertible on the domain, then the inverse of $\mathbf{M}$ and $\mathbf{K}$ are given by

$$\mathbf{M}^{-1} = \mathbf{S}^T\mathbf{I}\mathbf{S}, \quad \mathbf{K}^{-1} = \mathbf{S}^T\mathbf{\Lambda}^{-1}\mathbf{S}. \tag{5.15}$$

However, for a single element with no boundary conditions, the Laplacian is singular and no inverse exists. Fischer and Lottes [30] compute the pseudo-inverse of the diagonal $\mathbf{\Lambda}^+$, which is given by taking the reciprocal of the non-zero values in $\mathbf{\Lambda}$. We instead compute an approximate inverse to the Laplacian by finding the true inverse of a perturbed Laplacian, given by $\boldsymbol{\mathcal{K}} = \mathbf{K} + \epsilon\mathbf{I}$. We denote this approximate inverse by $\mathbf{K}^+$.



(a) 1D Subdomain Error

(b) 2D Subdomain Error

Figure 5.2: Subdomain Pseudo-inverse Comparison

In Figure 5.2, we compare these two methods of forming a pesudo-inverse of the Laplacian on a reference element. We attempt to invert the action of the Laplacian, $\mathbf{Ku}$, on a two and three-dimensional reference element with polynomial bases of varying order with Dirichlet boundary conditions on the primal vertices. The values in the vector $\mathbf{u}$ are given by $f(\mathbf{x}) = -\sum_d (\mathbf{x}_d + 1)(\mathbf{x}_d - 1)$.

Figure 5.2a shows the difference in the L^2 norm of the error between forming the pseudo-inverse by taking the reciprocal of the non-zero values in the eigenvalues of the two-dimensional Laplacian and using the true inverse of the two-dimensional Laplacian with a perturbation given by $\boldsymbol{\mathcal{K}} = \mathbf{K} + 1^{-6}\mathbf{I}$. Figure 5.2b shows the difference in the L^2 norm of the error between forming the pseudo-inverse by taking the reciprocal of the non-zero values in the eigenvalues of the three-dimensional Laplacian and using the true inverse of the three-dimensional Laplacian with a perturbation given by $\boldsymbol{\mathcal{K}} = \mathbf{K} + 1^{-8}\mathbf{I}$. In both figures we see that forming the pseudo-inverse on the reference element by computing the true inverse of the perturbed Laplacian results in a more accurate subdomain inverse.

For separable problems that can be expressed as a linear combination of the mass and Lapla-

cian matrices

$$\mathbf{A} = a\mathbf{M} + b\mathbf{K} \tag{5.16}$$

the FDM can provide a fast direct solver if the problem is invertible or a fast approximate inverse if the problem is not.

$$\mathbf{A}^{-1} = \mathbf{S}^T \left(a\mathbf{I} + b\mathbf{\Lambda}\right)^{-1} \mathbf{S} \tag{5.17}$$

Note that the inverse given in Equation 5.17 is similar to the element operator form given in Equation 3.4. If we have an element operator defined by $\mathbf{A}^e = a\mathbf{M} + b\mathbf{K}$, then we can define an element inverse operator as $\mathbf{A}^{e,-1} = \mathbf{B}^T\mathbf{D}\mathbf{B}$, with $\mathbf{B} = \mathbf{S}$ and $\mathbf{D} = (a\mathbf{I} + b\mathbf{\Lambda})^{-1}$. If the operator is not invertible, such as the Laplacian on a single element subdomain, we can still compute the approximate element inverse in this form.

This discussion of the FDM has, thus far, neglected the effects of irregular geometry, varying coefficients, or more complex PDEs. These effects destroy the separability of this problem and the FDM cannot be used as fast direct solver for these problems. However, as discussed by Li and Widlund in [54], an inexact subdomain solver is adequate when BDDC is used as a preconditioner.

Couzy in [16] and Fischer, Miller, and Tofu in [31] presented a simple modification to the FDM to address irregular geometry. In their work, the Poisson problem is defined on a regular parallelepiped with average dimensions in each coordinate direction that match the finite element with the more complex geometry. Here we present a further generalization which allows us to create an approximate FDM subdomain solver, $\mathbf{A}_{\mathrm{r,r}}^{-1}$.

First we compute the simultaneous diagonalization of the mass and perturbed Laplacian matrices, as given in Equation 5.17. The eigenvectors from the diagonalization become the basis interpolation operator for our approximate FDM subdomain solver. Our goal is to provide an adequate diagonal operator $\mathbf{D}$ to create a sufficiently accurate approximate subdomain inverse.

Recall that the operator representing the application of the weak form at quadrature points, $\mathbf{D}$, is block diagonal with the action of the PDE on each quadrature point independent from the other quadrature points. We compute an average element coefficient by dividing out the quadrature

weight for each entry in $\mathbf{D}$ and averaging the resulting matrix entries.

Definition 5.1.6 (Average Element Coefficient) We denote the diagonal block for quadrature point i by $\mathbf{D}_i$ and compute an *average element coefficient* as

$$\bar{k}^e = \sum_{i \in 1,\ldots,q^d} \sum_{j,k \in 1,\ldots,m} \frac{(\mathbf{D}_i)_{j,k}}{\text{nnz}\,(\mathbf{D})\,\mathbf{W}_i} \tag{5.18}$$

where $\text{nnz}\,(\mathbf{D})$ is the number of nonzero entries in the sparse matrix representation of the diagonal operator, $\mathbf{W}_i$ is the quadrature weight for quadrature point i, and m is the number of evaluation modes for the operator, 1 for interpolated values, d for derivatives in d dimensions, and $1 + d$ for a weak form with both interpolated values and derivatives at quadrature points.

This average element coefficient is used to compute the approximate subdomain inverse diagonal

$$\mathbf{\Lambda}^e = \bar{k}^e \mathbf{\Lambda}, \quad \mathbf{\Lambda}^{e,-1} = \tfrac{1}{k^e} \mathbf{\Lambda}^{-1}, \tag{5.19}$$

where the value of $\mathbf{\Lambda}$ is given by

$$\begin{aligned} \mathbf{\Lambda}_N &= (I \otimes I \otimes I) \\ \mathbf{\Lambda}_D &= (\Lambda \otimes I \otimes I) + (I \otimes \Lambda \otimes I) + (I \otimes I \otimes \Lambda) \\ \mathbf{\Lambda}_{N,D} &= \mathbf{\Lambda}_N + \mathbf{\Lambda}_D \end{aligned} \tag{5.20}$$

based upon if the diagonal operator $\mathbf{D}$ has interpolated values, derivatives, or both, respectively, at quadrature points.

We have an approximate FDM inverse for the entire subdomain, $\mathbf{A}^{e,+} = \mathbf{B}^T \mathbf{D} \mathbf{B}$, but we need a subdomain solver for only the broken space, $\mathbf{A}_{\text{r,r}}^{-1,e}$. We form a saddle point problem to constrain the subdomain vertex nodes and form a Dirichlet problem from the full subdomain Neumann problem. The subdomain problem is therefore given by

$$\mathbf{A}_{\text{r,r}}^e = \boldsymbol{\mathcal{R}}^T \begin{bmatrix} \mathbf{A}^e & \boldsymbol{\mathcal{V}}^T \\ \boldsymbol{\mathcal{V}} & \mathbf{0} \end{bmatrix} \boldsymbol{\mathcal{R}}, \tag{5.21}$$

where $\boldsymbol{\mathcal{R}}$ is an injection operator from the broken element space to the full element augmented with duplicate primal nodes to constrain and $\boldsymbol{\mathcal{V}}$ is the primal restriction operator, which restricts to the duplicated primal vertices from the full broken space.

