HIGH PERFORMANCE COMPUTING METHODS FOR EARTHQUAKE CYCLE SIMULATIONS

by

ALEXANDRE CHEN

A DISSERTATION

Presented to the Department of Computer Science
and the Division of Graduate Studies of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

June 2024

DISSERTATION APPROVAL PAGE

Student: Alexandre Chen

Title: High Performance Computing Methods for Earthquake Cycle Simulations

This dissertation has been accepted and approved in partial fulfillment of the requirements for the Doctor of Philosophy degree in the Department of Computer Science by:

Brittany A. Erickson	Chair
Boyana Norris	Core Member
Jee Choi	Core Member
Richard Tran Mills	Core Member
Leif Karlstrom	Institutional Representative

and

Krista M. Chronister	Interim Vice Provost for Graduate Studies

Original approval signatures are on file with the University of Oregon Division of Graduate Studies.

Degree awarded June 2024

DISSERTATION ABSTRACT

Alexandre Chen

Doctor of Philosophy

Department of Computer Science

June 2024

Title: High Performance Computing Methods for Earthquake Cycle Simulations

Earthquakes often occur on complex faults of multiscale physical features, with different time scales between seismic slips and interseismic periods for multiple events. Single event, dynamic rupture simulations have been extensively studied to explore earthquake behaviors on complex faults, however, these simulations are limited by artificial prestress conditions and earthquake nucleations. Over the past decade, significant progress has been made in studying and modeling multiple cycles of earthquakes through collaborations in code comparison and verification. Numerical simulations for such earthquakes lead to large-scale linear systems that are difficult to solve using traditional methods in this field of study. These challenges include increased computation and memory demands. In addition, numerical stability for simulations over multiple earthquake cycles requires new numerical methods. Developments in High performance computing (HPC) provide tools to tackle some of these challenges. HPC is nothing new in geophysics since it has been applied in earthquake-related research including seismic imaging and dynamic rupture simulations for decades in both research and industry. However, there's little work in applying HPC to earthquake cycle modeling. This dissertation presents a novel approach to applying the latest advancements in HPC

and numerical methods to solving computational challenges in earthquake cycle simulations.

CURRICULUM VITAE

NAME OF AUTHOR: Alexandre Chen

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon, Eugene, OR, USA
Fudan University, Shanghai, China

DEGREES AWARDED:

Bachelor of Science, Physics, 2018, Fudan University

AREAS OF SPECIAL INTEREST:

Itereative Algorithms
Preconditioner
Numerical PDEs
SBP-SAT finite difference methods
High performance computing

PROFESSIONAL EXPERIENCE:

Teaching Assistant, Department of Computer Science, University of
2019S, 2019F, 2020S, 2020F, 2021W, 2022W, 2023W, 2023F
Researcher, Frontier Development Lab, June - August, 2022

GRANTS, AWARDS AND HONORS:

Graduate Teaching/Research Fellowship, University of Oregon

PUBLICATIONS:

Thomas, D. R. K., & Smith, A. B. (2009). System models for the study of procrastination in high school students. *American Journal of Everything*, 100, 7–20.

ACKNOWLEDGEMENTS

Here is an acknowledgment

To so-and-so...

TABLE OF CONTENTS

Chapter Page

Chapter Page

Chapter Page

LIST OF FIGURES

Figure Page

LIST OF TABLES

Table Page

CHAPTER I

INTRODUCTION

1.1 Introduction to earthquakes

Every year, around 20,000 earthquakes happen around the world. Some are not noticeable, while others cause huge damage to property and life. Earthquakes have been recorded for thousands of years, and they are considered as signs or punishments to humans from supernatural powers in many civilizations. Over the past centuries, with advancements in mathematics, physics, geology, and other natural sciences, we have a more structured understanding of earthquakes today.

An earthquake represents a complex process of fault slip and energy release within the Earth's crust, driven by tectonic forces and resulting in the shaking of the ground and potentially causing damage to structures and infrastructure. An earthquake occurs due to the sudden release of accumulated stress along a fault line, resulting in rapid movement known as fault slip. This movement can be described in terms of several key components:

1.1.1 Fault rupture. The earthquake begins with the rupture of the fault, where the stress accumulated along the fault plane exceeds the strength of the rocks, causing them to fracture and slide past each other. This rupture initiates the seismic event.

1.1.2 Sesmic waves. As the fault ruptures, it generates seismic waves that propagate through the Earth's crust and propagate outward from the fault. These weaves transmit energy in the form of vibrations, which cause the ground to shake.

1.1.3 Slip motion. Slip motions are the noticeable components of an earthquake movement. The fault slip during an earthquake involves two types of movements

- Primary slip (seismic slip): This is the sudden and rapid movement along the fault plane during the initial rupture. It is usually associated with the intense shaking and damage from the earthquakes
- Afterslip: Following the primary slip, there may be additional movement along the fault. This ongoing slip can continue for days, weeks, or even months after the initial earthquake.

1.1.4 Displacement. The amount of fault slip during an earthquake is measured in terms of displacements. This is the distance that one side of the fault moves relative to the other. Displacement can be horizontal, vertical, or both.

1.2 Seismic and seismic slip

The slip motion of an earthquake can further be classified into seismic and seismic slips.

1.2.1 Seismic slip. Seismic slip refers to the sudden release of accumulated tectonic stress along a fault plane, resulting in what is often called an earthquake. This type of fault slip is characterized by rapid and dynamic movement, which generates seismic waves propagating through Earth's crust, causing ground shake and potential damages to infrastructures. Seismic fault slip occurs when stress accumulated along a fault exceeds the frictional resistance holding the fault surface, causing sudden slip and rupture.

1.2.2 Aseismic slip. Aseismic slip, also known as creep or slow slip, refers to gradual continuous movement along a fault plane without generating

significant earthquakes and seismic waves. Unlike seismic slip, seismic slip occurs at rates that are usually much slower and may not produce noticeable ground shaking or seismic activity. Instead, seismic slip represents a steady release of tectonic stress along the fault, often occurring in between larger seismic events. However, the stress could still increase during seismic slip, leading to seismic slip in the future. Aseismic slip can contribute to the overall deformation of the Earth's crust and could play a potential role in seismic hazard assessments and forecasting. Modeling the behavior of aseismic slip has been essential in order to understand the nucleation of seismic slip and the mechanism behind multiple cycles of earthquakes.

1.2.3 Budget. In the context of seismology, budget refers to the distribution and allocation of accumulated tectonic stress or energy between seismic and aseismic slip events. Understanding the balance between seismic and aseismic slip budgets is important for assessing seismic hazard and fault behavior. It helps researchers and geoscientists to understand the mechanisms governing fault movement and stress release to forecast earthquakes. Here's a great review by Jean-Philippe discussing principles of fault slip budget determination and observational constraints on seismic and aseismic slip Avouac (2015).

1.3 Velocity weakening/strengthening

Understanding the friction behavior along the fault is the key to understanding the different behaviors of seismic and aseismic slips. Friction is often associated with the velocity of the fault displacement. Based on the different responses to sliding velocity, regions on the fault can be classified into two types: velocity weakening and velocity strengthening.

1.3.1 Veclocity weakening region. In a velocity weakening region, the friction resistance between the fault surfaces decreases with an increasing slip

velocity. In other words, as the sliding velocity along the fault increases, the friction strength decreases.

This phenomenon is crucial in earthquake dynamics because it promotes the instability of the fault and facilitates the rapid release of accumulated stress during earthquakes. When the friction resistance decreases with increasing velocity, the fault becomes more prone to slip suddenly and can generate earthquake waves. Velocity weakening regions are often associated with materials or conditions that exhibit unstable slip behavior, such as fault gouge, pore pressure, and fluids.

1.3.2 Velocity strengthening region.

1.4 Rate-and-state friction law

Since the recognition that earthquakes probably represent frictional slip instabilities in the 1960s, interest in determining frictional properties has been increased. The stability of frictional sliding depends on whether frictional resistance increases or drops during slip. Laboratory experiments have shown that frictional sliding is mainly a rate-dependent process in a steady-state regime with constant stress and steady velocity. Due to this, a state variable needs to be introduced to describe

- the transient behavior observed in non-steady-state experiments
- healing in hold-and-stick experiments

The formulation of such rate-and-state variable has significantly simplified the impacts of several parameters from rheology on the slip rate, which enables the development of numerical models to simulate earthquake cycles.

For most materials, it has been revealed by laboratory experiments on frictional sliding that the following conditions exist:

For most materials, it has been revealed by laboratory experiments on frictional sliding that the following conditions exist:

- The resistance to sliding depends on the sliding rate at steady rate, along with a logarithmic dependency of the coefficient of friction on the slip rate
- The resistance to sliding increases to a transient peak value when the imposed slip rate is suddenly changed, with the peak value being a logarithmic function of the slip rate
- The friction coefficient is approximately a linear function with the logarithmic of the time in hold-and-slip experiments.

Laboratory measurements at slow sliding rates can be reproduced relatively well with a rate-and-state formalism on the order of microns per second. Various laws have been proposed (Dieterich (1979a, 1979b); Marone (1998); Ruina (1983)).

One common such law is called aging law

$$\mu = \mu_* + (a - b) \ln \frac{V}{V_*} \tag{1.1}$$

$$\frac{d\theta}{dt} = 1 - \frac{V\theta}{D_c} \tag{1.2}$$

At steady state, the law is purely rate dependent

$$\mu_{ss} = \mu_* + (a - b) \ln \frac{V}{V_*} \tag{1.3}$$

For a single-degree-of-freedom system such as a spring-and-slider system, the stability analysis shows that the slip can be stable only if $a - b > 0$ and that unstable slip requires that $a - b < 0$ For unstable slip to occur, it requires $a - b$ that is smaller than a critical negative value, defining an intermediate domain of conditional stability. For a crack with size L embedded in an elastic medium with

shear modulus G, the condition for unstable slip is

$$a - b < -\lambda \frac{GD_c}{L\sigma_n'} \tag{1.4}$$

where λ is on the order of unity. σ_n' is the effective normal stress with $\sigma_n' - \sigma_n - P$ where P is pore pressure. In the limit when the pore pressure becomes near lithostatic, the critical value becomes infinite. This implies that high pore pressure should promote stable slip through the reduction of the effective normal stress. The rate-and-state friction and many definitions here are important concepts in the earthquake cycle simulations that are going to be discussed in later chapters.

CHAPTER II

METHODOLOGY

2.1 Numerical methods

Computational modeling of the natural world involves pervasive material and geometric complexities that are hard to understand, incorporate, and analyze. The partial differential equations (PDEs) governing many of these systems are subject to boundary and interface conditions, and all numerical methods share the fundamental challenge of how to enforce these conditions in a stable manner. Additionally, applications involving elliptic PDEs or implicit time-stepping require efficient solution strategies for linear systems of equations.

Most applications in the natural sciences are characterized by multiscale features in both space and time which can lead to huge linear systems of equations after discretization. Our work is motivated by large-scale ($\sim$hundreds of kilometers) earthquake cycle simulations where frictional faults are idealized as geometrically complex interfaces within a 3D material volume and are characterized by much smaller-scale features ($\sim$microns) Erickson and Dunham (2014); Kozdon, Dunham, and Nordström (2012). In contrast to the single-event simulations, e.g. Roten et al. (2016), where the computational work at each time step is a single matrix-vector product, earthquake cycle simulations must integrate with adaptive time-steps through the slow periods between earthquakes, and are tasked with a much more costly linear solve. For example, even with upscaled parameters so that larger grid spacing can be used, the 2D simulations in Erickson and Dunham (2014) generated matrices of size $\sim 10^6$, and improved resolution and 3D domains would increase the system size to $\sim 10^9$ or greater. Because iterative schemes

are most often implemented for the linear solve (since direct methods require a matrix factorization that is often too large to store in memory), it is no surprise that the sparse matrix-vector product (SpMV) arises as the main computational workhorse. The matrix sparsity and condition number depend on several physical and numerical factors including the material heterogeneity of the Earth's material properties, order of accuracy, the coordinate transformation (for irregular grids), and the mesh size. For large-scale problems, matrix-free (on-the-fly) techniques for the SpMV are fundamental when the matrix cannot be stored explicitly.