Definition 5.1.7 (Fast Diagonalization Method Approximate Subdomain Solver) The Fast Diagonalization Method approximate inverse subdomain solver is given by the direct sum of element subdomain solvers of the form

$$\mathbf{A}_{\mathrm{r,r}}^{e,-1} \approx \mathbf{A}_{\mathrm{r,r}}^{e,+} = \boldsymbol{\mathcal{R}}^T \begin{bmatrix} \mathbf{A}^{e,+} & -\mathbf{A}^{e,+}\boldsymbol{\mathcal{V}}^T\mathbf{S}^{-1} \\ \mathbf{0} & \mathbf{S}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\boldsymbol{\mathcal{V}}\mathbf{A}^{e,+} & \mathbf{I} \end{bmatrix} \boldsymbol{\mathcal{R}}, \tag{5.22}$$

where $\mathbf{S} = -\boldsymbol{\mathcal{V}}\mathbf{A}^{e,+}\boldsymbol{\mathcal{V}}^T$, $\boldsymbol{\mathcal{R}}$ is an injection operator from the duplicate primal vertices to the full element, and $\boldsymbol{\mathcal{V}}$ is the primal restriction operator, which restricts to the primal vertices from the full element. The approximate inverse for each subdomain is given by

$$\mathbf{A}^{e,+} = \mathbf{B}^T\mathbf{D}\mathbf{B}, \tag{5.23}$$

where $\mathbf{B}$ is a basis interpolating to the eigenspace of the simultaneous diagonalization of the mass and perturbed Laplacian matrices $\mathbf{M} = \mathbf{N}^T\mathbf{W}\mathbf{N}$ and $\boldsymbol{\mathcal{K}} = \mathbf{D}^T\mathbf{W}\mathbf{D} + \epsilon\mathbf{I}$. $\mathbf{D}$ is computed from the eigenvalues of this diagonalization and the average element coefficient value, from Definition 5.1.6,

$$\boldsymbol{\Lambda}^e = \bar{k}^e\boldsymbol{\Lambda}, \quad \boldsymbol{\Lambda}^{e,-1} = \tfrac{1}{\bar{k}^e}\boldsymbol{\Lambda}^{-1}, \tag{5.24}$$

where the value of $\boldsymbol{\Lambda}$ is given by

$$\begin{aligned} \boldsymbol{\Lambda}_N &= (I \otimes I \otimes I) \\ \boldsymbol{\Lambda}_D &= (\Lambda \otimes I \otimes I) + (I \otimes \Lambda \otimes I) + (I \otimes I \otimes \Lambda) \\ \boldsymbol{\Lambda}_{N,D} &= \boldsymbol{\Lambda}_N + \boldsymbol{\Lambda}_D \end{aligned} \tag{5.25}$$

based upon if the weak form for operator $\mathbf{A}^e$ has interpolated values, derivatives, or both, respectively, at quadrature points.

The subdomain interior solver can also be computed from this fast diagonalization of the element matrix. Reusing the fast diagonalization for both the subdomain solver and subdomain interior eliminates the higher setup cost of the Dirichlet BDDC compared to the lumped BDDC.

Definition 5.1.8 (Fast Diagonalization Method Approximate Subdomain Interior Solver) The Fast Diagonalization Method approximate inverse subdomain interior solver is given by the direct sum of element subdomain solvers of the form

$$\mathbf{A}_{\mathrm{r,r}}^{e,-1} \approx \mathbf{A}_{\mathrm{r,r}}^{e,+} = \boldsymbol{\mathcal{R}}^T \begin{bmatrix} \mathbf{A}^{e,+} & -\mathbf{A}^{e,+}\boldsymbol{\mathcal{V}}^T\mathbf{S}^{-1} \\ \mathbf{0} & \mathbf{S}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\boldsymbol{\mathcal{V}}\mathbf{A}^{e,+} & \mathbf{I} \end{bmatrix} \boldsymbol{\mathcal{R}}, \tag{5.26}$$

where $\mathbf{S} = -\boldsymbol{\mathcal{V}}\mathbf{A}^{e,+}\boldsymbol{\mathcal{V}}^T$, $\boldsymbol{\mathcal{R}}$ is an injection operator from the duplicate interface vertices to the full element, and $\boldsymbol{\mathcal{V}}$ is the interface restriction operator, which restricts to the interface vertices from the element. The approximate inverse for each subdomain is given by

$$\mathbf{A}^{e,+} = \mathbf{B}^T\mathbf{D}\mathbf{B}, \tag{5.27}$$

where $\mathbf{B}$ is a basis interpolating to the eigenspace of the simultaneous diagonalization of the mass and perturbed Laplacian matrices $\mathbf{M} = \mathbf{N}^T\mathbf{W}\mathbf{N}$ and $\boldsymbol{\mathcal{K}} = \mathbf{D}^T\mathbf{W}\mathbf{D} + \epsilon\mathbf{I}$. $\mathbf{D}$ is computed from the eigenvalues of this diagonalization and the average element coefficient value, from Definition 5.1.6,

$$\boldsymbol{\Lambda}^e = \bar{k}^e\boldsymbol{\Lambda}, \quad \boldsymbol{\Lambda}^{e,-1} = \tfrac{1}{k^e}\boldsymbol{\Lambda}^{-1}, \tag{5.28}$$

where the value of $\boldsymbol{\Lambda}$ is given by

$$\begin{aligned} \boldsymbol{\Lambda}_N &= (I \otimes I \otimes I) \\ \boldsymbol{\Lambda}_D &= (\Lambda \otimes I \otimes I) + (I \otimes \Lambda \otimes I) + (I \otimes I \otimes \Lambda) \\ \boldsymbol{\Lambda}_{N,D} &= \boldsymbol{\Lambda}_N + \boldsymbol{\Lambda}_D \end{aligned} \tag{5.29}$$

based upon if the weak form for operator $\mathbf{A}^e$ has interpolated values, derivatives, or both, respectively, at quadrature points.

5.2 Local Fourier Analysis of Balancing Domain Decomposition by Constraints

In order to compute the symbol of the BDDC preconditioners, we need to find the symbol of the injection operators and the subassembled inverse separately.

5.2.1 Subassembled Operator

The symbol of the partially subassembled problem is delicate to compute because it requires both modes in the broken element space and the global primal variable space. We will form the symbol of the three components of the partially subassembled problem, $\mathbf{A}_{r,r}^{-1}$, $\hat{\mathbf{S}}_{\Pi}^{-1}$, and $\hat{\mathbf{A}}_{\Pi,r}$, separately.

The Fourier modes on each broken element subdomain are separate, so the mode localization operator $\mathbf{Q}$ is not used for the symbol of the subdomain solver. The subdomain solver is completely in the broken element space and therefore has a very straightforward symbol, given by

$$\tilde{\mathbf{A}}_{\mathrm{r,r}}^{-1}(\boldsymbol{\theta}) = \mathbf{A}_{\mathrm{r,r}}^{-1} \odot \left[e^{\imath(\mathbf{x}_j - \mathbf{x}_i)\cdot\boldsymbol{\theta}/\mathbf{h}} \right] \tag{5.30}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are the coordinates for the nodes in the partially subassembled space.

On the other hand, the Schur complement spans the entire primal space and requires the mode localization operator for the primal space $\mathbf{Q}_{\Pi}$. The symbol of the Schur complement is given by

$$\tilde{\hat{\mathbf{S}}}_{\Pi}^{-1}(\boldsymbol{\theta}) = \left(\mathbf{Q}_{\Pi}^{T} \left(\hat{\mathbf{S}}_{\Pi} \odot \left[e^{\imath(\mathbf{x}_j - \mathbf{x}_i)\cdot\boldsymbol{\theta}/\mathbf{h}} \right] \right) \mathbf{Q}_{\Pi} \right)^{-1} \tag{5.31}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are the coordinates for the nodes in the primal space. The primal vertex solve is global across the entire problem, so the corresponding Fourier modes are mapped to the global problem before being solved exactly.

For the mixed primal and subdomain operators, we use the Fourier mode localization operator only on the modes corresponding to the primal vertices, giving us

$$\tilde{\hat{\mathbf{A}}}_{\mathrm{r},\Pi}(\boldsymbol{\theta}) = \left(\hat{\mathbf{A}}_{\mathrm{r},\Pi} \odot \left[e^{\imath(\mathbf{x}_j - \mathbf{x}_i)\cdot\boldsymbol{\theta}/\mathbf{h}} \right] \right) \mathbf{Q}_{\Pi} \tag{5.32}$$

where $\mathbf{x}_i$ come from the partially subassembled space and $\mathbf{x}_j$ come from the primal variable space.

Definition 5.2.1 (Symbol of Partially Subassembled Operator Inverse) The symbol of the inverse of the partially subassembled problem is given by

$$\tilde{\hat{\mathbf{A}}}^{-1} = \begin{bmatrix} \mathbf{I} & -\tilde{\mathbf{A}}_{\mathrm{r,r}}^{-1} \tilde{\hat{\mathbf{A}}}_{\Pi,\mathrm{r}}^{T} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{A}}_{\mathrm{r,r}}^{-1} & \mathbf{0} \\ \mathbf{0} & \tilde{\hat{\mathbf{S}}}_{\Pi}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\tilde{\mathbf{A}}_{\Pi,\mathrm{r}} \tilde{\hat{\mathbf{A}}}_{\mathrm{r,r}}^{-1} & \mathbf{I} \end{bmatrix}. \tag{5.33}$$

In this notation, $\tilde{\mathbf{A}}$ indicates Fourier modes on the broken element space, while $\tilde{\hat{\mathbf{A}}}$ indicates Fourier modes on the global primal variable space.

The symbol of the subdomain solver is given by

$$\tilde{\mathbf{A}}_{\mathrm{r,r}}^{-1}(\boldsymbol{\theta}) = \mathbf{A}_{\mathrm{r,r}}^{-1} \odot \left[e^{\imath(\mathbf{x}_j - \mathbf{x}_i)\cdot\boldsymbol{\theta}/\mathbf{h}} \right] \tag{5.34}$$

while the symbol of the Schur complement solve is given by

$$\tilde{\hat{\mathbf{S}}}_{\Pi}^{-1}(\boldsymbol{\theta}) = \left(\mathbf{Q}_{\Pi}^{T} \left(\hat{\mathbf{S}}_{\Pi} \odot \left[e^{\imath(\mathbf{x}_j - \mathbf{x}_i)\cdot\boldsymbol{\theta}/\mathbf{h}} \right] \right) \mathbf{Q}_{\Pi} \right)^{-1} \tag{5.35}$$

and the symbol of the mixed primal and subdomain operators are given by

$$\tilde{\hat{\mathbf{A}}}_{\mathrm{r},\Pi}(\boldsymbol{\theta}) = \left(\hat{\mathbf{A}}_{\mathrm{r},\Pi} \odot \left[e^{\imath(\mathbf{x}_j - \mathbf{x}_i)\cdot\boldsymbol{\theta}/\mathbf{h}} \right] \right) \mathbf{Q}_{\Pi} \tag{5.36}$$

and $\tilde{\hat{\mathbf{A}}}_{\mathrm{r},\Pi}^{T}(\boldsymbol{\theta})$ is given by the conjugate transpose. The portion of the mode localization operator on the primal nodes and modes is denoted $\mathbf{Q}_{\Pi}$. The coordinates $\mathbf{x}_i$ and $\mathbf{x}_j$ are given in the partially subassembled space or primal space, as appropriate.

5.2.2 Injection Operators

The injection operators map the global degrees of freedom to the partially subassembled spaces on each subdomain, and the symbols of the injection operators map the Fourier modes on the global space to the Fourier modes on the partially subassembled space.

The scaled injection operator, $\mathbf{R}_1$, is a pointwise scaling operator, so the operator is its own symbol. We can partition $\mathbf{R}_1$ into its action on primal and subassembled nodes on each element,

$$\mathbf{R}_1^e = \begin{bmatrix} \mathbf{R}_{1,\Pi} & \\ & \mathbf{R}_{1,r} \end{bmatrix}. \tag{5.37}$$

In the symbol of the subassembled inverse, the Fourier modes on the primal nodes have already been localized to Fourier modes in the global space, so we only need the Fourier mode localization operator on the subassembled nodes, denoted $\mathbf{Q}_r$.

Definition 5.2.2 (Symbol of Scaled Injection Operator) The symbol of the scaled injection operator is given by

$$\tilde{\mathbf{R}}_1(\boldsymbol{\theta}) = \begin{bmatrix} \mathbf{I} & \\ & \mathbf{R}_{1,r} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \\ & \mathbf{Q}_r \end{bmatrix} \mathbf{P}_\pi, \tag{5.38}$$

where $\mathbf{R}_{1,r}$ gives the portion of the scaled injection operator acting on the subassembled nodes and $\mathbf{Q}_r$ gives the portion of the Fourier mode localization operator that localizes the Fourier modes on the subassembled nodes. $\mathbf{P}_\pi$ is a permutation matrix that re-orders the Fourier modes to agree with the order given for high-order operators in Definition 3.1.2.

For the symbol of the harmonic injection operator, we need the symbol of the harmonic extension and jump mapping.

As with the partially subassembled problem, the symbol of the harmonic extension is given by the symbols of the local operators,

$$\tilde{\boldsymbol{\mathcal{H}}} = -\tilde{\mathbf{A}}_{I,I}^{-1}\tilde{\mathbf{A}}_{\Gamma,I}^{T} \tag{5.39}$$

where

$$\tilde{\mathbf{A}}_{I,I}^{-1}(\boldsymbol{\theta}) = \mathbf{A}_{I,I}^{e,-1} \odot \left[e^{\imath(\mathbf{x}_j - \mathbf{x}_i)\cdot\boldsymbol{\theta}/\mathbf{h}}\right], \quad \tilde{\mathbf{A}}_{\Gamma,I}^{T}(\boldsymbol{\theta}) = \mathbf{A}_{\Gamma,I}^{e,T} \odot \left[e^{\imath(\mathbf{x}_j - \mathbf{x}_i)\cdot\boldsymbol{\theta}/\mathbf{h}}\right]. \tag{5.40}$$

The jump mapping is a pointwise scaling operator, so it is its own symbol, just as $\mathbf{R}_1$ is its own symbol. The symbol $\tilde{\mathbf{J}}$ is a operator with entries on the diagonal given by

$$\tilde{\mathbf{J}}_{i,i} = -\frac{|\mathcal{N}(x_i)| - 1}{|\mathcal{N}(x_i)|} \tag{5.41}$$

and off diagonal entries given by

$$\tilde{\mathbf{J}}_{i,j} = -\frac{1}{|\mathcal{N}(x_i)|} \text{ if } x_j \in \mathcal{N}(x_i). \tag{5.42}$$

Definition 5.2.3 (Symbol of Harmonic Injection Operator) The symbol of the harmonic injection operator is given by

$$\tilde{\mathbf{R}}_2(\boldsymbol{\theta}) = \begin{bmatrix} \mathbf{I} & \\ & \mathbf{R}_{1,r} - \mathbf{J}_r^T\boldsymbol{\mathcal{H}}_r^T \end{bmatrix} \begin{bmatrix} \mathbf{I} & \\ & \mathbf{Q}_r \end{bmatrix} \mathbf{P}_\pi, \tag{5.43}$$

where $\mathbf{R}_{1,r}$ gives the portion of the scaled injection operator acting on the subassembled nodes, $\mathbf{J}_r^T$ gives the portion of the jump mapping acting on the subassembled nodes, $\boldsymbol{\mathcal{H}}_r^T$ gives the portion of the harmonic extension acting on subassembled nodes, and $\mathbf{Q}_r$ gives the portion of the Fourier mode localization operator that localizes the Fourier modes on the subassembled nodes. $\mathbf{P}_\pi$ is a permutation matrix that re-orders the Fourier modes to agree with the order given for high-order operators in Definition 3.1.2.

The symbol of the harmonic extension is given by the symbols of the local operators,

$$\tilde{\boldsymbol{\mathcal{H}}} = -\tilde{\mathbf{A}}_{I,I}^{-1}\tilde{\mathbf{A}}_{\Gamma,I}^{T}, \tag{5.44}$$

where

$$\tilde{\mathbf{A}}_{I,I}^{-1}(\boldsymbol{\theta}) = \mathbf{A}_{I,I}^{e,-1} \odot \left[e^{\imath(\mathbf{x}_j-\mathbf{x}_i)\cdot\boldsymbol{\theta}/\mathbf{h}}\right], \quad \tilde{\mathbf{A}}_{\Gamma,I}^{T}(\boldsymbol{\theta}) = \mathbf{A}_{\Gamma,I}^{e,T} \odot \left[e^{\imath(\mathbf{x}_j-\mathbf{x}_i)\cdot\boldsymbol{\theta}/\mathbf{h}}\right]. \tag{5.45}$$

The coordinates $\mathbf{x}_i$ and $\mathbf{x}_j$ are given in the subdomain interior or on subdomain interface, as appropriate.

The symbol of the jump mapping, $\tilde{\mathbf{J}}$, is a operator with entries on the diagonal given by

$$\tilde{\mathbf{J}}_{i,i} = -\frac{|\mathcal{N}(x_i)|-1}{|\mathcal{N}(x_i)|} \tag{5.46}$$

and off diagonal entries given by

$$\tilde{\mathbf{J}}_{i,j} = -\frac{1}{|\mathcal{N}(x_i)|} \text{ if } x_j \in \mathcal{N}(x_i). \tag{5.47}$$

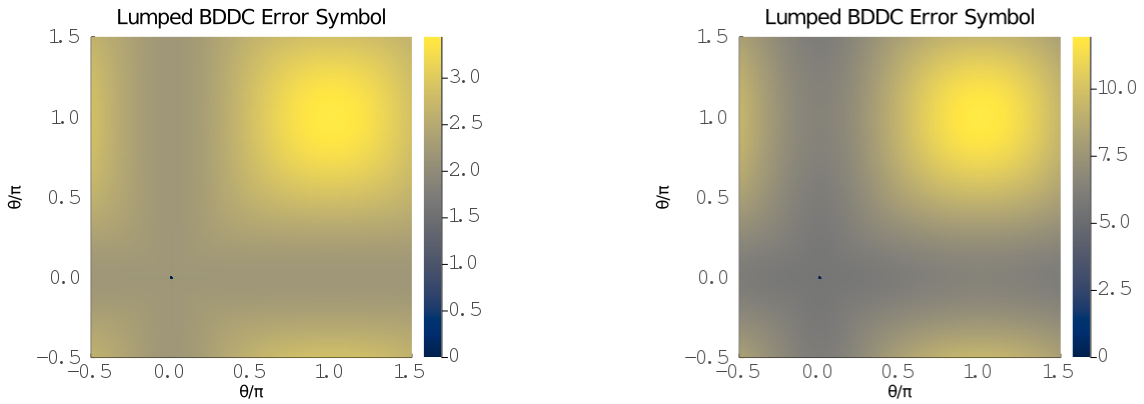
5.2.3 Balancing Domain Decomposition by Constraints Symbol

The symbol of the lumped and Dirichlet variants of BDDC are given by the symbol of the subassembled inverse and the respective injection operators.