In this work, we use summation-by-parts (SBP) finite difference methods Kreiss and Scherer (1974); Mattsson and Nordström (2004); Strand (1994); Svärd and Nordström (2014), which are distinct from traditional finite difference methods in their use of specific one-sided approximations at domain boundaries that enable the highly valuable proof of stability, a necessity for numerical convergence. Weak enforcement of boundary conditions has additional superior properties over traditional methods, for example, the simultaneous-approximation-term (SAT) technique, which relaxes continuity requirements (of the grid and the solution) across physical or geometrical interfaces, with low communication overhead for efficient parallel algorithms Del Rey Fernández, Hicken, and Zingg (2014).

For these reasons SBP-SAT methods are widely used in many areas of scientific computing, from the flow over airplane wings to biological membranes to earthquakes and tsunamigenesis Erickson and Day (2016); Lotto and Dunham (2015); Nordström and Eriksson (2010); Petersson and Sjögreen (2012); Swim et al. (2011); Ying and Henriquez (2007); these studies, however, have not been developed for linear solves or were limited to small-scale simulations.

With this work, we contribute a novel iterative scheme for linear systems based on SBP-SAT discretizations where nontrivial computations arise due to boundary treatment. These methods are integrated into our existing, public software for simulations of earthquake sequences. Specifically, we make the following contributions:

- Since preconditioning of iterative methods is a hugely consequential step towards improving convergence rates, we develop a custom geometric multigrid preconditioned conjugate gradient (MGCG) algorithm which shows a near-constant number of iterations with increasing system size. The required iterations (and time-to-solution) are much lower compared to several off-the-shelf preconditioners offered by the PETSc library Balay et al. (2023), a state-of-the-art library for scientific computing.

- We develop custom, matrix-free GPU kernels (specifically for SBP-SAT methods) for computations in the volume and boundaries, which show improved performance as compared to the native, matrix-explicit implementation while requiring only a fraction of memory.

- GPU-acceleration of our resulting matrix-free, preconditioned iterative scheme shows superior performance compared to state-of-the-art methods offered by NVIDIA.

Furthermore, the ubiquity of SBP-SAT methods in modern scientific computing applications means our work has the propensity to advance scientific studies currently limited to small-scale problems.

2.2 SBP-SAT methods

SBP methods approximate partial derivatives using one-sided differences at all points close to the boundary node, generating a matrix approximating a partial derivative operator. In this work we focus on second-order derivatives which appear in (**??**), however the matrix-free methods we derive are applicable to any second-order PDE. We consider SBP finite-difference approximations to boundary-value problem (**??**), i.e. on the square computational domain $\bar{\Omega}$; solutions on the physical domain Ω are obtained by the inverse coordinate transformation.

In this work, we focus on SBP operators with second-order accuracy which contains abundant complexity at domain boundaries to enable insight into implementation design extendable to higher-order methods. To provide background on the SBP methods we first describe the 1D operators, as Kronecker products are used to form their multi-dimensional counterparts.

2.2.0.1 1D Operators. We discretize the spatial domain $-1 \leq r \leq 1$ with $N+1$ evenly spaced grid points $r_i = -1 + ih, i = 0, \ldots, N$ with grid spacing $h = 2/N$. A function u projected onto the computational grid is denoted by $\boldsymbol{u} = [u_0, u_1, \ldots, u_N]^T$ and is often taken to be the interpolant of u at the grid points. We define the grid basis vector $\vec{e}_j$ to be a vector with value 1 at grid point j and 0 for the rest, which allows us to extract the jth component: $u_j = \vec{e}_j^T \vec{u}$.

Definition 1 (First Derivative)**.** *A matrix $\boldsymbol{D}_r$ is an SBP approximation to the first derivative operator $\partial/\partial r$ if it can be decomposed as $\boldsymbol{H}\boldsymbol{D}_r = \boldsymbol{Q}$ with $\boldsymbol{H}$ being SPD and $\boldsymbol{Q}$ satisfying $\vec{u}^T(\boldsymbol{Q} + \boldsymbol{Q}^T)\vec{v} = u_N v_N - u_0 v_0$.*

Here, $\boldsymbol{H}$ is a diagonal quadrature matrix and $\boldsymbol{D}_r$ is the standard central finite difference operator in the interior which transitions to one-sided at boundaries.

Definition 2 (Second Derivative)**.** *Letting $c = c(r)$ denote a material coefficient, we define matrix $\boldsymbol{D}_{rr}^{(c)}$ to be an SBP approximation to $\frac{\partial}{\partial r}\left(c\frac{\partial}{\partial r}\right)$ if it can be decomposed as $\boldsymbol{D}_{rr}^{(c)} = \boldsymbol{H}^{-1}(-\boldsymbol{M}^{(c)} + c_N\vec{e}_N\vec{d}_N^T - c_0\vec{e}_0\vec{d}_0^T)$ where $\boldsymbol{M}^{(c)}$ is SPD and $\vec{d}_0^T\vec{u}$ and $\vec{d}_N^T\vec{u}$ are approximations of the first derivative of u at the boundaries.*

With these properties, both $\boldsymbol{D}_r$ and $\boldsymbol{D}_{rr}^{(c)}$ mimic integration-by-parts in a discrete form which enables the proof of discrete stability Mattsson, Ham, and Iaccarino (2009); Mattsson and Nordström (2004).

$\boldsymbol{D}_{rr}^{(c)}$ is a centered difference approximation within the interior of the domain, but includes approximations at boundary points as well. For illustrative purposes alone, if $c = 1$ (e. g. a constant coefficient case), the matrix is given by

$$\boldsymbol{D}_{rr}^{(c)} = \frac{1}{h^2}\begin{bmatrix} 1 & -2 & 1 & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & 1 & -2 & 1 \end{bmatrix},$$

which, as highlighted in red, resembles the traditional (second-order-accurate) Laplacian operator in the domain interior.

2.2.0.2 2D SBP Operators. The 2D domain $\bar{\Omega}$ is discretized using $N + 1$ grid points in each direction, resulting in an $(N + 1) \times (N + 1)$ grid of points where grid point (i, j) is at $(x_i, y_j) = (-1 + ih, -1 + jh)$ for $0 \leq i, j \leq N$ with $h = 2/N$. Here we have assumed equal grid spacing in each direction, only for notational ease; the generalization to different numbers of grid points in each dimension does not impact the construction of the method and is implemented in our code. A 2D grid function $\boldsymbol{u}$ is ordered lexicographically and we

let $\boldsymbol{C}_{ij} = \text{ diag}(\boldsymbol{c}_{ij})$ define the diagonal matrix of coefficients, see Kozdon, Erickson, and Wilcox (2020).

In this work we imply summation notation whenever indices are repeated. Multi-dimensional SBP operators are obtained by applying the Kronecker product to 1D operators, for example, the 2D second derivative operators are given by

$$\begin{aligned}\frac{\partial}{\partial i} c_{ij} \frac{\partial}{\partial j} &\approx \tilde{\boldsymbol{D}}_{ij}^{c_{ij}} \\ &= (\boldsymbol{H} \otimes \boldsymbol{H})^{-1} \left[-\tilde{\boldsymbol{M}}_{ij}^{(c_{ij})} + \boldsymbol{T} \right], \end{aligned} \tag{2.1}$$

for $i, j \in \{r, s\}$. Here $\tilde{\boldsymbol{M}}_{ij}^{(c_{ij})}$ is the sum of SPD matrices approximating integrated second derivatives (i.e. sum over repeated indices i, j) for example $\int_{\bar{\Omega}} \frac{\partial}{\partial r} c_{rr} \frac{\partial}{\partial r} \approx \tilde{\boldsymbol{M}}_{rr}^{(c_{rr})}$ and matrix $\boldsymbol{T}$ involves the boundary derivative computations, see Erickson, Kozdon, and Harvey (2022) for complete details.

2.2.0.3 SAT Penalty Terms.

SBP methods are designed to work with various impositions of boundary conditions that lead to provably stable methods, for example through weak enforcement via the simultaneous-approximation-term (SAT) Carpenter, Gottlieb, and Abarbanel (1994) which we adopt here. As opposed to traditional finite difference methods that "inject" boundary data by overwriting grid points with the given data, the SAT technique imposes boundary conditions weakly (through penalization), so that all grid points approximate both the PDE and the boundary conditions up to a certain level of accuracy. The combined approach is known as SBP-SAT. Where traditional methods that use injection or strong enforcement of boundary/interface conditions destroy the discrete integration-by-parts property, using SAT terms enables proof of the method's stability (a necessary property for numerical convergence) Mattsson (2003).

2.2.1 An example of the SBP-SAT technique for PDE.

We use the following example from **?** to showcase an example of applying the SBP-SAT method for PDEs. Let's consider the advection problem in 1D.

$$\begin{aligned} \boldsymbol{u}_t + \boldsymbol{u}_x &= 0, \quad 0 < x < 1, t > 0 \\ \boldsymbol{u}(0,t) &= \boldsymbol{g}(t), \quad t > 0 \\ \boldsymbol{u}(x,t) &= \boldsymbol{h}(x), \quad 0 < x < 1 \end{aligned} \tag{2.2}$$

where both g and h are known for initial and boundary conditions. The problem Equation 2.2 has an energy-estimate and is well-posed. We can easily learn that the analytical solution for this equation is a right-traveling wave.

We discretize 1D domain with $N + 1$ points in a uniform grid on $[0, 1]$ using the method described in **??**. By applying SBP-SAT discretization in space to Equation 2.2, we get

$$\begin{aligned} \boldsymbol{u}_t + D_1\boldsymbol{u} &= P^{-1}\sigma(\boldsymbol{u}_0 - \boldsymbol{g})\boldsymbol{e}_0, \quad t > 0 \\ \boldsymbol{u}(0) &= \boldsymbol{h} \end{aligned} \tag{2.3}$$

where $\boldsymbol{u} = [u_0, \ldots, u_N]^T, \boldsymbol{h} = [h_0, \ldots, h_N]^T, \sigma \in \mathbb{R}$ is a penalty parameter which is determined through stability condition. $\boldsymbol{e}_0 = [1, 0, \ldots, 0]^T \in \mathbb{R}^{N+1}$. To determine the value for σ so that the problem Equation 2.3 is strongly stable, we have

$$||\boldsymbol{u}(t)||^2 \leq K(t)(||\boldsymbol{h}||^2 + \max_{\tau \in [0,t]} |\boldsymbol{g}(\tau)|^2) \tag{2.4}$$

The $K(t)$ in Equation 2.4 is independent of the data and bounded for any finite t and meshsize Δx. Further details about $K(t)$ are given in Svärd and Nordström (2014); **?**. Applying the energy method by multiplying the equation Equation 2.3 with $\boldsymbol{u}^T P$ and adding the transpose with the SBP property **??**, we find

$$\frac{d}{dt}||\boldsymbol{u}||_P^2 = -\frac{\sigma^2}{1+2\sigma}\boldsymbol{g}^2 - \boldsymbol{u}_N^2 + \frac{[(1+2\sigma)\boldsymbol{u}_0 - \sigma\boldsymbol{g}]^2}{1+2\sigma} \tag{2.5}$$

By time-integration, this leads to an estimate of the form Equation 2.4 for $\sigma < -1/2$.