Definition 5.2.4 (Symbol of Lumped Balancing Domain Decomposition by Constraints) The symbol of the lumped BDDC preconditioner is given by

$$\tilde{\mathbf{M}}_1^{-1} = \tilde{\mathbf{R}}_1^T \tilde{\hat{\mathbf{A}}}^{-1} \tilde{\mathbf{R}}_1 \tag{5.48}$$

where $\tilde{\mathbf{R}}_1$ is the symbol of the scaled injection operator, given by Definition 5.2.2, and $\tilde{\hat{\mathbf{A}}}^{-1}$ is the symbol of the inverse of the subassembled problem, given by Definition 5.2.1.



(a) Low-Order Lumped BDDC

(b) High-Order Lumped BDDC

Figure 5.3: Lumped BDDC Error Symbol for the 2D Laplacian

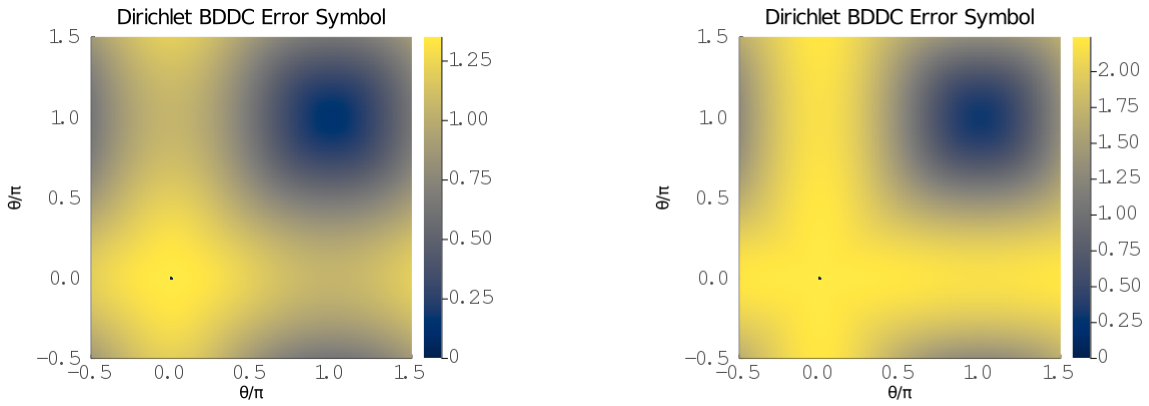
In Figure 5.3a we see the spectral radius of the symbol of the error operator of lumped BDDC for the scalar diffusion operator on a subdomain consisting of 16 linear H^1 Lagrange finite elements in two dimensions. In Figure 5.3b we see the spectral radius of the symbol of the error operator of lumped BDDC for the scalar diffusion operator on a subdomain consisting of a single finite element with a 4th order H^1 Lagrange basis in two dimensions. We can see that lumped BDDC is significantly less effective on the subdomain consisting of a single high-order element.

Definition 5.2.5 (Symbol of Dirichlet Balancing Domain Decomposition by Constraints) The symbol of the Dirichlet BDDC preconditioner is given by

$$\tilde{\mathbf{M}}_2^{-1} = \tilde{\mathbf{R}}_2^T \tilde{\hat{\mathbf{A}}}^{-1} \tilde{\mathbf{R}}_2 \tag{5.49}$$

where $\tilde{\mathbf{R}}_2$ is the symbol of the harmonic injection operator, given by Definition 5.2.3, and $\tilde{\hat{\mathbf{A}}}^{-1}$ is the symbol of the inverse of the subassembled problem, given by Definition 5.2.1.

In Figure 5.4a we see the spectral radius of the symbol of the error operator of Dirichlet BDDC for the scalar diffusion operator on a subdomain consisting of 16 linear H^1 Lagrange finite



(a) Low-Order Dirichlet BDDC

(b) High-Order Dirichlet BDDC

Figure 5.4: Dirichlet BDDC Error Symbol for the 2D Laplacian

elements in two dimensions. In Figure 5.4b we see the spectral radius of the symbol of the error operator of Dirichlet BDDC for the scalar diffusion operator on a subdomain consisting of a single finite element with a 4th order H^1 Lagrange basis in two dimensions. We can see that Dirichlet BDDC is less effective on the subdomain consisting of a single high-order element, but the effect is not as pronounced as with lumped BDDC.

5.2.4 Low-Order Macro-Element Validation

Previous work by Brown, He, and MacLachlan [13] provided LFA of lumped and Dirichlet BDDC on subdomains consisting of multiple low-order finite elements. This previous work provided very sharp LFA convergence estimates when compared to numerical experiments with PETSc [6] on a periodic mesh.

Table 5.1 shows the maximal eigenvalues and condition numbers for the symbol of the BDDC preconditioned operator $\tilde{\mathbf{M}}_i^{-1}(\boldsymbol{\theta})\,\tilde{\mathbf{A}}(\boldsymbol{\theta})$ for the two-dimensional Poisson problem on linear macro-element subdomains with varying numbers of elements in one dimension. The maximal eigenvalues were computed across 64 uniformly spaced frequencies. These values exactly agree with the previous work by Brown, He, and MacLachlan [13].

It is important to note the significantly improved condition number and resulting improved

m	Lumped BDDC			Dirichlet BDDC		
	$\lambda_{\min}$	$\lambda_{\max}$	κ	$\lambda_{\min}$	$\lambda_{\max}$	κ
$m = 4$	1.000	4.444	4.444	1.000	2.351	2.351
$m = 8$	1.000	12.269	12.269	1.000	3.196	3.196
$m = 16$	1.000	31.179	31.179	1.000	4.188	4.188
$m = 32$	1.000	75.761	75.761	1.000	5.335	5.335

Table 5.1: Condition Numbers and Maximal Eigenvalues for BDDC with Low-Order Macro-Elements

convergence for the Dirichlet BDDC. Lumped BDDC has approximately half of the setup costs when compared to Dirichlet BDDC when using assembled exact inverses. Brown, He, and MachLachlan investigated the use of multiplicative combinations of the lumped BDDC operator with a diagonal scaling operator to improve convergence while still retaining the benefit of cheaper setup for lumped BDDC. This technique helped mitigate the growth of the condition number for the lumped BDDC as the subdomain size grew, but the resulting condition number was still larger than the condition number for Dirichlet BDDC.

5.2.5 High-Order Single Element Numerical Results

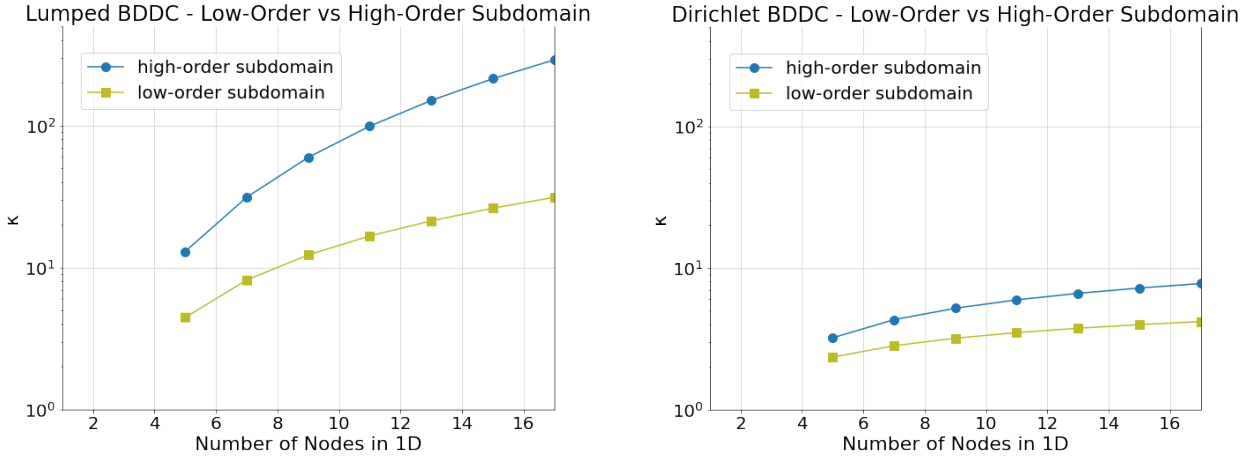
Single high-order element subdomains can be used in place of subdomains consisting of multiple low-order elements. This approach has been investigated for solid mechanics problems, such as in [68]; although, this work incorporated additional primal degrees of freedom from average values across subdomain edges and faces.

p	Lumped BDDC			Dirichlet BDDC		
	$\lambda_{\min}$	$\lambda_{\max}$	κ	$\lambda_{\min}$	$\lambda_{\max}$	κ
$p = 2$	1.000	2.800	2.800	1.000	2.042	2.042
$p = 4$	1.000	12.948	12.948	1.000	3.242	3.242
$p = 8$	1.000	59.563	59.563	1.000	5.197	5.197
$p = 16$	1.000	289.678	289.678	1.000	7.761	7.761

Table 5.2: Condition Numbers and Maximal Eigenvalues for BDDC with Single High-Order Element Subdomains

Table 5.2 shows the maximal eigenvalues and condition numbers for the symbol of the BDDC preconditioned operator $\tilde{\mathbf{M}}_i^{-1}(\boldsymbol{\theta})\,\tilde{\mathbf{A}}(\boldsymbol{\theta})$ for the two-dimensional Poisson problem on single high-order element subdomains with varying polynomial orders.