2.2.2 Poisson's equation with SBP-SAT Methods. We consider the 2D Poisson equation on the unit square Ω with both Dirichlet and Neumann conditions for generality, as each appears in earthquake problems (e.g. Earth's free surface manifests as a Neumann condition, and the slow motion of tectonic plates is usually enforced via a Dirichlet condition). This is an important and necessary first step before additional complexities such as variable material properties, complex geometries, and fully 3D problems. The governing equations are given by

$$-\Delta u = f, \qquad \text{for } (x,y) \in \Omega, \tag{2.6a}$$

$$u = g_{\mathrm{W}}, \quad x = 0, \tag{2.6b}$$

$$u = g_{\mathrm{E}}, \quad x = 1, \tag{2.6c}$$

$$\boldsymbol{n} \cdot \nabla u = g_{\mathrm{S}}, \quad y = 0, \tag{2.6d}$$

$$\boldsymbol{n} \cdot \nabla u = g_{\mathrm{N}}, \quad y = 1, \tag{2.6e}$$

where $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$, the field $u(x,y)$ is the unknown particle displacement, the scalar function $f(x,y)$ is the source function, and vector $\boldsymbol{n}$ is the outward pointing normal to the domain boundary $\partial\Omega$. The g's represent boundary data on the west, east, south, and north boundaries.

The SBP-SAT discretization of (2.6) is given by

$$-\boldsymbol{D}_2 \boldsymbol{u} = \boldsymbol{f} + \boldsymbol{b}^N + \boldsymbol{b}^S + \boldsymbol{b}^W + \boldsymbol{b}^E, \tag{2.7}$$

where $\boldsymbol{D}_2 = (\boldsymbol{I} \otimes \boldsymbol{D}_{xx}) + (\boldsymbol{D}_{yy} \otimes \boldsymbol{I})$ is the discrete Laplacian operator and $\boldsymbol{u}$ is the grid function approximating the solution, formed as a stacked vector of vectors. The SAT terms $\boldsymbol{b}^N, \boldsymbol{b}^S, \boldsymbol{b}^W, \boldsymbol{b}^E$ enforce all boundary conditions weakly. To illustrate the structure of these vectors, the SAT term enforcing Dirichlet data on the west

boundary is given by

$$\boldsymbol{b}^W = \alpha \left(\boldsymbol{H}^{-1} \otimes \boldsymbol{I}\right) \left(\boldsymbol{E}_{\mathrm{W}}\boldsymbol{u} - \boldsymbol{e}_{\mathrm{W}}^T \boldsymbol{g}_{\mathrm{W}}\right) \tag{2.8}$$

$$- \left(\boldsymbol{H}^{-1}\boldsymbol{e_0}\boldsymbol{d_0}^T \otimes \boldsymbol{I}\right) \left(\boldsymbol{E}_{\mathrm{W}}\boldsymbol{u} - \boldsymbol{e}_{\mathrm{W}}^T \boldsymbol{g}_{\mathrm{W}}\right), \tag{2.9}$$

where α again represents a penalty parameter, $\boldsymbol{E}_{\mathrm{W}}$ is a sparse boundary extraction operator, and $\boldsymbol{e}_{\mathrm{W}}^T$ is an operator that lifts the boundary data to the whole domain. Details of all the SAT terms can be found in Erickson and Dunham (2014). System (2.7) can be rendered SPD by multiplying from the left by $(\boldsymbol{H} \otimes \boldsymbol{H})$, producing the sparse linear system $\boldsymbol{Au} = \boldsymbol{b}$.

2.3 Iterative Method

This section will present a brief review of iterative methods for solving large linear systems in our research.

2.3.1 Stationary Iterative Methods. Stationary iterative methods can be expressed in the simple form

$$\boldsymbol{u}^{k+1} = \boldsymbol{Q}\boldsymbol{u}^k + \boldsymbol{q} \tag{2.10}$$

where $\boldsymbol{Q}$ and $\boldsymbol{q}$ are placeholders for a matrix and a vector respectively, both independent of iteration step k. Stationary iterative methods, such as the Gauss-Seidel method, act as smoothers for damping frequency components of the solution vectors. Further backgrounds of these iterative methods can be found in Saad (2003)

The considered problem for iterative methods is a linear equation system of the form $\boldsymbol{Au} = \boldsymbol{f}$. We introduce splitting matrix $\boldsymbol{S}$ as follows:

$$\boldsymbol{A} = \boldsymbol{S} + (\boldsymbol{A} - \boldsymbol{S}) \tag{2.11}$$

With this introduced splitting matrix $\boldsymbol{S}$, we can rewrite the linear equation system as

$$\boldsymbol{S}\boldsymbol{u} = (\boldsymbol{S} - \boldsymbol{A})\boldsymbol{u} + \boldsymbol{f} \tag{2.12}$$

and the iterative scheme of the splitting method is defined as

$$\boldsymbol{u}^{k+1} = \boldsymbol{S}^{-1}((\boldsymbol{S} - \boldsymbol{A})\boldsymbol{u}^k + f) \tag{2.13}$$

For further analysis, it is useful to introduce the iteration matrix $\boldsymbol{M}$ as

$$\boldsymbol{M} = \boldsymbol{S}^{-1}(\boldsymbol{S} - \boldsymbol{A}) \tag{2.14}$$

.

If we define $\boldsymbol{Q} = \boldsymbol{M}$ and $\boldsymbol{q} = \boldsymbol{S}^{-1}\boldsymbol{f}$, then Equation 2.13 can be written as $\boldsymbol{u} = \boldsymbol{Q}\boldsymbol{u} + \boldsymbol{q}$, and it satisfy

$$\boldsymbol{e}^{k+1} = \boldsymbol{M}\boldsymbol{e}^k \tag{2.15}$$

where $\boldsymbol{e}^k$ is the error $\boldsymbol{u^k} - \boldsymbol{u}$ for the iteration step k. Because $\boldsymbol{M}$ is only determined by the initial linear system and the splitting matrix $\boldsymbol{S}$, and it is not changed in each iteration step, this method is called the stationary iterative method.

The spectral radius ρ is the largest absolute eigenvalue of a matrix. The stationary iterative method converges if and only if the spectral radius ρ of the iteration matrix $\boldsymbol{M}$ satisfies the following condition

$$\rho(\boldsymbol{M}) < 1 \tag{2.16}$$

Such convergence holds for any initial guess $\boldsymbol{u}^0$ and any right-hand side $\boldsymbol{f}$. Different stationary iterative methods differ in the choice of splitting matrix $\boldsymbol{S}$. We will present the Jacobi method and the Gauss-Seidel method for comparison here.

2.3.1.1 The Jacobi Method.

In the Jacobi method, the diagonal $\boldsymbol{D}$ of the matrix $\boldsymbol{A}$ for the linear system is chosen as the splitting matrix $\boldsymbol{S}$. Hence the decomposition is expressed as

$$\boldsymbol{A} = \boldsymbol{D} + (\boldsymbol{A} - \boldsymbol{D}) \text{ or } \boldsymbol{A} = \boldsymbol{D} + (-\boldsymbol{L} - \boldsymbol{U}) \tag{2.17}$$

Where $-\boldsymbol{L}$ denotes the strictly lower triangle and $-\boldsymbol{U}$ denotes the strictly upper triangle of the matrix $\boldsymbol{A}$. Similar to Equation 2.13, the Jacobi method is then

$$\boldsymbol{u}^{k+1} = \boldsymbol{D}^{-1}((\boldsymbol{L} + \boldsymbol{U})\boldsymbol{u}^k + \boldsymbol{f}) \tag{2.18}$$

In terms of matrix indices, the Jacobi method can be written as

$$u_i^{k+1} = \frac{1}{A_{ii}}(f_i - \sum_{j=1, i \neq j} A_{ij} u_j^k) \tag{2.19}$$

For matrix-free forms, similar results can be obtained via slight modifications to this form.

2.3.1.2 The Gauss-Seidel Method.

In the Gauss-Seidel method, the splitting matrix is chosen as $\boldsymbol{S} = (\boldsymbol{D} - \boldsymbol{L})$. The decomposition is then expressed as

$$\boldsymbol{A} = (\boldsymbol{D} - \boldsymbol{L}) + (\boldsymbol{A} - (\boldsymbol{D} - \boldsymbol{L})) \text{ or } \boldsymbol{A} = \boldsymbol{D} - \boldsymbol{L} + (-\boldsymbol{U}) \tag{2.20}$$

Similar to Equation 2.13, the Gauss-Seidel method is then

$$\boldsymbol{u}^{k+1} = (\boldsymbol{D} - \boldsymbol{L})^{-1}(\boldsymbol{U}\boldsymbol{u}^k + \boldsymbol{f}) \tag{2.21}$$

To derive the index form of the Gauss-Seidel method, some further transformations are needed.

$$\boldsymbol{D}\boldsymbol{u}^{k+1} - \boldsymbol{L}\boldsymbol{u}^{k+1} = \boldsymbol{U}\boldsymbol{u}^k + \boldsymbol{f} \tag{2.22}$$

and

$$\boldsymbol{u}^{k+1} = \boldsymbol{D}^{-1}(\boldsymbol{L}\boldsymbol{u}^{k+1} + \boldsymbol{U}\boldsymbol{u}^k + \boldsymbol{f}) \tag{2.23}$$

and the index form is given as

$$u_i^{k+1} = \frac{1}{A_{ii}}(f_i - \sum_{j=1}^{i-1} A_{ij}u_j^{k+1} - \sum_{i+1}^{n} A_{ij}u_j^k) \tag{2.24}$$

2.3.1.3 Comparison of the Jacobi and the Gauss-Seidel Method.

It might seem that in Equation 2.23 the right-hand side contains the result from the iteration step k+1 and such an iterative scheme would fail. However, a closer observation would notice that the $\boldsymbol{u}^{k+1}$ is multiplied by the negation of the lower triangle $-\boldsymbol{L}$ of the matrix $\boldsymbol{A}$. This means for each element j in the vector $\boldsymbol{u}^{k+1}$, only newly updated elements before index j are used, hence there is no logical problem in this iterative scheme. This is more obvious in the index form Equation 2.24

This is the most important difference between the Jacobi and the Gauss-Seidel method. When computing the i-th element u_i^{k+1} in the iteration step k+1, the Gauss-Seidel method already uses all available iterates u_j^{k+1} with $j = 1 \ldots (i-1)$, while the Jacobi method only uses the iterates from the previous iteration step k. In other words, while the Jacobi method adds all increments *simultaneously* only after cycling through all degrees of freedom, the Gauss-Seidel method adds all increments *successively* as soon as available. As a result, the Gauss-Seidel method has the advantage that one vector is sufficient to update its vector elements i successively, in contrast to the Jacobi method where an additional vector is required. However, in terms of computational cost, the difference in memory requirement is negligible in practice.

On the other hand, the operations for the iteration of different vector elements do not coincide in the Jacobi method, which means the parallelization for the Jacobi method is straightforward. However, in the Gauss-Seidel method, the nodal ordering influences the convergence behavior. There are various nodal

orderings summarized in **?**, such as red-black, lexicographical, zebra-line, and four-color ordering. More advanced algorithms are required for the successful parallelization of the Gauss-Seidel method. Otherwise, uncontrolled splitting of the process leads to the so-called *chaotic* Gauss-Seidel method.

In terms of convergence, both stationary methods depend on the spectral radius of the corresponding iteration matrix $\boldsymbol{Q}$, which is affected by the splitting matrix $\boldsymbol{S}$ chosen for each of these two methods. If both methods converge, the convergence rate of the Gauss-Seidel method is better as each iteration would use the updated data as soon as available.

Specifically, it is sufficient for the Jacobi method to converge if the system Matrix $\boldsymbol{A}$ is strictly diagonally dominant **?**

$$|A_{ii}| > \sum_{j=1, j\neq i}^{n} |A_{ij}| \text{ for all } i \tag{2.25}$$

For a linear system that doesn't satisfy this condition, convergence can be achieved by additional damping. For the Gauss-Seidel method, other than the given condition in Equation 2.25, the convergence is also guaranteed if the system matrix A is positive definite. The second condition is usually satisfied for the finite difference method or the finite element method if properly restrained and stabilized. **?** Other than directly used as standalone iterative solvers, the damped Jacobi method or the Gauss-Seidel method can be applied as smoothers within the multigrid method.

2.3.1.4 Relaxation methods. Relaxation methods are also stationary iterative methods, thus they can also be presented in the form Equation 2.10. For each of the methods introduced previously, there also exists a corresponding relaxation method. In comparison to the precedent methods, the relaxation methods scale each increment by a constant relaxation factor ω. The general form

of the relaxation methods is given by the following simplified algorithmic expression

$$u_i^{k+1} := (1-\omega)u_i^k + \omega\breve{u}_i^{k+1} \tag{2.26}$$

where for each individual index i, the temporary variable $\breve{u}_i^{k+1}$ is computed as the u_i^{k+1} of the Jacobi method in the case of the simultaneous over-relaxation method (JOR method) or as the u_i^{k+1} of the Gauss-Seidel method in the case of the successive over-relaxation method (SOR method). Thus when $\omega = 1$, the relaxation scheme is identical to the Jacobi or the Gauss-Seidel method.

The optimum relaxation factor can be derived theoretically from the spectral radius of the iteration matrix. However, this is expensive. For more practical use, several methods for the determination of ω were proposed in **?** and **?**.

2.3.2 Krylov Subspace Methods. Stationary iterative methods have been applied for a long time in history, but over the last few decades, Krylov subspace methods become more popular. These methods focus on building Krylov subspaces, named after Aleksei Nikolaevich Krylov who used these spaces to analyze oscillations of mechanical systems **?**. The Krylov subspace takes the form

$$\mathcal{K}_k(A, \boldsymbol{v}) := \text{span}\{\boldsymbol{v}, A\boldsymbol{v}, \ldots, A^{k-1}\boldsymbol{v}\} \tag{2.27}$$

where $A \in \mathbb{C}^{n\times n}$ and $\boldsymbol{v} \in \mathbb{C}^n$

2.3.2.1 Conjugate Gradient Method. The conjugate gradient (CG) method was developed by *Hestenes & Stiefel* **?** as the first Krylov subspace method, and has been one of the most popular iterative methods in solving linear systems. Compared to the iterative methods mentioned in previous sections which are known to be stationary, the CG method is non-stationary. Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be a symmetric positive definite (SPD) matrix, and $\boldsymbol{f} \in \mathbb{R}^n$ be a real vector, then the

minimization problem of the quadratic form $F(x) = min$

$$F(\boldsymbol{u}) = \frac{1}{2}\boldsymbol{u}^T\boldsymbol{A}\boldsymbol{u} - \boldsymbol{f}^T\boldsymbol{u} \tag{2.28}$$

is equivalent to getting its derivative

$$\text{grad}F(\boldsymbol{u}) = \boldsymbol{A}\boldsymbol{u} - \boldsymbol{f} \tag{2.29}$$

equal to the zero vector

$$\text{grad}F(\boldsymbol{u}) = 0 \tag{2.30}$$

The CG method is an iterative minimizer of the given quadratic form and therefore an iterative solver for the linear equation system $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f}$ when $\boldsymbol{A}$ is SPD. The quadratic form is always minimized from an approximate vector $\boldsymbol{u}^k$ in the direction of a provided search vector $\boldsymbol{p}^k \neq 0$, which can be written as

$$F(\boldsymbol{u}^k + \lambda\boldsymbol{p}^k) = min \tag{2.31}$$

where both $\boldsymbol{u}^k$ and $\boldsymbol{p}^k$ are constant vectors $\in \mathbb{R}$ and a scalar variable $\lambda \in \mathbb{R}$. This leads to the following parabola function of λ

$$\begin{aligned}(\frac{1}{2}\boldsymbol{p}^{k^T}\boldsymbol{A}\boldsymbol{p}^k)\lambda^2 + (\boldsymbol{p}^{k^T}\boldsymbol{A}\boldsymbol{u}^k - \boldsymbol{p}^{k^T}\boldsymbol{f})\lambda \\ + ((\frac{1}{2}\boldsymbol{u}^{k^T}\boldsymbol{A}\boldsymbol{u}^k) - \boldsymbol{u}^{k^T}) = min\end{aligned} \tag{2.32}$$

This quadratic form is minimized for

$$\lambda = \frac{\boldsymbol{p}^{k^T}(\boldsymbol{f} - \boldsymbol{A}\boldsymbol{u}^k)}{\boldsymbol{p}^{k^T}\boldsymbol{A}\boldsymbol{p}^k} \tag{2.33}$$

The ideal search direction $\boldsymbol{p}^k$ would be the error $\boldsymbol{e}$, however, this would require us to know the exact solution $\boldsymbol{u}$. As a compromise, the negative gradient of the quadratic form at $\boldsymbol{u}^k$ is the best intuitive search direction from the local view of $\boldsymbol{u}^k$. The search direction corresponds to the residual $\boldsymbol{r}^k$ is now

$$-\text{grad}F(u^k) = \boldsymbol{f} - \boldsymbol{A}\boldsymbol{u}^k = \boldsymbol{r}^k \tag{2.34}$$

with $\boldsymbol{p}^k = \boldsymbol{r}^k$. We define the following equations

$$\lambda_k = \frac{\boldsymbol{r}^{k^T}\boldsymbol{r}^k}{\boldsymbol{r}^{k^T}\boldsymbol{A}\boldsymbol{r}^k} \tag{2.35}$$

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \lambda_k \boldsymbol{r}^k \tag{2.36}$$

to describe the iterative process for one iterative step, which is called the method of steepest descent due to the fact that for any iteration step k, the search direction $\boldsymbol{p}^k$ is defined by (-grad$F(\boldsymbol{u}^k)$).

The method of the steepest descent is a key step in the CG method, but the choice of search directions $\boldsymbol{p}^k$ is not the optimal one. As $\boldsymbol{u}^{k+1}$ is optimized with respect to the previous search direction $\boldsymbol{p}^k = \boldsymbol{r}^k$, it is clear that the successive search directions are not orthogonal ($-\text{grad}F(\boldsymbol{u}^{k+1}) \perp \boldsymbol{p}^k$). It can be shown that $\boldsymbol{r}^k \perp \boldsymbol{r}^{k+1}$ and $\boldsymbol{r}^{k+1} \perp \boldsymbol{r}^{k+2}$, but it is general not true for $\boldsymbol{r}^k \perp \boldsymbol{r}^{k+2}$. Therefore, $\boldsymbol{u}^{k+1}$ has lost its optimum with respect to the previously optimized direction $\boldsymbol{r}^k$.

If $\boldsymbol{u}^{k+1}$ is optimal with respect to $\boldsymbol{p}^k \neq 0$, then this property is passed to $\boldsymbol{u}^{k+1}$ if and only if

$$A\boldsymbol{p}^{k+1} \perp \boldsymbol{p}^k \tag{2.37}$$

The vectors $\boldsymbol{p}^{k+1}$ and $\boldsymbol{p}^k$ are called *conjugate*. In the conjugate gradient method, the search directions are pairwise conjugate. Each time a new search direction is derived from the actual residual and conjugated with the prior search direction. It is also conjugate to all previous search directions. Thus a system of conjugate search directions is obtained or equivalent to a system of orthogonal residuals. This can be proven by induction. The initial values are defined as

$$\begin{aligned} \boldsymbol{r}^0 &= \boldsymbol{f} - \boldsymbol{A}\boldsymbol{u}^0 \\ \boldsymbol{p}^0 &= \boldsymbol{r}^0 \end{aligned} \tag{2.38}$$

The following equations describe the algorithm of the conjugate gradient method

$$\lambda_k = \frac{\boldsymbol{r}^{k^T}\boldsymbol{p}^k}{\boldsymbol{p}^{k^T}\boldsymbol{A}\boldsymbol{p}^k} \tag{2.39}$$

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \lambda_k \boldsymbol{p}^k \tag{2.40}$$

$$\boldsymbol{r}^{k+1} = \boldsymbol{r}^k - \lambda_k \boldsymbol{A}\boldsymbol{p}^k \tag{2.41}$$

$$\boldsymbol{p}^{k+1} = \boldsymbol{r}^k - \frac{\boldsymbol{r}^{k+1^T}\boldsymbol{A}\boldsymbol{p}^k}{\boldsymbol{p}^{k^T}\boldsymbol{A}\boldsymbol{p}^k}\boldsymbol{p}^k \tag{2.42}$$

As shown in **?**, for an efficient implementation, it is possible to use an alternative form for λ_k and p^{k+1}

$$\lambda_k = \frac{\boldsymbol{r}^{k^T}\boldsymbol{r}^k}{\boldsymbol{p}^{k^T}\boldsymbol{A}\boldsymbol{p}^k} \tag{2.43}$$

$$\boldsymbol{p}^{k+1} = \boldsymbol{r}^k + \frac{\boldsymbol{r}^{k+1^T}\boldsymbol{r}^k}{\boldsymbol{r}^{k^T}\boldsymbol{r}^k}\boldsymbol{p}^k \tag{2.44}$$

It can be proven that the CG method will converge to the exact solution after given finite steps. In theory, this method can achieve the same level of accuracy as a direct solver. However, due to numerical round-off errors, the orthogonality is often lost and such ideal theoretical results can not be achieved. In practice, given reasonable error tolerance, the CG method can generally be terminated after the convergence criteria have been met. This supports the view of the CG method as an iterative method, while an iterative method often would not converge to the exact solution, especially in theory. Thus the CG method is sometimes treated as a semi-iterative method.

2.4 Multigrid Method

The multigrid method is a scheme applied to solving a linear equation system with iterative solvers. It provides a convergence acceleration that improves the performance of these iterative solvers using grid coarsening **? ?**. In practice,

it can be implemented as a standalone method or as a preconditioner for other iterative methods such as the conjugate gradient method **?**. Various multigrid methods are used in different branches of applied mathematics and engineering. such as electromagnetics **?** and fluid dynamics **?**.

The two important components of multigrid methods are the restriction and prolongation operators which transfer information between fine grids and coarse grids. These operators are typically based on linear interpolation procedures and are connected through variational properties **?** to ensure optimal coarse-grid correction in the A^h-norm with A^h being the left-hand side of the linear system defined on the fine grid. The multigrid method can be also applied to the SBP-SAT method with specific grid transfer operators. In this section, we will provide a brief review of the multigrid method and its implementation. We consider the following steady-state problem:

$$Lu = f, \text{ in } \Omega \tag{2.45}$$

$$Hu = g, \text{ on } \partial\Omega \tag{2.46}$$

where L is a differential operator on domain Ω, and H is a boundary operator on the boundary $\partial\Omega$. This is a generalization of many linear systems with various boundary conditions.

2.4.0.1 The multigrid algorithm. In general, the construction of a multigrid consists of the following four basic steps:

1. Fine-grid discretization
2. Error smoothing
3. Coarse-grid correction

4. Fine-grid update

Different combinations of these steps result in different multigrid schemes. The most simple scheme is a two-level multigrid V cycle. We will expand these four steps in the following sections.

2.4.0.2 Fine-grid discretization. Consider a *fine grid* meshing Ω_1 on Ω. A discrete linear system associated to Equation 2.45 on this fine grid Ω_1 has the general form

$$L_1\mathbf{u} = \mathbf{F} \tag{2.47}$$

where L_1 is the discrete version of the operator L in Equation 2.45 which also include boundary conditions in Equation 2.46. The vector $\mathbf{F}$ approximates f on the grid points of Ω_1 which already incorporates data for the boundary condition**g** in Equation 2.46. $\mathbf{u}$ is the discretization of the solution u in the steady-state problem. We assume L_1 to be positive definite which also implies that L_1 is invertible. This is property is usually satisfied from discretization methods.