The improved condition number, and therefore convergence, of the Dirichlet BDDC compared to the lumped BDDC is more significant for high-order elements. With single high-order element subdomains, the Fast Diagonalization Method approximate inverse subdomain solver can be used for both the subdomain subassembled inverse and subdomain interior inverse, which eliminates the advantage in reduced setup costs for lumped BDDC.



(a) Lumped BDDC

(b) Dirichlet BDDC

Figure 5.5: Low-Order and High-Order Subdomain BDDC Condition Number Comparison

In Figure 5.5a we compare the growth of the condition number for the lumped BDDC preconditioned operator for a subdomain consisting of several linear elements and a subdomain consisting of a single high-order element for the two-dimensional Poisson problem. The condition number grows much more rapidly for a high-order subdomain with an equivalent number of degrees of freedom.

In Figure 5.5b we compare the growth of the condition number for the Dirichlet BDDC preconditioned operator for a subdomain consisting of several linear elements and a subdomain consisting of a single high-order element for the two-dimensional Poisson problem. In contrast with

Figure 5.5a, we see that the condition number of the preconditioned operator on the high-order subdomain grows only slightly faster than the condition number of the low-order subdomain.

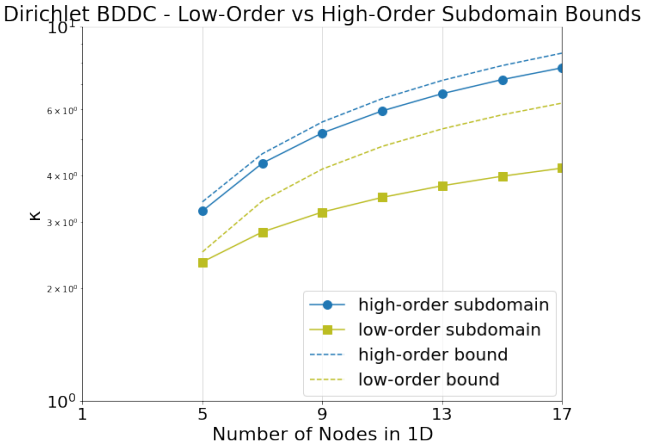


Figure 5.6: Low-Order and High-Order Subdomain Dirichlet BDDC Condition Number Bounds

The condition numbers of the Dirichlet BDDC preconditoner operator is bounded by

$$\kappa \leq \mathcal{C}\left(1+\log\left(p^2\frac{H}{h}\right)\right)^2 \tag{5.50}$$

where $\mathcal{C} > 0$ is independent of the polynomial order p, the element size h, and the subdomain size H [45]. In Figure 5.6, we can see that our LFA-predicted condition numbers closely agree with this bound.

Klawonn, Pavarino, and Rheinbach investigated the performance of BDDC and FETI-DP preconditioners for the two-dimensional Poisson problem with discontinuous coefficients [45] and provided some experimentally derived maximal eigenvalues and condition numbers for the two-dimensional Poisson problem.

In Table 5.3 we see the LFA-predicted maximal eigenvalues and condition numbers for the scalar Poisson problem in two dimensions with constant coefficients compared to numerical results on a domain with 576 subdomains consisting of a single high-order finite element. The LFA-predicted condition numbers generally agree with the experimental results, typically providing a mildly pessimistic prediction when compared to the experimental results.

In Table 5.3 we see the LFA-predicted maximal eigenvalues and condition numbers for the

p	LFA			Experimental Results		
	$\lambda_{\min}$	$\lambda_{\max}$	κ	$\lambda_{\min}$	$\lambda_{\max}$	κ
$p=2$	1.000	2.042	2.042	1.001	1.66	1.66
$p=3$	1.000	2.614	2.614	1.000	2.38	2.38
$p=4$	1.000	3.241	3.241	1.001	3.03	3.03
$p=8$	1.000	5.195	5.194	1.001	5.01	5.00
$p=16$	1.000	7.756	7.756	1.001	7.62	7.61
$p=32$	1.000	10.961	10.961	1.002	10.86	10.84

Table 5.3: Condition Numbers and Maximal Eigenvalues for Dirichlet BDDC with Single Element High-Order Subdomains

p	LFA			Experimental Results		
	$\lambda_{\min}$	$\lambda_{\max}$	κ	$\lambda_{\min}$	$\lambda_{\max}$	κ
$p=2$	1.000	2.700	2.700	1.001	2.33	2.33
$p=3$	1.000	3.568	3.568	1.001	3.21	3.21
$p=4$	1.000	4.305	4.305	1.002	4.00	3.99
$p=8$	1.000	6.511	6.511	1.003	6.26	6.24
$p=16$	1.000	9.354	9.354	1.002	9.16	9.14
$p=32$	1.000	12.856	12.856	1.002	12.69	12.66

Table 5.4: Condition Numbers and Maximal Eigenvalues for Dirichlet BDDC 4 Element High-Order Subdomains

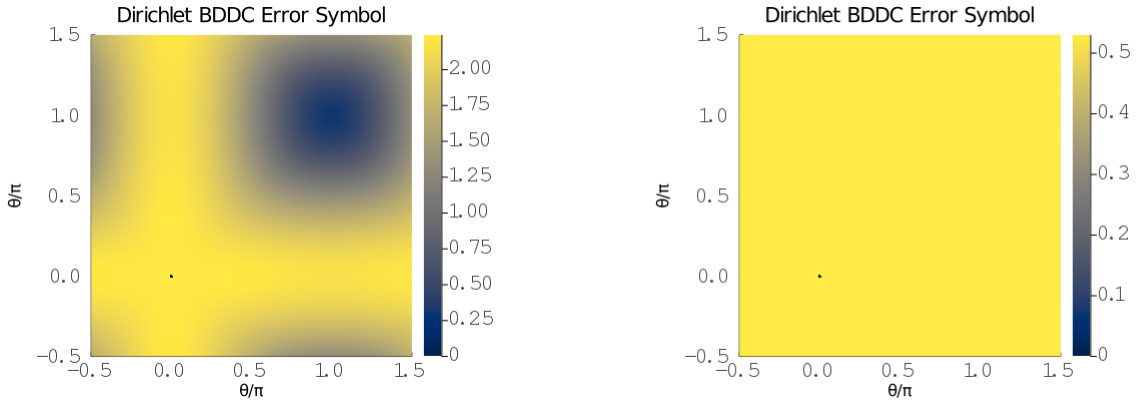
scalar Poisson problem in two dimensions with constant coefficients compared to numerical results on a domain with 576 subdomains consisting of four high-order finite elements.

5.3 Balancing Domain Decomposition by Constraints Smoother for *P*-Multigrid

Jacobi and Chebyshev smoothers provided rather poor convergence factors for p-multigrid with aggressive coarsening. In this section, we explore using BDDC on single high-order element subdomains as a smoother for p-multigrid.

Given the ill-conditioning seen in Figure 5.5a for the lumped BDDC preconditioned operator with single high-order element subdomains, we restrict our analysis to Dirichlet BDDC. As mentioned in Section 5.1.2, the higher setup cost associated with Dirichlet BDDC for subdomains

consisting of multiple low-order elements can be eliminated with the use of FDM approximate solvers for the broken subdomain and subdomain interior problems.



(a) Lumped BDDC

(b) Dirichlet BDDC

Figure 5.7: Symbol of Error Operator for Dirichlet BDDC of the 2D Laplacian for $p = 4$

In Figure 5.7a we see the symbol of the error operator for Dirichlet BDDC, given by

$$\tilde{\mathbf{E}}(\boldsymbol{\theta}) = \mathbf{I} - \tilde{\mathbf{M}}_2^{-1}\tilde{A}(\boldsymbol{\theta}), \tag{5.51}$$

for the two-dimensional Laplacian. The error operator's symbol has a spectral radius greater than one, indicating that the Dirichlet BDDC preconditioner as described above may not be an appropriate smoother for p-multigrid.

However, if we introduce a relaxation parameter, ω, as used in the Jacobi preconditioner, then we see an improved spectral radius in Figure 5.7b for the symbol of the error operator. The symbol of the error operator with a relaxation parameter is given by

$$\tilde{\mathbf{E}}(\boldsymbol{\theta}, \omega) = \mathbf{I} - \omega\tilde{\mathbf{M}}_2^{-1}\tilde{A}(\boldsymbol{\theta}). \tag{5.52}$$

In Table 5.5, we see the two-grid convergence factor for p-multigrid with the weighted Dirichlet BDDC smoother for the two-dimensional Laplacian. When compared to Table 4.5, we can see that the weighted Dirichlet BDDC smoother provides a significantly improved two-grid convergence factor for aggressive coarsening with high-order elements when compared to fourth-order Chebyshev smoothing. Additionally, the weighted Dirichlet BDDC smoother provides analogous two-grid

p_{fine} to p_{coarse}	$\rho_{\min}$	ω_{opt}
$p = 2$ to $p = 1$	0.121	0.66
$p = 4$ to $p = 2$	0.272	0.48
$p = 4$ to $p = 1$	0.281	0.47
$p = 8$ to $p = 4$	0.409	0.38
$p = 8$ to $p = 2$	0.462	0.33
$p = 8$ to $p = 1$	0.462	0.32
$p = 16$ to $p = 8$	0.504	0.32
$p = 16$ to $p = 4$	0.579	0.25
$p = 16$ to $p = 2$	0.597	0.23
$p = 16$ to $p = 1$	0.597	0.23

Table 5.5: Two-Grid Convergence Factor for P-Multigrid with Dirichlet BDDC Smoother for the 2D Laplacian

convergence factors when compared to more conservative coarsening strategies with second or third-order Chebyshev smoothing.

Even when Dirichlet BDDC smoothing provides analogous convergence factors to third or fourth-order Chebyshev smoothing, the Dirichlet BDDC smoother offers benefits in global communication requirements. A kth-order Chebyshev smoother requires k full operator evaluations, each of which requires global communication. In contrast, the application of the Dirichlet BDDC smoother requires two applications of the subassembled operator on the primal space, one solve of the Schur complement on the primal space, and the global communication from the transpose injection operation. The primal space is significantly smaller than the global problem, which means that much less communication is required to apply the Dirichlet BDDC smoother compared to a third or fourth-order Chebyshev smoother.

In Table 5.6, we see the two-grid convergence factor for p-multigrid with the weighted Dirichlet BDDC smoother for the three-dimensional Laplacian. As anticipated, a primal space consisting only of element vertices on the corners in three dimensions is inadequate for fourth-order or higher elements. Technically, a Richardson iteration with fourth-order or eight-order elements with the weighting factors given above will converge, but as seen in Table 3.1, this iteration will converge

p_{fine} to p_{coarse}	$\rho_{\min}$	ω_{opt}
$p = 2$ to $p = 1$	0.601	0.25
$p = 3$ to $p = 2$	0.723	0.15
$p = 3$ to $p = 1$	0.723	0.15
$p = 4$ to $p = 3$	0.903	0.05
$p = 4$ to $p = 2$	0.903	0.05
$p = 4$ to $p = 1$	0.903	0.05
$p = 8$ to $p = 4$	0.980	0.01
$p = 8$ to $p = 2$	0.980	0.01
$p = 8$ to $p = 1$	0.980	0.01

Table 5.6: Two-Grid Convergence Factor for P-Multigrid with Dirichlet BDDC Smoother for the 3D Laplacian

very slowly. However, a vertex only primal space is still sufficient for cubic and quadratic elements.

p_{fine} to p_{coarse}	$\rho_{\min}$	ω_{opt}
$p = 2$ to $p = 1$	0.785	0.12
$p = 3$ to $p = 2$	0.875	0.07
$p = 3$ to $p = 1$	0.884	0.06
$p = 4$ to $p = 3$	0.960	0.02
$p = 4$ to $p = 2$	0.960	0.02
$p = 4$ to $p = 1$	0.960	0.02

Table 5.7: Two-Grid Convergence Factor for P-Multigrid with Dirichlet BDDC Smoother for 3D Linear Elasticity

In Table 5.7, we see the two-grid convergence factor for p-multigrid with the weighted Dirichlet BDDC smoother for the three-dimensional linear elasticity problem. The linear elasticity problem is more complex, and we see a decreased effectiveness in the preconditioner. For practical problems, an enriched primal space is even more important for proper convergence.

Chapter 6

Conclusions

In this chapter, we provide a brief overview of the conclusions from this dissertation and provide areas for future work.

6.1 Summary of Conclusions

High-order matrix-free finite element operators offer superior performance on modern high performance computing hardware when compared to assembled sparse matrices, both with respect to floating point operations needed for operator evaluation and the memory transfer needed for a matrix-vector product. However, high-order matrix-free operators require iterative solvers, such as Krylov subspace methods, and these methods converge slowly for ill-conditioned operators, such as high-order finite element operators. Preconditioning techniques can significantly improve the converge of these iterative solvers for high-order matrix-free finite element operators. In particular, p-multigrid and domain decomposition methods are particularly well suited for problems on unstructured meshes, but these methods may have parameters that require careful tuning to ensure proper convergence. Local Fourier Analysis (LFA) of these preconditioners analyzes the frequency modes found in the error on each iteration and can provide sharp convergence estimates.

In Chapter 2, we presented a representation of arbitrary second-order partial differential equations for high-order finite element discretizations that facilitates matrix-free implementation. This representation is used by the Center for Efficient Exascale Discretizations to provide performance portable high-order matrix-free implementations of finite element operators for arbitrary

second-order partial differential equations. We presented the benefits of high-order matrix-free finite element discretizations and provided some examples of mini-applications using these formulations.

In Chapter 3, we developed LFA of finite element operators represented in this fashion. This framework provides sharp convergence factor estimates for preconditioners for high-order matrix-free finite element operators. We provided examples of preconditioning Jacobi and Chebyshev semi-iterative method smoothers, which are commonly used in multigrid methods.

In Chapter 4, this LFA was expanded to create the LFA of p-multigrid for Continuous Galerkin discretizations. This LFA of p-multigrid was validated with numerical experiments. Furthermore, we extended this LFA to reproduce previous work with h-multigrid by using macro-elements consisting of multiple low-order finite elements.

In Chapter 5, we also developed the LFA of lumped and Dirichlet versions of Balancing Domain Decomposition by Constraints (BDDC) preconditioners. With macro-elements consisting of multiple low-order finite elements, we can exactly reproduce previous work on the LFA of BDDC. By using Fast Diagonalization Method (FDM) approximate subdomain solvers, the increased setup costs for the Dirichlet BDDC preconditioner, relative to the lumped variant, can be substantially reduced, which makes BDDC an attractive preconditioner.

The performance of p-multigrid in parallel is partially determined by the number of multigrid levels. A larger number of multigrid levels requires additional neighbor communication on parallel machines, which is a bottleneck on modern HPC hardware. Furthermore, these intermediate representations are less computationally efficient than the fine grid representation. However, aggressive coarsening is not supported by traditional polynomial smoothers; aggressive coarsening leaves the smoother responsible for wide range of error frequency modes, which causes degraded smoothing unless a very high order smoother is used. Dirichlet BDDC can be used as a smoother for p-multigrid to more efficiently target error mode frequencies than polynomial smoothers such as the Chebyshev semi-iterative method, which facilitates more aggressive coarsening. We provided LFA of p-multigrid with Dirichlet BDDC smoothing to demonstrate the suitability of this combination for preconditioning high-order matrix-free finite element discretizations.

6.2 Future Work

This framework for LFA of arbitrary order finite element discretizations has several opportunities for continued research.

The work in this dissertation focused on LFA of nodal finite element discretizations where all fields used the same discretization. One natural extension of this work is the LFA of PDEs discretized with mixed finite element methods, such as those seen in solid mechanics applications with a continuous displacement space and a discontinuous pressure space. Another natural extension is the LFA of PDE with modal or hierarchical bases, where the basis is not associated with nodal locations.

We described the LFA of p-multigrid and h-multigrid methods. This framework can also analyze hp-multigrid methods, where the prolongation operator is again defined by the evaluation of the coarse grid basis on the fine grid macro-element space. Exploration of these techniques with LFA is an area for future work.

Currently, our LFA of BDDC is restricted to primal spaces only consisting of subdomain vertices, but it is common to augment these points with subdomain edge and face averages, especially in three dimensional problems. Extending the LFA to use richer primal spaces would allow for more general and practical analysis.

Another popular family of domain decomposition method is Schwartz methods, such as additive Schwartz. Developing the LFA of Schwartz methods, or other overlapping domain decomposition techniques, would help quantify convergence differences between overlapping methods and BDDC and facilitate a broader comparison between these two types of methods in terms of both convergence rates and global communication required.

This LFA framework is designed to support applications. The results from the LFA in this dissertation indicate two new areas of note for application development.

First, this LFA indicates that using matrix-free Dirichlet BDDC with single high-order element subdomains is an attractive preconditioner. Single high-order element subdomains can use

the FDM to provide high-order matrix-free subdomain solvers. Furthermore, using a FDM based subdomain solver can eliminate the increased setup time for Dirichlet BDDC compared to lumped BDDC. A fully matrix-free BDDC implementation using libCEED and PETSc is in development and will be a novel implementation.

Secondly, this LFA indicates that BDDC has better smoothing properties for p-multigrid than Jacobi preconditioned Chebyshev iterations. This improved smoothing also requires reduced communication, which makes BDDC a very attractive smoother for p-multigrid. BDDC smoothing for p-multigrid is also completely novel and development of an example will follow a fully matrix-free BDDC implementation using libCEED and PETSc.