2.4.0.3 Error smoothing. Error smoothing is required prior to grid coarsening. Suppose we have an initial guess $\mathbf{u}^0$, the iterative approach towards the solution to Equation 2.47 is through solving

$$\mathbf{w}_\tau + L_1\mathbf{w}(\tau) = \mathbf{F},\ 0 < \tau < \Delta\tau \tag{2.48}$$

$$\mathbf{w}(0) = \mathbf{u}^{(0)} \tag{2.49}$$

where $\Delta_t > 0$ is the *smoothing step*. The solution to this equation is

$$\mathbf{w}(\Delta\tau) = e^{-L_1\Delta\tau}\mathbf{u}^0 + (I_1 - e^{-L_1\Delta\tau})L_1^{-1}\mathbf{F} \tag{2.50}$$

where I_1 is the identity matrix on Ω_1, and the following condition holds for any norm if L_1 is positive definite

$$||\mathbf{w}(\Delta\tau) - \mathbf{u}|| < ||\mathbf{u}^{(0)} - \mathbf{u}|| \tag{2.51}$$

Smoothing technique for the solution can be defined as follows

$$\begin{aligned} \mathbf{w}^k &= S\mathbf{w}^{k-1} + (I_1 - S)L_1^{-1}\mathbf{F}, k = 1, \ldots \nu \\ \mathbf{w}^0 &= \mathbf{u}^0 \end{aligned} \tag{2.52}$$

where S is the smoother. If S is an exponential smoother $S_{\exp} = e^{-L_1\Delta\tau}$, this will yield the pseudo time-marching procedure in Equation 2.48. This iterative method would converge after ν steps to

$$\mathbf{w} = S^\nu \mathbf{u}^0 + (I_1 - S^\nu)L_1^{-1}\mathbf{F} \tag{2.53}$$

The convergence criteria for this procedure is mentioned in the overview of iterative methods.

2.4.0.4 Coarse-grid correction. Next, consider the error $\mathbf{e} = L_1^{-1}\mathbf{F} - \mathbf{w}$ and the residual problem

$$L_1\mathbf{e} = \mathbf{F} - L_1\mathbf{w} \tag{2.54}$$

Instead of solving this system directly, we introduce a subset of Ω_1 called the *coarse grid* Ω_2, and solve the associated coarse grid problem on Ω_2

$$L_2\mathbf{d} = I_r(\mathbf{F} - L_1\mathbf{w}) \tag{2.55}$$

This problem is obtained from the finer grid problem Equation 2.54 by using the following operators

1. a restriction operator $I_r : \Omega_1 \to \Omega_2$
2. a coarse-grid operator L_2: $\Omega_2 \to \Omega_2$

The coarse-grid operator can be built by using the Galerkin condition

$$L_2 = I_r L_1 I_p \tag{2.56}$$

where $I_p : \Omega_2 \to \Omega_1$ is a the prolongation operator. In some situations, L_2 can be built independently through the direct use of discretization methods, but I_r and I_p needs to be carefully defined so the Galerkin condition Equation 2.56 still holds.

The prolongation operator I_p is commonly chosen through linear interpolation. Assume Ω_1 had a grid spacing of $\Delta x = 1/N$, and Ω_2 consists of the even grid points of Ω_1. This leads to

$$(I_p \mathbf{v})_m = \begin{cases} v_j, & m = 2j, j = 0, \ldots, N/2 \\ \frac{1}{2}(v_j + v_{j+1}), & m = 2j+1, j = 0, \ldots, N/2 \end{cases} \tag{2.57}$$

As we already define the prolongation operator, the restriction operator is given as

$$I_r = I_p^T / C \tag{2.58}$$

which is called the variational property. The C is a constant determined by the discretization method. In this problem, the value for C is 2.

2.4.0.5 Fine-grid update. Finally, we update the fine grid solution with correction $\mathbf{d}$ as

$$\mathbf{u}^{(1)} = \mathbf{w} + I_p \mathbf{d} \tag{2.59}$$

The relation Equation 2.59, together with Equation 2.53 and Equation 2.55 provides an iterative method for solving the steady-state problem

$$\mathbf{u}^{n+1} = M\mathbf{u}^n + N\mathbf{F} \tag{2.60}$$

where

$$M = CS^{\nu} \tag{2.61}$$

$$C = I_1 - I_p L_2^{-1} I_r L_1 \tag{2.62}$$

$$N = (I_1 - M) L_1^{-1} \tag{2.63}$$

M is called the multigrid iteration matrix here and C is referred to as the coarse grid correction operator. Here, M plays a central role in the convergence of the iterative method. We can see this by the definition of the error at step n $\mathbf{e}^{(n)} = \mathbf{u}^{(n)} - L_1^{-1}\mathbf{F}$ as we get

$$\mathbf{e}^{(n+1)} = M\mathbf{e}^{(n)} \tag{2.64}$$

which again leads to the same convergence criteria for the iterative method depending on the spectral radius of M.

To demonstrate the actual process of a multigrid scheme, we use the following two-grid correction scheme as an example

1. Relax ν_1 times on $L_1^h \mathbf{u}^{(1)} = \mathbf{F}^{(1)}$ on Ω_1 with the initial guess $\mathbf{v}^1$
2. Compute the fine-grid residual $\mathbf{r}^{(1)} = \mathbf{F}(1) - L_1\mathbf{v}^{(1)}$ and restrict it to the coarse grid by $\mathbf{r}^{(2)} = I_r\mathbf{r}^{(1)}$
3. Solve $L_2\mathbf{e}^{(2)} = \mathbf{r}^{(2)}$ (or relax ν_1 times) on Ω_2
4. Interpolate the coarse-grid error to the fine grid by $\mathbf{e}^{(1)} = I_p\mathbf{e}^{(2)}$ and correct the fine-grid approximation by $\mathbf{v}^1 \leftarrow \mathbf{v}^{(1)} + \mathbf{e}^{(1)}$
5. Relax ν_2 times on $L_1\mathbf{u}^{(1)} = \mathbf{F}^{(1)}$ on Ω_1 with the initial guess $\mathbf{v}^{(1)}$

There are more schemes for multigrid, and the main schemes are summarized inFigure 1.

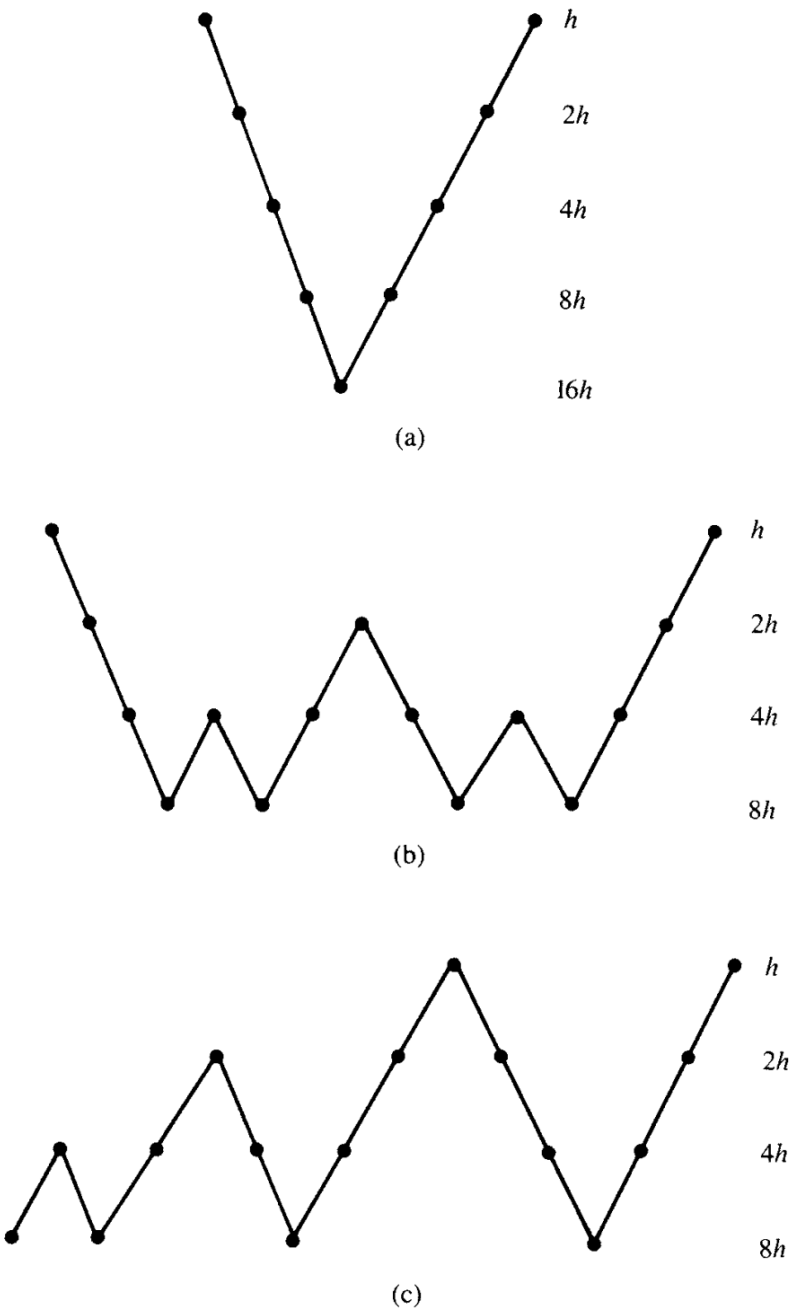

Figure 1. Schedule of grids for (a) V-cycle, (b) W-cycle, and (c) FMG scheme, all on four levels. **?**

Earlier work in multigrid relies on the geometric structure to construct coarse problems, thus this approach is called geometric multigrid. In problems where the computational domain is not composed of well-structured meshes, the multigrid method can be also applied via algebraic operators rather than a geometric grid. This approach is called the algebraic multigrid. We will cover this approach in the next subsection.

2.4.1 Algebraic Multigrid. The classical multigrid formed around the geometric structure has been generalized that the multigrid is analyzed in terms of the matrix properties **?**. This algebraic approach to theory was further extended to form the basis for much of the early development that led to the so-called Ruge-Stüben or classical algebraic multigrid (CAMG) method **???**. A detailed overview of the algebraic multigrid can be found in this recent paper **?**. Here, we want to present it more concisely. We begin this subsection with the following theorem in linear algebra.

Theorem 1 (Solvability and the Fundamental Theorem of Linear Algebra)**.** *Suppose we have a matrix $A \in \mathbf{R}^{m\times n}$. The fundamental theorem of linear algebra states that the range (column space) of the matrix, $\mathcal{R}(A)$, is equal to the orthogonal complement of $\mathcal{N}(A^T)$, the null space of A^T. Thus, spaces $\mathbf{R}^m$ and $\mathbf{R}^n$ can be orthogonally decomposed as follows:*

$$\mathbf{R}^m = \mathcal{R}(A) \oplus \mathcal{N}(A^T) \tag{2.65}$$

$$\mathbf{R}^n = \mathcal{R}(A) \oplus \mathcal{N}(A) \tag{2.66}$$

For the equation $A\mathbf{u} = \mathbf{f}$ to have a solution, it is necessary that the vector $\mathbf{f}$ lie in $\mathcal{R}(A)$. Thus, an equivalent condition is that $\mathbf{f}$ be orthogonal to every vector in $\mathcal{N}(A^T)$. For the equation $A\mathbf{u} = \mathbf{f}$ to have a unique solution, it is necessary that

$\mathcal{N}(A) = \{\mathbf{0}\}$. *Otherwise, if* $\mathbf{u}$ *is a solution and* $\mathbf{v} \in (A)$*, then* $A(\mathbf{u}+\mathbf{v}) = A\mathbf{u}+A\mathbf{v} = \mathbf{f}+\mathbf{0} = \mathbf{f}$*, so the solution is not unique*?.

This is another point to view the coarse-grid correction scheme, and this leads to the algebraic multigrid. More theories related to this topic and the spectral picture of multigrid can be found in ?.

The unique aspect of the CAMG is that the coarse problem is defined on a subset of the degrees of freedom of the initial problem, thus resulting in both coarse and fine points, which leads to the term CF-based AMG. A different approach to constructing algebraic multigrid is called *smoothed aggregation* AMG (SA), where collections of degrees-of-freedom define a coarse degree-of-freedom ?. Together, CF and SA form the basis of AMG and led to several developments that extend AMG to a wider class of problems and architectures.

AMG does not depend on the geostructure of the problem and discretization schemes, and due to this generalizability, it has been implemented in different forms in many software libraries. The original CAMG algorithm and its variants are available as `amg1r5` and `amg1r6` ?. A parallel implementation of the CF-based AMG can be found in the BoomerAMG package in the Hypre library Yang et al. (2002). The Trilinos package includes `ML` as a parallel SA-based AMG solver ?. Finally, PyAMG includes a number of AMG variants for testings, and Cusp distributes with a standard SA implementation for use on a GPU ??.

2.4.2 The Multigrid Method Within the SBP-SAT Scheme.

Since the SBP-SAT scheme is a framework for discretization to form a linear system, it is compatible with the multigrid method and can be accelerated using this technique. The key challenge from simply applying the common prolongation and restriction operators with the Galerkin condition Equation 2.56 is that the

summation-by-parts property would not be preserved for the coarse grid operators. In order to accurately represent the coarse-grid correction problem for the SBP-SAT scheme, a more suitable class of interpolation operators needs to be proposed. Many works have been done to address this issue **??**.

To overcome this issue, consider defining the restriction operator as

$$I_r = H_2^{-1} I_p^T H_1 \tag{2.67}$$

which was first introduced in **?**. This involves the coarse grid SBP norm H_2 and is obtained by enforcing that two scalar products

$$(\phi_1, \psi_1)_{H_1} = (\phi_1 H_1 \psi_1) \tag{2.68}$$

$$(\phi_2, \psi_2)_{H_2} = (\phi_2 H_2 \psi_2) \tag{2.69}$$

are equal for $\phi_1 = I_p \phi_2$ and $\psi_2 = I_r \psi_1$. ϕ and ψ correspond to the $\boldsymbol{u}$ and $\boldsymbol{v}$ in section **??**. We use these new notations to avoid confusion with the $\mathbf{u}$ used in the previous subsection on the multigrid method. As a result, the interpolation operators I_r and I_p are adjoints to each other with respect to the SBP-based scalar products defined in **?**.

$$(I_p, \xi_2, \xi_1)_{H_1} = (\xi_2, I_r \xi_1)_{H_2} \tag{2.70}$$

by using Equation 2.67, it is possible to build pairs of consistent and accurate prolongation and restriction operators. The following definition of the SBP-preserving interpolation operators was given in **?**, where the operators were used to couple SBP-SAT formulations on grids with different mesh sizes with numerical stability.

Definition 3. *Let the row-vectors $\mathbf{x}_1^k$ and $\mathbf{x}_2^k$ be the projections of the monomial x^k onto equidistant 1-D grids corresponding to a fine and coarse grid, respectively. I_r and I_p are then called 2q-th order accurate SBP-preserving interpolation operators*

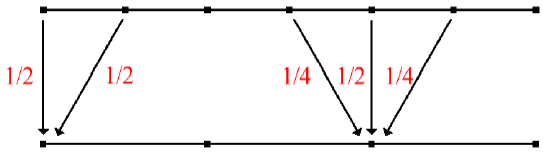

Figure 2. The 2nd-order SBP-preserving restriction operator I_r

if $I_r\mathbf{x}_1^k - \mathbf{x}_2^k$ *and* $I_p\mathbf{x}_2^k - \mathbf{x}_1^k$ *vanish for* $k = 0, ..., 2q-1$ *in the interior and for* $k = 0, ..., q-1$ *at the boundaries.*

The sum of the orders of the prolongation and restriction operators should be at least equal to the order of the differential equation. As a consequence, the use of high-order interpolation is not required here to solve the linear system with the multigrid method. However, high-order grid transfer operators can be used in combination with high-order discretization **?**.

SBP-preserving interpolation operators with minimal bandwidth are given in Appendix A. The restriction operator I_r, which differs from the conventional one at boundary nodes, is shown in Figure 2.

2.4.2.1 SBP-preserving interpolation applied to the first derivative.

Using Galerkin condition Equation 2.56 and SBP-preserving operators, we can construct the linear system with the multigrid method. We first consider the first derivative fine-grid SBP operator $D_{1,1}$ and its coarse-grid counterpart $D_{1,2}$ constructed as follow

$$D_{1,2} = I_r D_{1,1} I_p \tag{2.71}$$

We now show that $D_{1,2}$ preserves SBP property in such ways. To start with, we rewrite the left-hand side of the following SBP property

$$(\phi, D_1\psi)_H = \phi_N\psi_N - \phi_0\psi_0 - (D_1\phi, \psi)_H \tag{2.72}$$

with the adjoint relation Equation 2.70 as follows

$$\begin{aligned}(\phi_2, D_{1,2}\psi_2)_{H_2} &= (\phi_2, I_r(D_{1,1}I_p\phi_2))_{H_2} \\ &= (I_p\phi_2, D_{1,1}(I_p\psi_2))_{H_1}\end{aligned} \tag{2.73}$$

Next, the SBP property for the finite-grid operator D_1 leads to

$$\begin{aligned}(\phi_2, D_{1,2}\psi_2)_{H_2} &= (I_p, \phi_2)_N(I_p, \psi_2)_N \\ &\quad - (I_p, \phi_2)_0(I_p, \psi_2)_0 - (D_{1,1}(I_p\phi_2), I_p\psi_2)_{H_1}\end{aligned} \tag{2.74}$$

Both grids are conforming to the domain boundaries, and the prolongation onto the boundary nodes of the fine grid is exact. Furthermore, by applying Equation 2.70 to the right-hand side of Equation 2.74, we obtain

$$(\phi_2, D_{1,2}\psi_2)_{H_2} = \phi_{2,N/2}\psi_{2,N/2} - \phi_{2,0}\psi_{2,0} - (D_{1,2}\phi_2, \psi_2)_{H_2} \tag{2.75}$$

And we've shown that the coarse grid operator $D_{1,2}$ constructed in a such way preserves the SBP property. Also, the coarse grid first derivative SBP operator $D_{1,2}$ retains the order of accuracy of the original scheme at the interior nodes if $2q$th order SBP-preserving interpolations are used. The proof can be found in **?**.

2.4.2.2 SBP-preserving interpolation applied to the second derivative.

The SBP-preserving interpolation can also be applied to the second derivative operator. Similar to the proof for the first derivative operator, we can prove that the coarse grid operator constructed in such ways preserves the SBP property.

The interpolation operators in Equation 2.67 lead to a coarse-grid second derivative operator $D_{2,2}$ which preserves the summation-by-parts property **??**. We

can show that by rewriting the left hand side of the **??** for $D_{2,2}$ and the two coarse-grid functions ϕ_2 and ψ_2 by using Equation 2.70.

$$(\phi_2, D_{2,2}\psi_2)_{H_2} = (\phi_2, I_r, D_{2,1}I_p\psi_2)_{H_2} = (I_p\phi_2, D_{2,1}I_p\psi_2)_{H_1} \tag{2.76}$$

By applying the SBP property **??** for the fine-grid second derivative $D_{2,1}$, we have

$$\begin{aligned}(\phi_2, D_{2,2}\psi_2)_{H_2} &= (I_p\phi_2)_N(SI_p\psi_2)_N \\ &\quad - (I_p\phi_2)_0(SI_p\psi_2)_0 - (SI_p\phi_2)^T A(SI_p\psi_2)_{H_2}\end{aligned} \tag{2.77}$$

Both grids are conforming to domain boundaries, implying that $(I_p\phi_2)_i = \phi_{2,i/2}$ and $(SI_p\psi_2) = (S\phi_2)_{i/2}$ for $i \in \{0, N\}$. Thus

$$\begin{aligned}(\phi_2, D_{2,2}\psi_2)_{H_2} &= \phi_{2,N/2}(S\phi_2)_{N/2} \\ &\quad - \phi_{2,0}(S\phi_2)_0 - (SI_p\phi_2)^T A(SI_p\psi_2)_{H_2}\end{aligned} \tag{2.78}$$

where S is equivalent to $\boldsymbol{d}_0^T$ in **??** which approximates the first derivative at the boundaries.

Additional proofs or propositions to SBP-preserving interpolations can also be found in **?**. Furthermore, several model problems have been tested with multigrid iteration schemes using these SBP-preserving interpolations. These problems include a Poisson equation, the anisotropic elliptic problem, and the advection-diffusion problem. Numerical experiments show that the SBP-preserving interpolation improves convergence properties of the multigrid scheme for SBP-SAT discretizations regardless of the order of the discretization and smoother chosen. Moreover, the excellent performance in combination with the smoother SOR, clearly indicates that multigrid algorithms with SBP-preserving interpolation can be designed to get fast convergence. The paper mainly covers the steady model problem to compare the effect of different grid transfer operators. For time-

dependent problems, the effectiveness of multigrid algorithms with these SBP-preserving interpolations needs to be tested **?**.

2.5 Geometric multigrid for SBP-SAT method

Algorithm 1 $(k+1)$-level MG for $\boldsymbol{A}_h \boldsymbol{u}_h = \boldsymbol{f}_h$, with smoothing $S_{h_k}^{\nu}$ applied ν times. SBP-preserving restriction and interpolation operators are applied. Grid coarsening $(k \rightarrow k+1)$ is done through successive doubling of grid spacing until reaching the coarsest grid. The multigrid cycle can be performed $N_{maxiter}$ times. $\boldsymbol{r}$ represents the residual, and $\boldsymbol{v}$ represents the solution to the residual equation used during the correction step. This algorithm is adapted from Liu and Henshaw (2023).

function MG($\boldsymbol{f}_{h_k}$, $\boldsymbol{A}_{h_k}$, $\boldsymbol{u}_{h_k}^{(0)}$, k, $N_{maxiter}$)
  for $n = 0, 1, 2, \ldots, N_{maxiter}$ **do**
    $\boldsymbol{u}_{h_k}^{(n)} \xleftarrow{S_{h_k}^{\nu_1}} \boldsymbol{u}_{h_k}^{(n)}$ ▷ Pre-smoothing ν_1 times
    $\boldsymbol{r}_{h_k}^{(n)} = \boldsymbol{f}_{h_k}^{(n)} - \boldsymbol{A}_{h_k}^{(n)} \boldsymbol{u}_{h_k}^{(n)}$ ▷ Calculating residual
    $\tilde{\boldsymbol{r}}_{h_k} = (\boldsymbol{H}_k \otimes \boldsymbol{H}_k)^{-1} \boldsymbol{r}_{h_k}^{(n)}$ ▷ Removing grid info
    $\boldsymbol{r}_{h_{k+1}} = (\boldsymbol{H}_{k+1} \otimes \boldsymbol{H}_{k+1}) \boldsymbol{I}_{h_k}^{h_{k+1}} \tilde{\boldsymbol{r}}_{h_k}$ ▷ Restriction
    if $k+1 = k_{\max}$ **then**
      $\boldsymbol{v}_{h_{k+1}}^{(n)} \xleftarrow{S_{h_{k+1}}^{\nu_2}} \boldsymbol{0}_{h_{k+1}}$ ▷ Smoothing on coarsest grid
    else
      $\boldsymbol{v}_{h_{k+1}}^{(n)} = \mathrm{MG}(\boldsymbol{r}_{h_{k+1}}, \boldsymbol{A}_{h_{k+1}}, \boldsymbol{0}_{h_{k+1}}, k+1, 1)$ ▷ Recursive definition of MG
    end if
    $\boldsymbol{v}_k^n = \boldsymbol{I}_{h_{k+1}}^{h_k} \boldsymbol{v}_{k+1}^{(n)}$ ▷ Interpolation
    $\boldsymbol{u}_k^{(n+1)} = \boldsymbol{u}_k^{(n)} + \boldsymbol{v}_k^n$ ▷ Correction
    $\boldsymbol{u}_k^{(n+1)} \xleftarrow{S_{h_k}^{\nu_3}} \boldsymbol{u}_{h_k}^{(n+1)}$ ▷ Post-smoothing ν_3 times
  end for
end function