Bibliography

[1] Ahmad Abdelfattah, Valeria Barra, Natalie Beams, Jed Brown, Jean-Sylvain Camier, Veselin Dobrev, Yohann Dudouit, Leila Ghaffari, Tzanio Kolev, David Medina, Thilina Rathnayake, Jeremy L Thompson, and Stanimire Tomov. libCEED User Manual, July 2021.

[2] Mark Adams, Marian Brezina, Jonathan Hu, and Ray Tuminaro. Parallel multigrid smoothing: polynomial versus Gauss-Seidel. Journal of Computational Physics, 188(2):593–610, 2003.

[3] Mark Adams, Jed Brown, John Shalf, Brian Straalen, Erich Strohmaier, and Samuel Williams. HPGMG 1.0: A benchmark for ranking high performance computing systems. LBNL Technical Report, 05 2014.

[4] Arash Mehraban, Jed Brown, Valeria Barra, Henry Tufo, Jeremy Thompson, and Richard Regueiro. Efficient residual and matrix-free Jacobian evaluation for three-dimensional triquadratic hexahedral finite elements with nearly-incompressible Neo-Hookean hyperelasticity applied to soft materials on unstructured meshes in parallel, with PETSc and libCEED. In Procedings of the 2020 International Mechanical Engineering Congress and Exposition, July 2020.

[5] Ivo Babuska and Barna Szabo. On the rates of convergence of the finite element method. International Journal for Numerical Methods in Engineering, 18(3):323–341, 1982.

[6] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William D. Gropp, Dmitry Karpeyev, Dinesh Kaushik, Matthew G. Knepley, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Karl Rupp, Patrick Sanan, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.15, Argonne National Laboratory, 2021.

[7] Achi Brandt. Multi-level adaptive solutions to boundary-value problems. Mathematics of computation, 31(138):333–390, 1977.

[8] Achi Brandt. Guide to multigrid development. In Multigrid methods, pages 220–312. Springer, 1982.

[9] James Brannick, Xiaozhe Hu, Carmen Rodrigo, and Ludmil Zikatanov. Local Fourier analysis of multigrid methods with polynomial smoothers and aggressive coarsening. Numerical Mathematics: Theory, Methods and Applications, 8(1):1–21, 2015.

[10] William L Briggs, Van Emden Henson, and Steve F McCormick. A multigrid tutorial. SIAM, 2000.

[11] Jed Brown. Efficient nonlinear solvers for nodal high-order finite elements in 3D. Journal of Scientific Computing, 45(1-3):48–63, 2010.

[12] Jed Brown, Ahmad Abdelfattah, Valeria Barra, Natalie Beams, Jean Sylvain Camier, Veselin Dobrev, Yohann Dudouit, Leila Ghaffari, Tzanio Kolev, David Medina, Will Pazner, Thilina Ratnayaka, Jeremy Thompson, and Stan Tomov. libCEED: Fast algebra for high-order element-based discretizations. Journal of Open Source Software, 6(63):2945, 2021.

[13] Jed Brown, Yunhui He, and Scott MacLachlan. Local Fourier analysis of balancing domain decomposition by constraints algorithms. SIAM Journal on Scientific Computing, 41(5):S346–S369, 2019.

[14] Jed Brown, Yunhui He, Scott MacLachlan, Matt Menickelly, and Stefan M. Wild. Tuning multigrid methods with robust optimization and local Fourier analysis. SIAM Journal of Scientific Computing, 43:A109–A138, 2021, 2021.

[15] CEED bake-off problems (benchmarks). `https://ceed.exascaleproject.org/bps/`. Accessed: 2021-06-06.

[16] Wouter Couzy. Spectral element discretization of the unsteady Navier-Stokes equations and its iterative solution on parallel computers. Technical report, EPFL, 1995.

[17] Denis Davydov, Jean-Paul Pelteret, Daniel Arndt, Martin Kronbichler, and Paul Steinmann. A matrix-free approach for finite-strain hyperelastic problems using geometric multigrid. International Journal for Numerical Methods in Engineering, 2020.

[18] Leszek Demkowicz, J Tinsely Oden, Waldemar Rachowicz, and O Hardy. Toward a universal hp adaptive finite element strategy, part 1. constrained approximation and data structure. Computer Methods in Applied Mechanics and Engineering, 77(1-2):79–112, 1989.

[19] Michel O Deville, Paul F Fischer, and Ernest H Mund. High-order methods for incompressible fluid flow. Cambridge University Press, 2002.

[20] Clark R Dohrmann. A preconditioner for substructuring based on constrained energy minimization. SIAM Journal on Scientific Computing, 25(1):246–258, 2003.

[21] Clark R Dohrmann. An approximate BDDC preconditioner. Numerical Linear Algebra with Applications, 14(2):149–168, 2007.

[22] Jack Dongarra, Michael A Heroux, and Piotr Luszczek. High-performance conjugate-gradient benchmark. International Journal of High Performance Computing Applications, 30(1):3–10, 2016.

[23] Maksymilian Dryja. An additive Schwarz algorithm for two-and three-dimensional finite element elliptic problems. Domain decomposition methods, pages 168–172, 1989.

[24] Maksymilian Dryja and Olof B Widlund. Towards a unified theory of domain decomposition algorithms for elliptic problems. New York University, Courant Institute of Mathematical Sciences, Division of Computer Science, 1989.

[25] Howard Elman, Victoria E Howle, John Shadid, Robert Shuttleworth, and Ray Tuminaro. A taxonomy and comparison of parallel block multi-level preconditioners for the incompressible Navier-Stokes equations. Journal of Computational Physics, 227(3):1790–1808, 2008.

[26] Charbel Farhat, Michael Lesoinne, and Kendall Pierson. A scalable dual-primal domain decomposition method. Numerical linear algebra with applications, 7(7-8):687–714, 2000.

[27] Charbel Farhat and Francois-Xavier Roux. A method of finite element tearing and interconnecting and its parallel solution algorithm. International Journal for Numerical Methods in Engineering, 32(6):1205–1227, 1991.

[28] Paul Fischer, Misun Min, Thilina Rathnayake, Som Dutta, Tzanio Kolev, Veselin Dobrev, Jean-Sylvain Camier, Martin Kronbichler, Tim Warburton, Kasia Swirydowicz, and Jed Brown. Scalability of high-performance PDE solvers. International Journal of High Performance Computing Applications, 34(5), 6 2020.

[29] Paul F Fischer. An overlapping Schwarz method for spectral element solution of the incompressible Navier-Stokes equations. Journal of Computational Physics, 133(1):84–101, 1997.

[30] Paul F Fischer and James W Lottes. Hybrid Schwarz-multigrid methods for the spectral element method: Extensions to Navier-Stokes. In Domain Decomposition Methods in Science and Engineering, pages 35–49. Springer, 2005.

[31] Paul F Fischer, Henry M Tufo, and NI Miller. An overlapping Schwarz method for spectral element simulation of three-dimensional incompressible flows. In Parallel Solution of Partial Differential Equations, pages 159–180. Springer, 2000.

[32] Yannis Fragakis and Manolis Papadrakakis. The mosaic of high performance domain decomposition methods for structural mechanics: Formulation, interrelation and numerical efficiency of primal and dual methods. Computer methods in applied mechanics and engineering, 192(35-36):3799–3830, 2003.

[33] F. X. Giraldo, M. Restelli, and M. Läuter. Semi-implicit formulations of the Navier-Stokes equations: Application to nonhydrostatic atmospheric modeling. SIAM Journal on Scientific Computing, 32(6):3394–3425, 2010.

[34] G. H. Golub and Van Loan C. F. Matrix Computations. Johns Hopkins University Press, 2nd edition, 1989.

[35] Gene H Golub and Richard S Varga. Chebyshev semi-iterative methods, successive over-relaxation iterative methods, and second order Richardson iterative methods. Numerische Mathematik, 3(1):157–168, 1961.

[36] B Guo and I Babuška. The hp version of the finite element method. Computational Mechanics, 1(1):21–41, 1986.

[37] Martin H. Gutknecht and Stefan Röllin. The Chebyshev iteration revisited. Parallel Computing, 28(2):263–283, 2002.

[38] Yunhui He and Scott MacLachlan. Two-level Fourier analysis of multigrid for higher-order finite-element discretizations of the laplacian. Numerical Linear Algebra with Applications, 27(3):e2285, 2020.

[39] Magnus Rudolph Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems, volume 49. NBS Washington, DC, 1952.

[40] JJ Heys, TA Manteuffel, Steve F McCormick, and LN Olson. Algebraic multigrid for higher-order finite elements. Journal of computational Physics, 204(2):520–532, 2005.

[41] Gerhard Holzapfel. Nonlinear solid mechanics: a continuum approach for engineering. Wiley, Chichester New York, 2000.