Algorithm 2 Matrix-Free GPU kernel Action of matrix-free A for interior nodes.

function mfA!(odata, idata, $c_{rr}, c_{rs}, c_{ss}, h_r, h_s$)
  i, j =get_global_thread_IDs()
  $g = (i-1)*(N+1)+j$ ▷ compute global index
  if $2 \le i, j \le N$ **then** ▷ interior nodes
    odata[g] = (hs/hr)(- $(0.5c_{rr}[g\text{-}1] + 0.5c_{rr}[g])$idata[$g$-1] +
      + $(0.5c_{rr}[g\text{-}1] + c_{rr}[g]$ - $0.5c_{rr}[g{+}1])$idata[g] +
      - $(0.5c_{rr}[g] + 0.5c_{rr}[g{+}1])$idata[$g$+1]) +
      ▷ compute M_{rr} stencil

      + $0.5c_{rs}[g\text{-}1](-0.5$idata[$g$-$N$-2] + 0.5idata[$g$+$N$]) +
      - $0.5c_{rs}[g{+}1](-0.5$idata[g-N] + 0.5idata[g+N+1]) +
      ▷ compute M_{rs} stencil

      + $0.5c_{rs}[g\text{-}N\text{-}1](-0.5$idata[$g$-$N$-2] + 0.5idata[$g$-$N$]) +
      - $0.5c_{rs}[g{+}N{+}1](-0.5$idata[$g$-$N$] + 0.5idata[$g$+$N$+2]) +
      ▷ compute M_{sr} stencil

      - $(0.5c_{ss}[g\text{-}N\text{-}1] + 0.5c_{ss}[g])$idata[$g$-$N$-1] +
      + $(0.5c_{ss}[g\text{-}N\text{-}1] + c_{ss}[g] + 0.5c_{ss}[g{+}N{+}1])$idata[$g$] -
      - $(0.5c_{ss}[g] + 0.5c_{ss}[g{+}N{+}1])$idata[$g$+$N$+1]))
      ▷ compute M_{ss} stencil
  end if
  ... ▷ boundary nodes, e.g. Algorithm 3
  return *nothing*
end function

Algorithm 3 Matrix-Free GPU kernel Action of matrix-free A for west boundary (face 1).

if $2 \leq i \leq N$ and $j = 1$ **then** ▷ interior west nodes
  odata$[g] = (M_{rr}^{int} + M_{rs}^{int} + M_{sr}^{int} + M_{sr}^{int} + C_1^{int})$ (idata)
    ▷ apply boundary M and C stencils
  odata$[g{+}1] = C_1^{int}$(idata) ▷ apply interior C stencil
  odata$[g{+}2] = C_1^{int}$(idata) ▷ apply interior C stencil
end if
if $i = 1$ and $j = 1$ **then** ▷ southwest corner node
  odata$[g] = (M_{rr}^{sw} + M_{rs}^{sw} + C_1^{sw})$ (idata)
    ▷ apply southwest partial M and C stencils
  odata$[g{+}1] = C_1^{sw}$(idata) ▷ apply southwest interior boundary C stencil
  odata$[g{+}2] = C_1^{sw}$(idata) ▷ apply southwest interior boundary C stencil
end if
if $i = N + 1$ and $j = 1$ **then** ▷ northwest corner node
  odata$[g] = (M_{rr}^{nw} + M_{rs}^{nw} + C^{nw})$ (idata)
    ▷ apply northwest partial M and C stencils
  odata$[g{+}1] = C^{nw}$(idata) ▷ apply northwest interior boundary C stencil
  odata$[g{+}2] = C^{nw}$(idata) ▷ apply northwest interior boundary C stencil
end if

CHAPTER III

SCIENTIFIC COMPUTING LIBRARIES AND LANGUAGES

3.1 PETSc

PETSc, which stands for Portable, Extensible Toolkit for Scientific Computation, is a software library developed primarily by Argonne National Library to facilitate the development of high-performance parallel numerical code written in C/C++, Fortran and Python. It provides a wide range of functionality for solving linear and nonlinear algebraic equations, ordinary and partial differential equations, and also optimization problems (provided by TAO) on parallel computing architectures. In addition, PETSc includes support for managing parallel PDE discretizations including parallel matrix and vector assembly routines.

Key features of PETSc include:

- Parallelism: PETSc is designed for parallel computing, especially distributed-memory parallel computing architectures. It is intended to run efficiently on parallel computing systems where multiple processors or nodes communicate over the network via a message passing interface (MPI). These architectures include clusters, supercomputers, and other HPC platforms.

- Modularity and Extensibility: PETSc is highly modular and extensible, allowing users to combine different numerical techniques and algorithms to solve complex problems efficiently. It provides a flexible framework for implementing new algorithms and incorporating external libraries. It mainly contains the following objects

    * Algebraic objects

        - Vectors (Vec) containers for simulation solutions, right-hand sides of linear systems
        - Matrices (Mat) containers for Jacobians and operators that define linear systems
    * Solvers
        - Linear solvers based on preconditioners (PC) and Krylov subspace methods (KSP)
        - Nonlinear solvers (SNES) that use data-structure-neutral implementations of Newton-like methods
        - Time integrators (TS) for ODE/PDE, explicit, implicit, IMEX
        - Optimization (TAO) with equality and inequality constraints, first and second order Newton methods
        - Eigenvalue/Eigenvectors (SLEPc) and related algorithms

– Efficiency and Performance: PETSc is optimized for performance, with algorithms and data structures designed to minimize memory usage and maximize computational efficiency. It supports parallel matrix and vector operations as well as efficient iterative solvers and preconditioners via the objects mentioned previously

– Flexibility: PETSc supports a wide range of numerical methods and algorithms and has built-in discretization tools. It provides interfaces for solving problems in various scientific and engineering disciplines, including computational fluid dynamics (CFD), solid mechanics, etc with documented examples and tutorials for researchers.

– PETSc is portable across different computing platforms and operating systems, including UNIX/Linux, macOS, and Windows. It provides a consistent interface and functionalities across different architectures, making it easy to develop and deploy simulation code across multiple platforms.

3.2 AmgX

AmgX is a proprietary software library developed by NVIDIA to accelerate the solution of large-scale linear systems arising from finite element and finite volume discretizations typically found in computational fluid dynamics (CFD) and computational mechanics simulations on NVIDIA GPUs. AmgX stands for Algebraic Multigrid Accelerated. It provides wrappers to work with other libraries like PETSc and programming languages like Julia.

Key features of AMGX include:

– Preconditioning: AmgX offers a variety of advanced preconditioning techniques, including algebraic multigrid (AMG), smoothed aggregation and hybrid methods to accelerate the convergence of iterative solvers for sparse linear systems. These preconditioners are designed for and tested in real-world engineering problems in collaboration with companies like ANSYS, a provider of leading CFD software Fluent.

– Parallelism: AmgX is optimized for NVIDIA GPUs and provides support for OpenMP to allow acceleration via heterogeneous computing and MPI to run large simulations across multiple GPUs and clusters.

– Flexibility and Customization: AmgX offers a flexible and extensible framework for configuring and customizing the solver algorithms via JSON files.

The limitation of AmgX is due to its link with NVIDIA. It can not run on GPUs from other vendors, such as AMD and Intel. Some of the latest exascale supercomputers are built with CPUs and GPUs from AMD and Intel.

3.3 HYPRE

HYPRE is a software library of high performance numerical algorithms including preconditioners and solvers for large, sparse linear systems of equations on massively parallel computers Falgout, Jones, and Yang (2006b). The HPYRE library was created to provide users with advanced parallel preconditioners. It features parallel multigrid solvers for both structured and unstructured grid problems. These solvers are called from application code via HYPRE's conceptual linear system interfaces Falgout, Jones, and Yang (2006a), which allow a variety of natural problem descriptions.

Key features of PETSc include:

- Scalable preconditioners: HYPRE contains several families of preconditioners focused on scalable solutions of very large linear systems. HYPRE includes "grey box" algorithms including structured multigrid that use more than just the matrix to solve certain classes of problems more efficiently.

- Common iterative methods: HYPRE provides several common Krylov-based iterative methods in conjunction with scalable preconditioners. This includes methods for symmetric matrices such as Conjugate Gradient (CG) and nonsymmetric matrices such as GMRES.

- Grid-centric interfaces for complicated data structures and advanced solvers: HYPRE has improved usability from earlier generations of sparse linear solver libraries in that users do not have to learn complicated sparse matrix data

structures. HYPRE builds these data structures for users through a variety of conceptual interfaces for different classes of users. These include stencil-based structured/semi-structured interfaces most suitable for finite difference methods, unstructured interfaces for finite element methods, and linear algebra based interfaces for general applications. Each conceptual interface provides access to several solvers without the need to manually write code for new interfaces.

- User options for beginners through experts: HYPRE allows users with various levels of expertise to write their code easily. The beginner users can set up runnable code with a minimal amount of effort. Expert users can take further control of the solution process through various parameters

- Configuration options for different platforms: HYPRE allows a simple and flexible installation on various computing platforms. Users have options to configure for different platforms during the installation. Additional options include debug mode which offers more info and optimized mode for better performance. It also allows users to change different libraries such as MPI and BLAS.

- Interfaces to multiple languages: HYPRE is written in C, but it also provides an interface for Fortran users.

3.4 Review of several languages for scientific computing

3.4.1 Fortran. There are many languages designed for high performance computing. Traditionally, Fortran has been used to write high performance numerical code. It is short for "Formula Translation". As the name suggest, it is one of the oldest and most enduring programming languages in

scientific computing. Developed in the 1950s by IBM, it was designed to facilitate numerical and scientific computations, particularly for high-performance computing on mainframe computers.

Fortran was specifically designed for efficient numerical and scientific computing, with optimized operations handling mathematical operations, arrays, and complex computations. It provides a rich set of built-in functions and libraries for numerical analysis, linear algebra, differential equations, and other mathematical tasks. It is a statically typed language, meaning that variable types are declared explicitly at compile time and do not change during runtime. This allows compilers to perform extensive type checking and optimization to generate efficient code for execution.

Fortran codes are also highly portable across different computing platforms. While early versions of Fortran (66, 77) were designed for specific hardware architectures, modern Fortran standards, such as Fortran 90, 95, 2003, 2008, and 2018 (formerly 2015) have introduced features that enhance portability and interoperability with other languages and systems. Fortran is also known for its excellent backward compatibility, with newer language standards preserving compatibility with older databases. This allows legacy Fortran programs to continue running without modification on modern compilers and systems, ensuring long-term viability and support for existing applications, which is very important in scientific research where many simulation codes are built on top of decades of previous work.

Because of these reasons, despite its age, Fortran remains widely used in scientific and engineering computing.

3.4.2 C and C++.

C was created in 1972 as a general-purpose programming language. C++ was created in 1979 to enhance C language with object-oriented design and many useful standard template libraries. Despite the historical dominance of Fortran in scientific and engineering computing, C and C++ have gradually replaced Fortran in many scientific computing and HPC codes due to their performance, flexibility, and rich ecosystem of tools and libraries.

While Fortran continues to be used in certain domains, particularly in legacy codebases and specialized applications, the adoption of C and C++ as the default language in many modern packages reflects the evolving needs and preferences of HPC developers for modern, versatile programming languages.

C and C++ are known for their performance and efficiency. In fact, they are often used as the standard to compare the performance of various programming languages. This is because they provide low-level control over hardware resources and memory management, allowing programmers to write code that executes with high speed and minimal overhead. The performance is crucial for HPC applications, which often involve computationally extensive tasks and large-scale simulations. Known as high-level languages, C and C++ strike a balance between high-level abstractions and low-level control. They support multiple programming paradigms including procedural, and object-oriented. C/C++ can also be extended to handle parallel processing via pragma directives. This allows the creation of modular, reusable code with encapsulation, inheritance, polymorphism, and templates. Standard Libraries built on top of these features provide implementations of fundamental data structures, algorithms, and utilities.

In addition to their language features, C and C++ offer support for concurrency and parallelism via low-level features like threads, mutexes, condition

variables, atomic operations, and parallel algorithms. Modern C++ standards (such as C++11, C++17 and C++20) have introduced high-level features to manage asynchronous execution, parallel computation, and parallelism-aware data structures. All these efforts further enhance the capability of C and C++ as high performance computing languages.

As general-purpose programming languages, C and C++ codes are highly portable across different platforms and architectures. The portability is essential for deploying HPC applications on diverse computing platforms, including cloud servers, clusters, and supercomputers. C and C++ also have excellent interoperability with other programming languages and systems. They can be easily integrated with libraries and tools including most common HPC languages like Fortran, Python, and CUDA. This interoperability allows developers to leverage existing software components and take advantage of specialized and optimized libraries for specific computational tasks. However, the impact on the performance needs to be considered carefully when interoperating C and C++ with other languages.

3.4.3 MATLAB. MATLAB is a high-level programming language usually used in an interactive development environment (IDE) from the software with the same name. Developed by MathWorks, it is widely used for numerical computing, data analysis, visualization, and algorithm development. Compared to compiled languages that can generate binary executables running natively on operation systems, MATLAB requires an interpreter (usually by MATLAB) to "translate" the code whenever the code is run. To avoid ambiguity, we refer to both the language and the IDE as MATLAB here. As a proprietary language and tool, MATLAB offers limited access to the source code, and it is prohibitively expensive

for people outside of academia without an educational license. GNU Octave is used as an open source alternative to MATLAB as it is mostly compatible with MATLAB. Octave is free and lightweight, however, it often comes with the cost of worse performance. Despite being a proprietary software, MATLAB is still often used in scientific computing, especially in academia for the following reasons:

MATLAB is easy to use because of its intuitive syntax for mathematicians and comprehensive set of built-in functions for numerical computing, including matrix manipulation, linear algebra, and optimizations. For these functions, MATLAB offers extensive examples and tutorials, making it a great choice for beginners for learning and advanced users for writing code.

MATLAB has an interactive environment with visualization tools that enable users to iterate quickly on algorithms. It offers a command-line interface that is similar to read-evaluate-print-loop (REPL) in interpreted languages like Python, and also integrates many common functionalities via UI buttons in its IDE. Like many IDEs, MATLAB provides tools for organizing code, debugging, profiling, and version control. More importantly, MATLAB's functionality can be extended through its proprietary and third-party toolboxes, which are collections of specialized functions and algorithms for specific domains of applications such as signal processing, control theory, and statistics.

Because of these features and accessibility via academic licenses through educational institutes, many people start numerical coding in MATLAB and continue to develop in MATLAB for research purposes. Although MATLAB is designed to run numerical calculations efficiently and also provides some limited support for parallel and GPU computing, it was not designed as a HPC language running on clusters, supercomputers, and cloud infrastructures. Researchers often

use MATLAB for quick implementation and testing during the prototyping stage and then rewrite their code in HPC languages such as FORTRAN and C/C++. This raises the so-called "two-language" problem which inspires the development of the Julia language.

3.5 Julia langauge

CHAPTER IV

PROBLEMS AND FORMULATIONS

CHAPTER V

RESULTS

5.1 Chapter Three Section One

5.1.1 Chapter three section one sub-section one.

5.1.1.1 Chapter three section one sub-section one sub-sub-section one.

CHAPTER VI

CONCLUSION

6.1 Chapter Four Section One

6.1.1 Chapter four section one sub-section one.

6.1.1.1 Chapter four section one sub-section one sub-sub-section one.

This is a sample citation: Schwartz (2012).

REFERENCES CITED

Avouac, J.-P. (2015). From geodetic imaging of seismic and aseismic fault slip to dynamic modeling of the seismic cycle [Journal Article]. *Annual Review of Earth and Planetary Sciences*, *43*(Volume 43, 2015), 233-271. Retrieved from https://www.annualreviews.org/content/journals/10.1146/annurev-earth-060614-105302 doi: https://doi.org/10.1146/annurev-earth-060614-105302

Balay, S., Abhyankar, S., Adams, M. F., Benson, S., Brown, J., Brune, P., ... Zhang, J. (2023). *PETSc Web page.* https://petsc.org/. Retrieved from https://petsc.org/

Carpenter, M. H., Gottlieb, D., & Abarbanel, S. (1994). Time-stable boundary conditions for finite-difference schemes solving hyperbolic systems: Methodology and application to high-order compact schemes. *Journal of Computational Physics*, *111*(2), 220–236. doi: 10.1006/jcph.1994.1057

Del Rey Fernández, D. C., Hicken, J. E., & Zingg, D. W. (2014). Review of summation-by-parts operators with simultaneous approximation terms for the numerical solution of partial differential equations. *Computers & Fluids*, *95*, 171-196.

Dieterich, J. H. (1979a). Modeling of rock friction: 1. experimental results and constitutive equations. *Journal of Geophysical Research: Solid Earth*, *84*(B5), 2161-2168. Retrieved from https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/JB084iB05p02161 doi: https://doi.org/10.1029/JB084iB05p02161

Dieterich, J. H. (1979b). Modeling of rock friction: 2. simulation of preseismic slip. *Journal of Geophysical Research: Solid Earth*, *84*(B5), 2169-2175. Retrieved from https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/JB084iB05p02169 doi: https://doi.org/10.1029/JB084iB05p02169

Erickson, B. A., & Day, S. M. (2016). Bimaterial effects in an earthquake cycle model using rate-and-state friction. *Journal of Geophysical Research: Solid Earth*, *121*, 2480–2506. doi: 10.1002/2015JB012470

Erickson, B. A., & Dunham, E. M. (2014). An efficient numerical method for earthquake cycles in heterogeneous media: Alternating subbasin and surface-rupturing events on faults crossing a sedimentary basin. *Journal of Geophysical Research: Solid Earth*, *119*(4), 3290–3316. doi: 10.1002/2013JB010614

Erickson, B. A., Kozdon, J. E., & Harvey, T. (2022). A non-stiff summation-by-parts finite difference method for the scalar wave equation in second order form: Characteristic boundary conditions and nonlinear interfaces. *Journal of Scientific Computing*, *93*(1), 17.

Falgout, R. D., Jones, J. E., & Yang, U. M. (2006a, jan). Conceptual interfaces in hypre. *Future Gener. Comput. Syst.*, *22*(1), 239–251.

Falgout, R. D., Jones, J. E., & Yang, U. M. (2006b). The design and implementation of hypre, a library of parallel high performance preconditioners.. Retrieved from `https://api.semanticscholar.org/CorpusID:3237430`

Kozdon, J. E., Dunham, E. M., & Nordström, J. (2012). Interaction of waves with frictional interfaces using summation-by-parts difference operators: Weak enforcement of nonlinear boundary conditions. *Journal of Scientific Computing*, *50*, 341-367. doi: 10.1007/s10915-011-9485-3

Kozdon, J. E., Erickson, B. A., & Wilcox, L. C. (2020). Hybridized summation-by-parts finite difference methods. *Journal of Scientific Computing*. doi: 10.1007/s10915-021-01448-5

Kreiss, H., & Scherer, G. (1974). Finite element and finite difference methods for hyperbolic partial differential equations. In *Mathematical aspects of finite elements in partial differential equations; proceedings of the symposium* (pp. 195–212). Madison, WI. doi: 10.1016/b978-0-12-208350-1.50012-1

Liu, C., & Henshaw, W. (2023). Multigrid with nonstandard coarse-level operators and coarsening factors. *Journal of Scientific Computing*, *94*(3), 58.

Lotto, G. C., & Dunham, E. M. (2015). High-order finite difference modeling of tsunami generation in a compressible ocean from offshore earthquakes. *Computational Geosciences*, *19*(2), 327–340. doi: 10.1007/s10596-015-9472-0

Marone, C. (1998). Laboratory-derived friction laws and their application to seismic faulting [Journal Article]. *Annual Review of Earth and Planetary Sciences*, *26*(Volume 26, 1998), 643-696. Retrieved from `https://www.annualreviews.org/content/journals/10.1146/annurev.earth.26.1.643` doi: https://doi.org/10.1146/annurev.earth.26.1.643

Mattsson, K. (2003, Feb 01). Boundary procedures for summation-by-parts operators. *Journal of Scientific Computing*, *18*(1), 133-153.

Mattsson, K., Ham, F., & Iaccarino, G. (2009). Stable boundary treatment for the wave equation on second-order form. *Journal of Scientific Computing*, *41*(3), 366–383. doi: 10.1007/s10915-009-9305-1

Mattsson, K., & Nordström, J. (2004). Summation by parts operators for finite difference approximations of second derivatives. *Journal of Computational Physics*, *199*(2), 503–540. doi: 10.1016/j.jcp.2004.03.001

Nordström, J., & Eriksson, S. (2010). Fluid structure interaction problems: The necessity of a well posed, stable and accurate formulation. *Communications in Computational Physics*, *8*(5), 1111–1138. doi: https://doi.org/10.4208/cicp.260409.120210a

Petersson, N. A., & Sjögreen, B. (2012). Stable and efficient modeling of anelastic attenuation in seismic wave propagation. *Communications in Computational Physics*, *12*(1), 193–225. doi: 10.4208/cicp.201010.090611a

Roten, D., Cui, Y., Olsen, K. B., Day, S. M., Withers, K., Savran, W. H., ... Mu, D. (2016). High-frequency nonlinear earthquake simulations on petascale heterogeneous supercomputers. In *Sc '16: Proceedings of the international conference for high performance computing, networking, storage and analysis* (p. 957-968). doi: 10.1109/SC.2016.81

Ruina, A. (1983). Slip instability and state variable friction laws. *Journal of Geophysical Research: Solid Earth*, *88*(B12), 10359-10370. Retrieved from `https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/JB088iB12p10359` doi: https://doi.org/10.1029/JB088iB12p10359

Saad, Y. (2003). *Iterative methods for sparse linear systems* (Second ed.). Society for Industrial and Applied Mathematics. Retrieved from `https://epubs.siam.org/doi/abs/10.1137/1.9780898718003` doi: 10.1137/1.9780898718003

Schwartz, S. H. (2012). An overview of the Schwartz theory of basic values. *Online Readings in Psychology and Culture*, *2*(1), 1–20.

Strand, B. (1994). Summation by parts for finite difference approximations for d/dx. *Journal of Computational Physics*, *110*(1), 47–67. doi: 10.1006/jcph.1994.1005

Svärd, M., & Nordström, J. (2014). Review of summation-by-parts schemes for initial-boundary-value problems. *Journal of Computational Physics*, *268*, 17-38. doi: https://doi.org/10.1016/j.jcp.2014.02.031

Swim, E., Benra, F.-K., Dohmen, H. J., Pei, J., Schuster, S., & Wan, B. (2011). A comparison of one-way and two-way coupling methods for numerical analysis of fluid-structure interactions. *Journal of Applied Mathematics*, *2011*, 1–16. doi: 10.1155/2011/853560

Yang, U. M., et al. (2002). BoomerAMG: A parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, *41*(1), 155–177.

Ying, W., & Henriquez, C. (2007). Hybrid finite element method for describing the electrical response of biological cells to applied fields. *IEEE Transactions on Bio-medical Engineering*, *54*(4), 611–620. doi: 10.1109/TBME.2006.889172