[42] Ning Hu, Xian-Zhong Guo, and I Katz. Bounds for eigenvalues and condition numbers in the p-version of the finite element method. Mathematics of computation, 67(224):1423–1450, 1998.

[43] Thomas JR Hughes. The finite element method: linear static and dynamic finite element analysis. Courier Corporation, 2012.

[44] Karsten Kahl and Nils Kintscher. Automated local Fourier analysis (aLFA). BIT Numerical Mathematics, 60(3):651–686, 2020.

[45] Axel Klawonn, Luca F. Pavarino, and Oliver Rheinbach. Spectral element FETI-DP and BDDC preconditioners with multi-element subdomains. Computer Methods in Applied Mechanics and Engineering, 198(3):511–523, 2008.

[46] Axel Klawonn, Oliver Rheinbach, and Luca F Pavarino. Exact and inexact FETI-DP methods for spectral elements in two dimensions. In Domain decomposition methods in science and engineering XVII, pages 279–286. Springer, 2008.

[47] Axel Klawonn and Olof Widlund. FETI and Neumann-Neumann iterative substructuring methods: connections and new results. Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences, 54(1):57–90, 2001.

[48] Dana A Knoll and David E Keyes. Jacobian-free Newton–Krylov methods: a survey of approaches and applications. Journal of Computational Physics, 193(2):357–397, 2004.

[49] Tzanio Kolev, Paul Fischer, Misun Min, Jack Dongarra, Jed Brown, Veselin Dobrev, Tim Warburton, Stanimire Tomov, Mark Shephard, Ahmad Abdelfattah, Valeria Barra, Natalie Beams, Jean-Sylvain Camier, Noel Chalmers, Yohann Dudouit, Ali Karakus, Ian Karlin, Stefan Kerkemeier, Yu-Hsiang Lan, and Vladimir Tomov. Efficient exascale discretizations: High-order finite element methods. The International Journal of High Performance Computing Applications, 06 2021.

[50] Martin Kronbichler and Karl Ljungkvist. Multigrid for matrix-free high-order finite element computations on graphics processors. ACM Transactions on Parallel Computing (TOPC), 6(1):1–32, 2019.

[51] Prashant Kumar, Carmen Rodrigo, Francisco J Gaspar, and Cornelis W Oosterlee. On local Fourier analysis of multigrid methods for PDEs with jumping and random coefficients. SIAM Journal on Scientific Computing, 41(3):A1385–A1413, 2019.

[52] Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. United States Governm. Press Office Los Angeles, CA, 1950.

[53] Cornelius Lanczos. Solution of systems of linear equations by minimized iterations. J. Res. Nat. Bur. Standards, 49(1):33–53, 1952.

[54] Jing Li and Olof B Widlund. On the use of inexact subdomain solvers for BDDC algorithms. Computer Methods in Applied Mechanics and Engineering, 196(8):1415–1428, 2007.

[55] LIBXSMM. `https://github.com/hfp/libxsmm`, 2021.

[56] David G Luenberger. Introduction to linear and nonlinear programming, volume 28. Addison-Wesley Reading, MA, 1973.

[57] Robert E Lynch, John R Rice, and Donald H Thomas. Direct solution of partial difference equations by tensor product methods. Numerische Mathematik, 6(1):185–199, 1964.

[58] Scott Maclachlan and Cornelis Oosterlee. A local Fourier analysis framework for finite-element discretizations of systems of PDEs. Numerical Linear Algebra With Applications - NUMER LINEAR ALGEBR APPL, 01 2011.

[59] Scott P. MacLachlan and C. W. Oosterlee. Local Fourier analysis for multigrid with overlapping smoothers applied to systems of PDEs. Numerical Linear Algebra with Applications, 18(4):751–774, 2011.

[60] Jan Mandel. Balancing domain decomposition. Communications in numerical methods in engineering, 9(3):233–241, 1993.

[61] Jan Mandel and Bedřich Sousedík. BDDC and FETI-DP under minimalist assumptions. Computing, 81(4):269–280, 2007.

[62] Dave A May, Patrick Sanan, Karl Rupp, Matthew G Knepley, and Barry F Smith. Extreme-scale multigrid components within PETSc. In Proceedings of the Platform for Advanced Scientific Computing Conference, pages 1–12, 2016.

[63] John D McCalpin. Memory bandwidth and system balance in HPC systems. Invited talk, Supercomputing, 2016.

[64] Arash Mehraban, Jeremy Thompson, Jed Brown, Richard Regueiro, Valeria Barra, and Henry Tufo. Simulating compressible and nearly-incompressible linear elasticity using an efficient parallel scalable matrix-free high-order finite element method. In 14th WCCM-ECCOMAS Congress 2020, volume 1400, 2021.

[65] Hans Meuer, Erich Strohmaier, Jack Dongarra, Horst Simon, and Martin Meuer. Top 500 list, 2020.

[66] J Tinsley Oden, L Demkowicz, W Rachowicz, and TA Westermann. Toward a universal hp adaptive finite element strategy, part 2. a posteriori error estimation. Computer methods in applied mechanics and engineering, 77(1-2):113–180, 1989.

[67] TC Papanastasiou, N Malamataris, and Ellwood K. A new outflow boundary condition. International Journal for Numerical Methods in Fluids, 14:587–608, March 1992.

[68] Luca F Pavarino, Olof B Widlund, and Stefano Zampini. BDDC preconditioners for spectral element discretizations of almost incompressible elasticity in three dimensions. SIAM Journal on Scientific Computing, 32(6):3604–3626, 2010.

[69] Antoine Petitet, R. Clint Whaley, Jack Dongarra, and Andrew Cleary. HPL-a portable implementation of the high-performance Linpack benchmark for distributed-memory computers, 2004.

[70] W Rachowicz, J Tinsley Oden, and L Demkowicz. Toward a universal hp adaptive finite element strategy part 3. design of hp meshes. Computer Methods in Applied Mechanics and Engineering, 77(1-2):181–212, 1989.

[71] Hannah Rittich. Extending and automating Fourier analysis for multigrid methods. PhD thesis, Universität Wuppertal, Fakultät für Mathematik und Naturwissenschaften, 2018.

[72] Daniel J Rixen, Charbel Farhat, Radek Tezaur, and Jan Mandel. Theoretical comparison of the FETI and algebraically partitioned FETI methods, and performance comparisons with a direct sparse solver. International Journal for Numerical Methods in Engineering, 46(4):501–533, 1999.

[73] Carmen Rodrigo, Francisco J Gaspar, and Ludmil T Zikatanov. On the validity of the local Fourier analysis. Journal of Computational Mathematics, 37(3):340–348, 2019.

[74] Einar M Rønquist and Anthony T Patera. Spectral element multigrid. I. formulation and numerical results. Journal of Scientific Computing, 2(4):389–406, 1987.

[75] Karl Rupp. CPU-GPU-MIC comparision charts, 2020.

[76] Hermann Amandus Schwarz. Gesammelte mathematische abhandlungen, volume 2. Springer, 1890.

[77] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[78] Klaus Stüben and Ulrich Trottenberg. Multigrid methods: Fundamental algorithms, model problem analysis and applications. In Multigrid methods, pages 1–176. Springer, 1982.

[79] Jeremy L. Thompson. LFAToolkit. `https://github.com/jeremylt/LFAToolkit.jl`, 2021.

[80] Andrea Toselli and Olof Widlund. Domain decomposition methods-algorithms and theory, volume 34. Springer Science & Business Media, 2005.

[81] Andrea Toselli and Olof Widlund. Domain Decomposition Methods: Algorithms and Theory, volume 34. Springer Science & Business Media, 2006.

[82] U. Trottenberg, C. W. Oosterlee, and A. Schüller. Multigrid. Academic Press, Inc., San Diego, CA, 2001. With contributions by A. Brandt, P. Oswald and K. Stüben.

[83] Valeria Barra, Jed Brown, Jeremy Thompson, and Yohann Dudouit. High-performance operator evaluations with ease of use: libCEED's Python interface. In Meghann Agarwal, Chris Calloway, Dillon Niederhut, and David Shupe, editors, Proceedings of the 19th Python in Science Conference, pages 75–80, July 2020.

[84] Jacobus JW van der Vegt and Sander Rhebergen. Discrete Fourier analysis of multigrid algorithms. Memorandum Department of Applied Mathematics, University of Twente, Enschede, The Netherlands, 2011.

[85] Olof Widlund and Maksymilian Dryja. An additive variant of the Schwarz alternating method for the case of many subregions. Technical Report 339, Ultracomputer Note 131. Department of Computer Science, Courant Institute, dec 1987.

[86] Roman Wienands and Wolfgang Joppich. Practical Fourier analysis for multigrid methods. CRC press, 2004.

[87] Harry Yserentant. Old and new convergence proofs for multigrid methods. Acta numerica, 2:285–326, 1993.

[88] Rui Zhang, Mengping Zhang, and Chi-Wang Shu. On the order of accuracy and numerical performance of two classes of finite volume weno schemes. Communications in Computational Physics, 9(3):807–827, 2011.